Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Оренбургский государственный университет»

# РАЗРАБОТКА ПРИКЛАДНЫХ БИБЛИОТЕК ДЛЯ САПР КОМПАС-ГРАФИК НА ЯЗЫКЕ C++

Учебное пособие

Рекомендовано ученым советом федерального государственного бюджетного образовательного учреждения высшего образования «Оренбургский государственный университет» для обучающихся по образовательным программам высшего образования по направлениям подготовки 09.03.01 Информатика и вычислительная техника, 15.03.04 Автоматизация технологических процессов и производств.

Оренбург
2021

УДК 658.512.26:004.925.8
ББК 30.2-5-05
Р17

Рецензент – доцент, доктор технических наук А.С. Боровский

Авторы: А.И. Сергеев, С.Ю. Шамаев, А.С. Русяев, М.В. Овечкин

Р17 **Разработка прикладных библиотек для САПР КОМПАС-График на языке С++ [Электронный ресурс] : учебное пособие для обучающихся по образовательным программам высшего образования по направлениям подготовки 09.03.01 Информатика и вычислительная техника, 15.03.04 Автоматизация технологических процессов и производств / А. И. Сергеев [и др.]; М-во науки и высш. образования Рос. Федерации, Федер. гос. бюджет. образоват. учреждение высш. образования "Оренбург. гос. ун-т". - Оренбург : ОГУ. - 2021. - 152 с- Загл. с тит. экрана.**

ISBN

Учебное пособие содержит теоретические сведения о разработке прикладных программных модулей для системы автоматизированного проектиро-вания «КОМПАС- График». Представлен материал о настройке среды про-граммирования, об основных функциях КОМПАС МАСТЕР. В пособие вклю-чены задания для выполнения практических работ. Для самоподготовки по каждой теме приводятся контрольные вопросы.

Учебное пособие предназначено для обучающихся по образовательным программам высшего образования по направлениям подготовки 09.03.01 Ин-форматика и вычислительная техника, 15.03.04 Автоматизация технологиче-ских процессов и производств. Издание может быть полезно и для студентов других направлений подготовки при изучении дисциплин, связанных с разра-боткой подсистем автоматизированного проектирования.

УДК 658.512.26:004.925.8
ББК 30.2-5-05



ISBN

# Содержание

# Введение

Каждое предприятие сталкивается со специфическими задачами при проектировании изделий, поэтому почти во всех САПР присутствует механизм, с помощью которого пользователь может разрабатывать собственные встраиваемые в систему прикладные программные модули, решающие специализированные отраслевые задачи. Программный пакет КОМПАС - это САПР общего назначения, предназначенная для автоматизации подготовки чертежей и для решения задач трехмерного моделирования. КОМПАС является открытой системой, на базе которой возможна разработка специализированных прикладных САПР и расширение функциональных возможностей этой системы с помощью модулей, реализованных независимыми разработчиками. Для этого предназначены библиотеки КОМПАС-МАСТЕР, представляющих собой интерфейс прикладного программирования для доступа к возможностям КОМПАС из внешних программ, написанных на различных языках программирования.

Данное пособие предназначено для изучения основных функциональных возможностей КОМПАС-МАСТЕР, доступных из программ на языке Си++ посредством технологии экспортных функций. Предполагается, что после изучения материала обучающийся сможет самостоятельно выполнять разработку библиотек для КОМПАС-График.

Текст работ рассчитан на чтение и воспроизведение описанных в работах действий с последующим выполнением заданий. Приведенные фрагменты кода проверены в версии КОМПАС v19. Содержание пособия основано на примерах, распространяемых вместе с КОМПАС-МАСТЕР и на методических материалах по программированию с использованием API (Application Programming Interface - прикладной программный интерфейс) системы КОМПАС [1], представленных на сайте АСКОН.

# 1 Инструментальные средства разработки прикладных САПР

## 1.1 Способы расширения функционала пакетов прикладных программ

Широкое распространение персональных компьютеров привело к появлению пакетов прикладных программ, предназначенных для решения задач в определенной предметной области и рассчитанных на массового пользователя.

Примерами таких пакетов являются:

– MS Office для автоматизации офисного делопроизводства (рисунок 1.1, а);

– 1С:Бухгалтерия для компьютеризации бухгалтерского учета (рисунок 1.1, б);

– КОМПАС-3D для автоматизации подготовки конструкторской документации (рисунок 1.1, в).


а

б


в

а – MS Office; б – 1С:Бухгалтерия; в – КОМПАС-3D

Рисунок 1.1 – Пакеты прикладных программ, предназначенные для решения задач в определенной предметной области

Количество пользователей подобных пакетов очень велико, и это вызывает проблему перегрузки программ функциями, которые могут быть и не нужны многим пользователям. С другой стороны, у отдельных групп пользователей возникают специфические требования, которые разработчикам основного программного пакета не удается своевременно учесть или они не представляют большого интереса для большинства других пользователей.

По этим причинам в программах массового распространения часто содержатся средства для расширения их функциональности силами самих пользователей, хотя, конечно, в большинстве случаев реалистичнее полагать, что это расширение будут проводить по запросам пользователей грамотные специалисты, возможно, профессиональные программисты.

### 1.1.1 Макросы

В качестве простейшего примера средств расширения функционала программного пакета можно назвать макросы (например, в MS Office) – средства, позволяющие запомнить часто повторяющуюся последовательность команд пакета и затем воспроизводить ее всего одной новой командой.

Макросы в MS Office в интерактивном режиме записываются на языке Visual Basic for Applications (VBA). Этот язык, кроме макросов, позволяет разрабатывать отдельные функции и целые приложения, выполняемые в среде Word, Excel, Access или PowerPoint. Для программиста язык VBA выглядит как интерпретируемый язык Basic со средой разработки, вызываемой изнутри прикладной программы (например, Word) и со специфической библиотекой функций и объектов, предоставляющих доступ к данным и командам конкретной программы пакета MS Office. Благодаря массовому распространению пакета MS Office применение языка VBA стало одним из наиболее известных способов расширения программных пакетов.

КОМПАС-Макро – это интегрированная в систему КОМПАС-3D среда разработки конструкторских приложений на основе языка программирования Python.

Python распространяется бесплатно и, как следствие, нет никаких ограничений на использование программ, написанных на нем.

На сегодняшний день Python – один из самых простых и понятных языков программирования. И при всей своей простоте он мало в чем уступает таким «китам» объектно-ориентированного программирования, как C++ или Delphi.

По сути, КОМПАС-Макро является обычной библиотекой, подключаемой к КОМПАС, только с очень большими возможностями. После установки среды Python и КОМПАС-Макро (их дистрибутивы входят в установочный комплект системы КОМПАС) библиотеку можно подключить к системе как обычный прикладной модуль – с помощью менеджера библиотек.

### 1.1.2 Использование специализированных языков программирования

Другим вариантом расширения программных пакетов является использование специализированных языков программирования.

Это может быть целесообразно, когда требуется работать со сложными структурами данных и объемными исходными текстами, для разработки которых язык VBA не слишком удобен. Данный подход применяется в известном бухгалтерском пакете 1С:Бухгалтерия, и, вообще, в семействе программ 1С:Предприятие. В этих пакетах есть поддержка собственного языка программирования, на котором программист может написать функции, настраивающие эти программы для нужд конкретного предприятия.

В КОМПАС-3D, например, интегрирована система КОМПАС-Макро – среда разработки конструкторских приложений на основе языка программирования Python.

### 1.1.3 Наращивания функциональности пакета – разработка дополнительных модулей

Распространен еще один способ наращивания функциональности пакета – разработка дополнительных модулей (plug-in) на компилируемых языках программирования общего назначения, таких, как Delphi, C# или C++.

Известные примеры реализации этого подхода – разработка модулей обработки изображений для графических программ типа Adobe Photoshop, расчетных модулей для Autodesk 3Ds MAX и других. Каждый дополнительный модуль можно считать библиотекой с одной или несколькими функциями, которые пользователь может вызывать из среды конкретного пакета (базового пакета). Из модуля можно обращаться к базовому пакету, обмениваться с ним данными, согласованно показывать какие-либо диалоговые окна, «встраиваться» в интерфейс пользователя базового пакета.

Таким образом, можно выделить три основных способа расширения функциональности программных пакетов:

1) макросы;

2) специфический язык программирования;

3) дополнительные модули.

Самый распространенный пример расширения возможностей КОМПАС – создание конструкторских библиотек для автоматизации действий конструктора. Эти библиотеки расширения КОМПАС называются прикладными библиотеками. Их можно рассматривать как приложения, функционирующие «в среде КОМПАС».

Прикладная библиотека для КОМПАС является динамической библиотекой (DLL - Dynamic Link Library) для ОС Windows. Но эта DLL должна быть написана в соответствии с требованиями, предъявляемым к прикладным библиотекам КОМПАС, а именно, содержать в себе несколько функций с предопределенными именами, которые КОМПАС будет вызывать у этой библиотеки. Обычно прикладные библиотеки КОМПАС имеют расширение RTW, а не DLL, чтобы их можно было

отличать от обычных DLL по именам. Поэтому, далее будем называть прикладные библиотеки RTW-библиотеками.

### 1.2 Настройка проекта MS Visual Studio для работы с КОМПАС

Перед созданием проекта необходимо убедиться, что в Visual Studio установлены необходимые для этого компоненты: MFC библиотека C++ (рисунок 1.2). Тип создаваемого проекта должен быть 64-битным (рисунок 1.3).

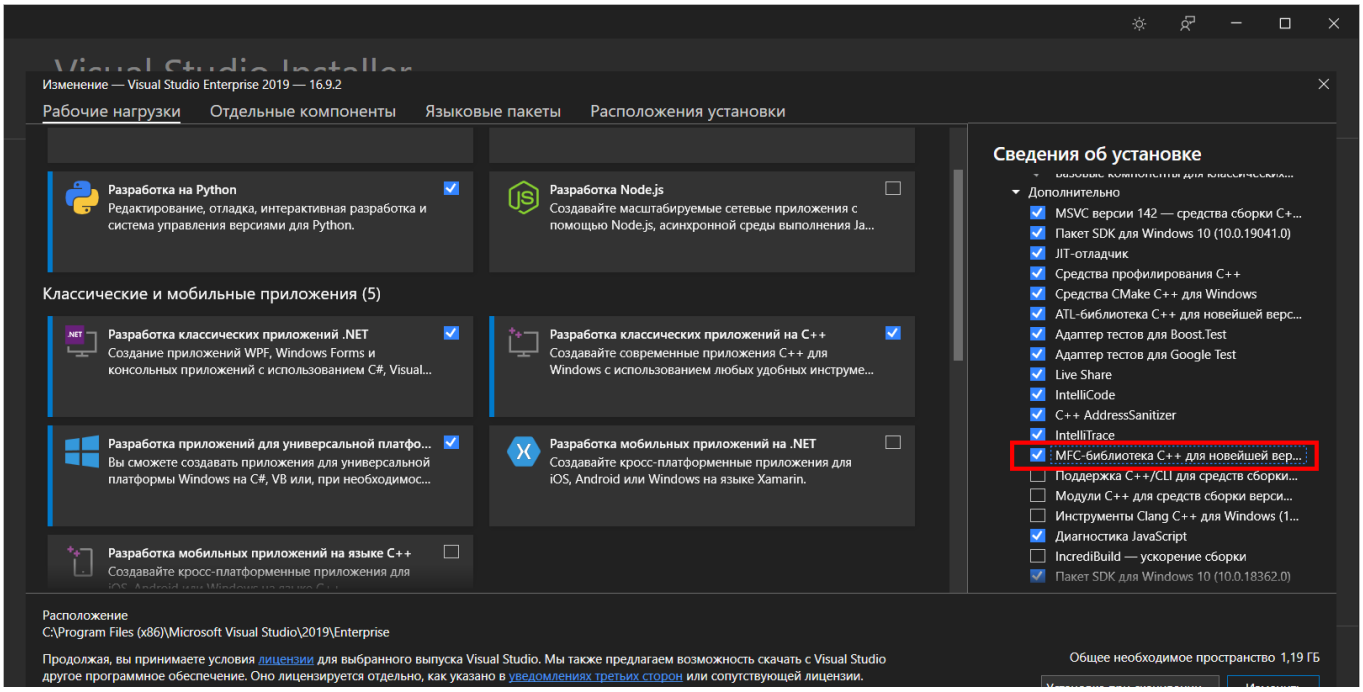



Рисунок 1.2 – Выбор компонентов MFC библиотеки C++

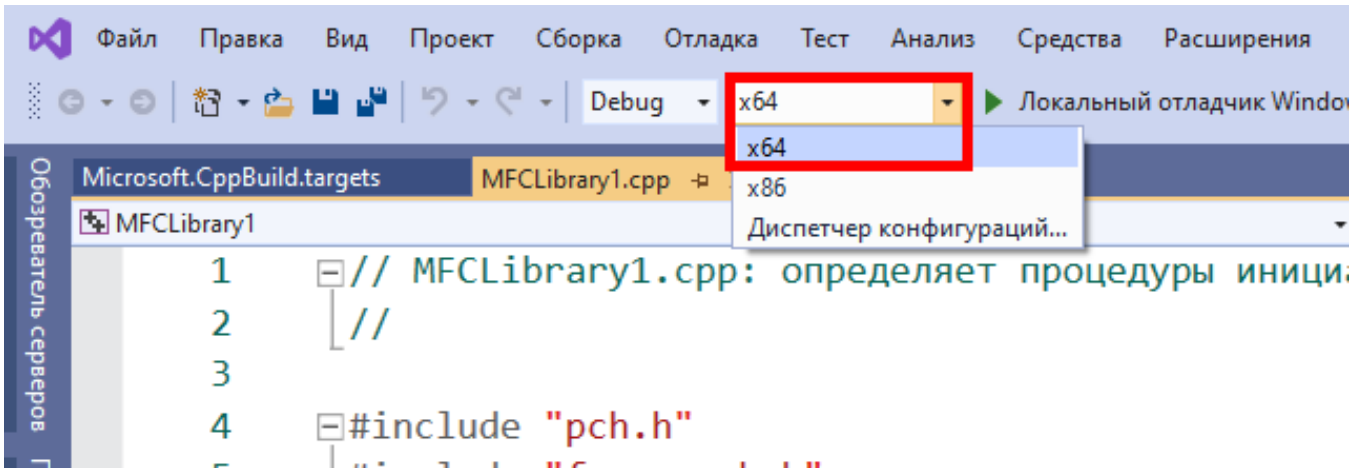


Рисунок 1.3 – Выбор типа проекта

Рассмотрим процесс создания прикладной библиотеки для САПР КОМПАС в Visual Studio на языке C++. Для этого необходимо выполнить следующие действия.

Шаг 1. Вызвать из меню Файл → Создать → Проект.

Шаг 2. Для Visual Studio 2017 выбрать пункт Visual C++, затем MFC/ATL, далее выбрать из списка вариант Библиотека динамической компоновки MFC (рисунок 1.4). Задать имя проекта и каталог расположения исходных файлов и нажать кнопку ОК.

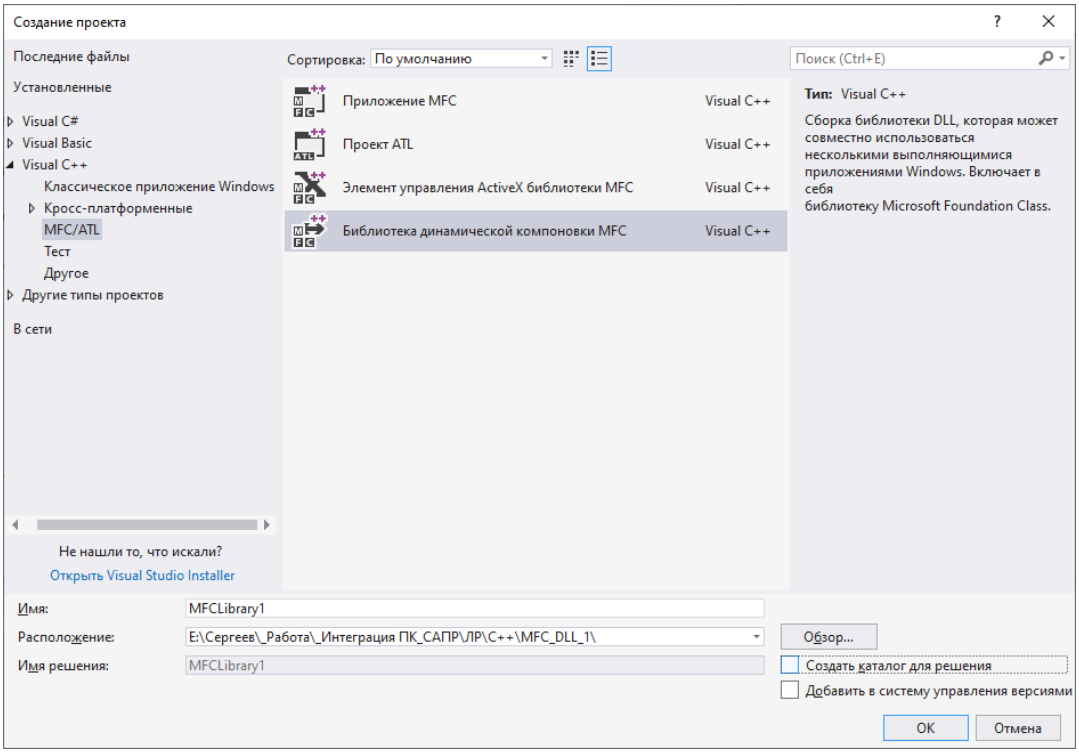



Рисунок 1.4 – Создание прикладной библиотеки в Visual Studio 2017

Для Visual Studio 2019 необходимо настроить параметры поиска соответствующего шаблона: язык C++, платформа Windows, тип проекта Библиотека, выбрать из списка вариант Библиотека динамической компоновки MFC (рисунок 1.5), нажать кнопку Далее, в следующем окне указать имя и расположение библиотеки, нажать Создать.

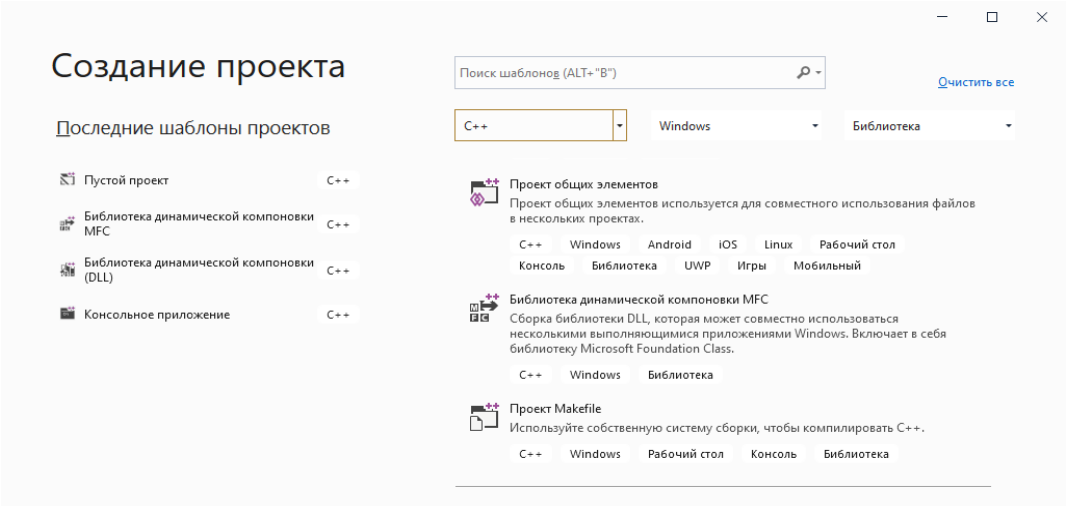



Рисунок 1.5 – Создание прикладной библиотеки в Visual Studio 2019

Шаг 3. В появившемся на экране диалоге Библиотека DLL MFC выбрать в группе Тип DLL вариант Обычная библиотека DLL с использованием общей библиотеки DLL MFC. Другие настройки этого диалога оставить без изменения и нажать кнопку ОК (рисунок 1.6).

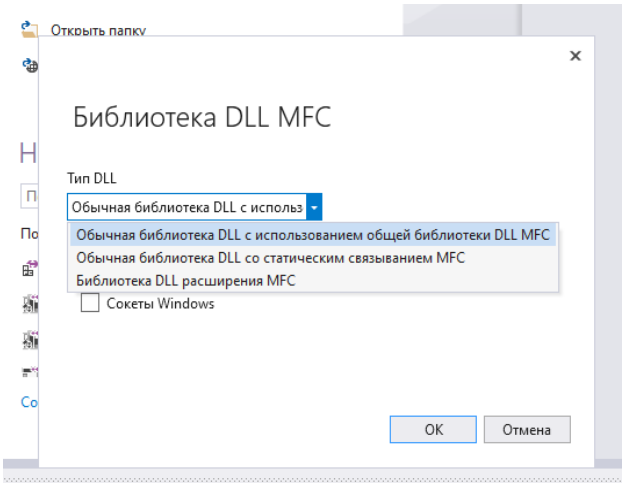



Рисунок 1.6 – Выбор типа библиотеки

Будет создан проект прикладной библиотеки, исходный код которой приведен в листинге 1.1 (часть комментариев удалена).

Листинг 1.1 – Исходный код созданной библиотеки

```
#include "pch.h"
#include "framework.h"
#include "MFCLibrary1.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#endif
// CMFCLibrary1App
BEGIN_MESSAGE_MAP(CMFCLibrary1App, CWinApp)
END_MESSAGE_MAP()
// Создание CMFCLibrary1App
CMFCLibrary1App::CMFCLibrary1App() {
	// TODO: добавьте код создания,
	// Размещает весь важный код инициализации в InitInstance
}
// Единственный объект CMFCLibrary1App
CMFCLibrary1App theApp;
// Инициализация CMFCLibrary1App
BOOL CMFCLibrary1App::InitInstance() {
	CWinApp::InitInstance();
	return TRUE;
}
```

Шаг 4. Теперь необходимо настроить проект, для того чтобы он корректно компилировался с учетом взаимодействия с API КОМПАС. Для этого необходимо выбрать пункт меню Проект → Свойства:… В открывшемся диалоговом окне для Visual Studio 2017 нужно в разделе Свойства конфигурации → Общие (рисунок 1.7) изменить параметр Конечное расширение с .dll на .rtw, чтобы создаваемая библиотека автоматически получила расширение .rtw. Для Visual Studio 2019 данные настройки будут находиться в разделе Свойства конфигурации → Дополнительно (рисунок 1.8).

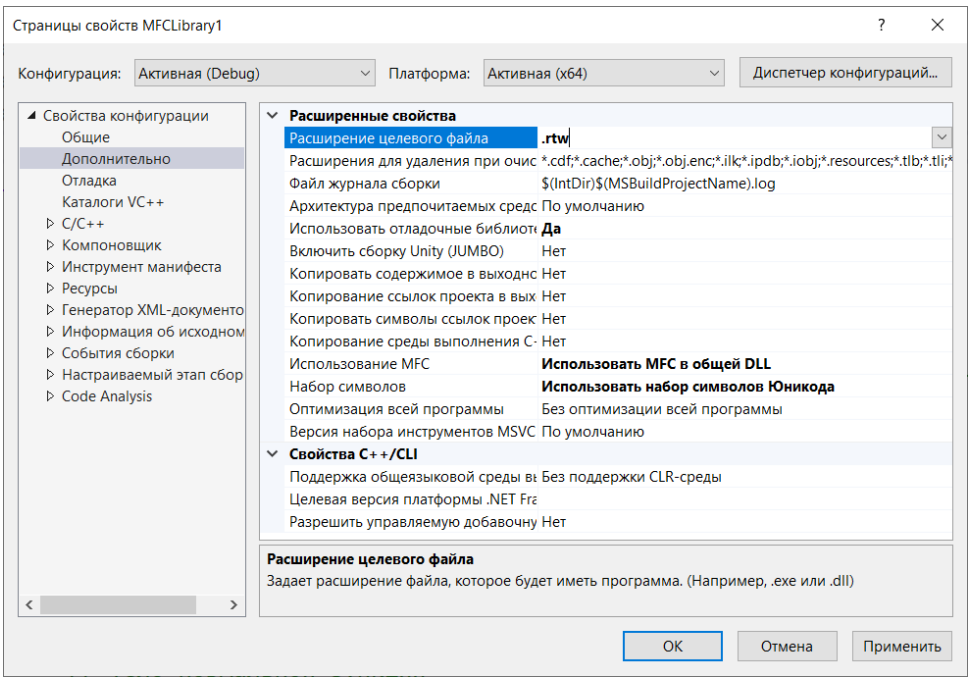



Рисунок 1.7 – Настройка расширения библиотеки в Visual Studio 2017

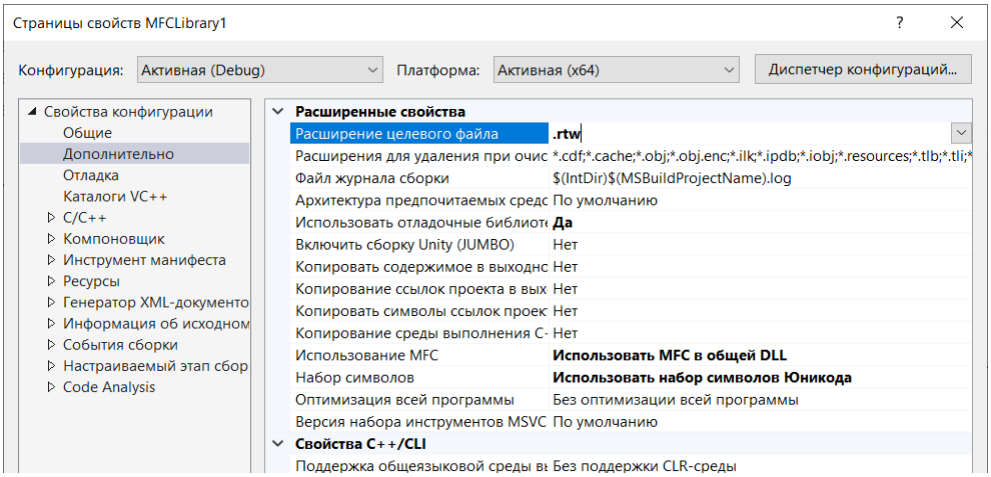



Рисунок 1.8 – Настройка расширения библиотеки в Visual Studio 2019

Шаг 5. В разделе C/C++ → Общие в параметре Дополнительные каталоги включаемых файлов добавить $(KOMPAS_SDK)\Include и $(KOMPAS_SDK)\lib (рисунок 1.9). Система автоматически определит пути к каталогам, в которых расположены файлы и библиотеки прикладного программного интерфейса КОМПАС.

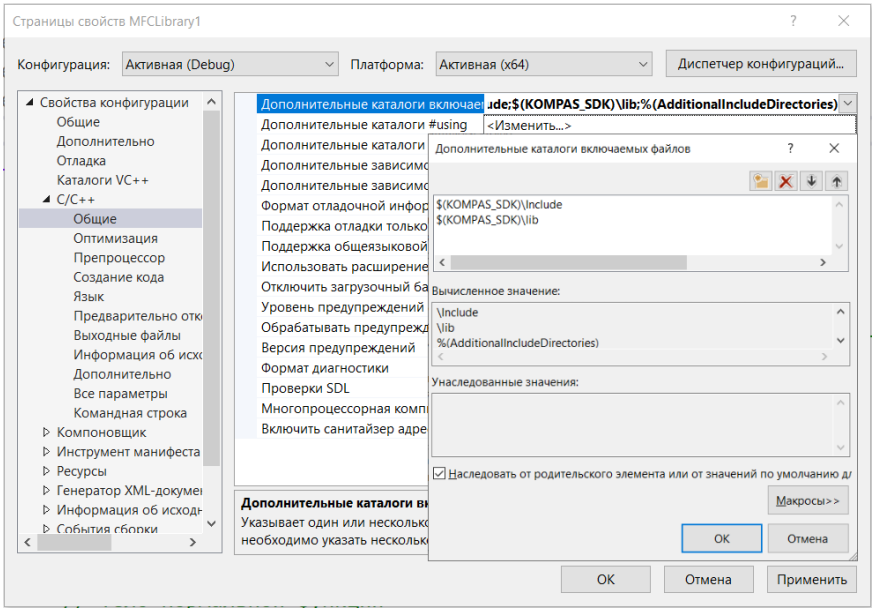



Рисунок 1.9 – Подключение дополнительных каталогов

Шаг 6. В разделе Компоновщик → Общие в параметре Дополнительные каталоги библиотек добавить $(KOMPAS_SDK)\lib64 (рисунок 1.10). Система автоматически определит пути к каталогам, в которых расположены библиотеки для корректной компоновки проекта для 64-разрядной версии КОМПАС.

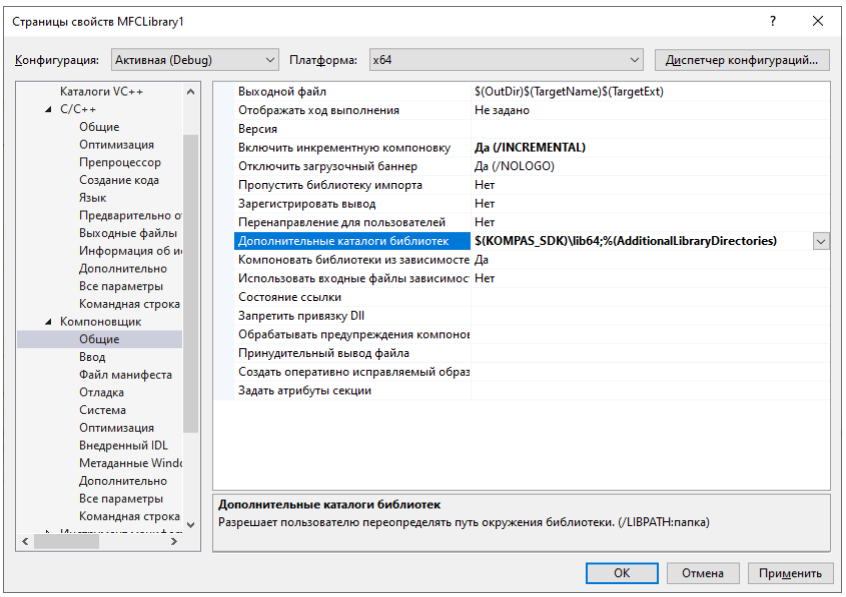



Рисунок 1.10 – Подключение дополнительных библиотек

Шаг 7. В разделе Компоновщик → Ввод в параметре Дополнительные зависимости добавить kapi2d5.lib (рисунок 1.11). Тем самым будет подключена библиотека экспортных функций для работы в 2D версии API5. Этой версии достаточно для большинства разрабатываемых проектов.

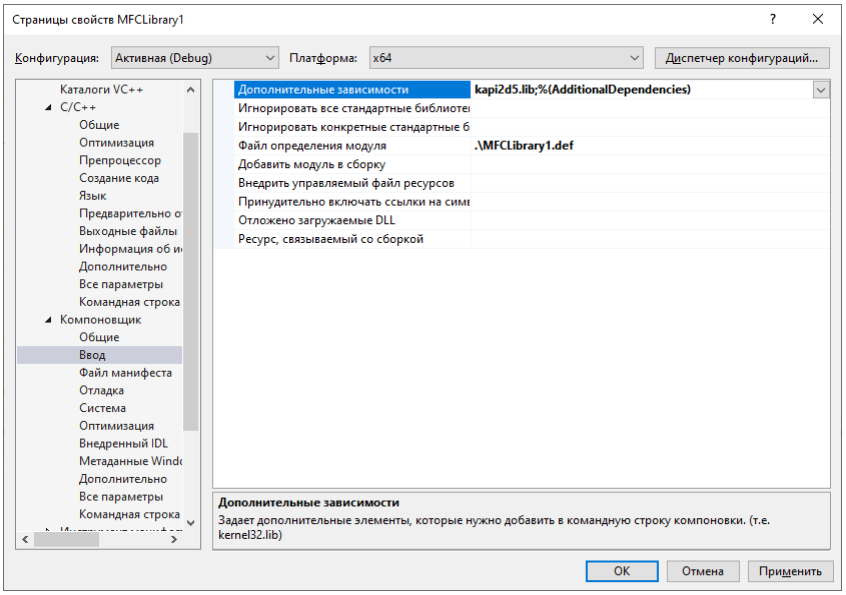



Рисунок 1.11 – Подключение дополнительных зависимостей

## 1.3 Разработка библиотеки КОМПАС

Для создания библиотеки для САПР КОМПАС используются правила оформления библиотек, обеспечивающих согласование системы КОМПАС и приложения.

Функция `LIBRARYENTRY` является обязательной, это точка входа в библиотеку, или, проще говоря, процедура, выполняемая при запуске библиотеки. Ее наличие позволяет системе КОМПАС идентифицировать произвольный RTW-файл как собственную библиотеку, ей передается управление при обращении к приложению. Остальные функции являются необязательными и позволяют определить дополнительные параметры приложения:

`LIBRARYNAME` - функция, возвращающая текстовое название библиотеки;

`LIBRARYID` - функция, возвращающая целый идентификатор библиотеки;

Все функции оформления библиотек должны быть объявлены в *.def файле как экспортные, в нашем примере в файл MFCLibrary1.def нужно добавить описание двух экспортных функций LIBRARYENTRY @2, LIBRARYNAME @3.

Тогда *.def файл будет выглядеть следующим образом:

```
; MFCLibrary1.def: объявляет параметры модуля для библиотеки DLL.

LIBRARY

EXPORTS
 ; Сюда можно направлять явные операции экспорта
 LIBRARYENTRY @2
 LIBRARYNAME @3
```

Добавим в исходный файл C++ указанные в файле с расширением .def функции (остальную часть кода пока оставим как есть, листинг 1.2):

Листинг 1.2 – Исходный код библиотеки для КОМПАС

```
#include "pch.h"
#include "framework.h"
#include "MFCLibrary1.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#endif
#ifndef __LIBTOOL_H
#include "libtool.h"
#endif
char* WINAPI LIBRARYNAME() {
	return "Простая библиотека";
}
void WINAPI LIBRARYENTRY(unsigned int comm) {
	Message("Строим отрезок!");
	LineSeg(10, 10, 100, 100, 1); // x1, y1, x2, y2, тип линии " основная
}
```

После компиляции система выдаст сообщение об ошибке (рисунок 1.12), что невозможно запустить программу.

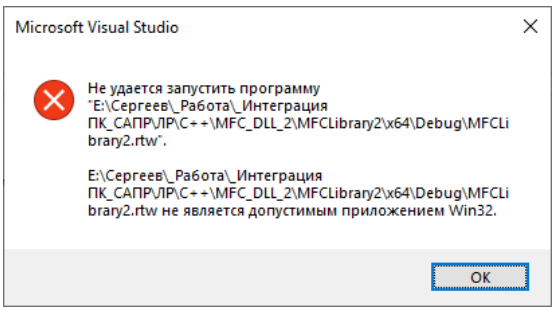



Рисунок 1.12 – Сообщение об ошибки запуска библиотеки

Это говорит о том, что компиляция выполнена успешно, но так как это не исполняемая программа, а динамическая библиотека, то запуск ее на выполнение должен осуществляться из использующего приложения, в нашем случае из под САПР КОМПАС.

Для этого необходимо запустить САПР КОМПАС, выбрать пункт меню Приложения → Добавить приложение (рисунок 1.13), в появившемся диалоговом окне выбрать созданную библиотеку с расширением .rtw (…\MFCLibrary1\x64\Debug\).

После чего в пункте меню Приложения появится новый пункт Простая библиотека, выбор которого запустит разработанную библиотеку на выполнение.

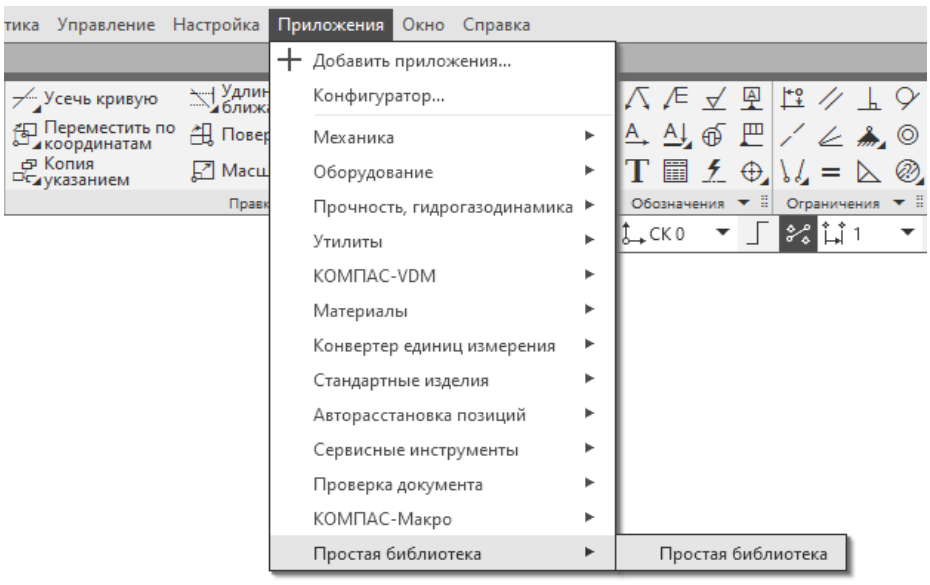



Рисунок 1.13 – Подключение библиотеки в КОМПАС

В результате работы библиотеки появится сообщение (функция Message();), после нажатия на кнопку Закрыть выполнится построение отрезка с помощью функции

```
LineSeg(10, 10, 100, 100, 1); // x1, y1, x2, y2, тип линии «основная»
```

Функция `LineSeg(x1, y1, x2, y2, type)` строит отрезок от точки `x1`, `y1` до точки `x2`, `y2` со стилем линии `type` (рисунок 1.14). Все рассматриваемые далее функции возвращают ссылку на созданный объект, которую можно запомнить в переменную типа `Reference`.

Чтобы перекомпилировать библиотеку, ее необходимо сначала отключить от САПР КОМПАС. Для этого нужно выбрать пункт меню Приложения → Конфигуратор, в появившемся окне выбрать раздел Приложения и в нем название созданной библиотеки (в нашем примере это Простая библиотека), затем справа нажать на команду Приостановить (рисунок 1.15). Здесь же можно полностью отключить библиотеку от КОМПАС.

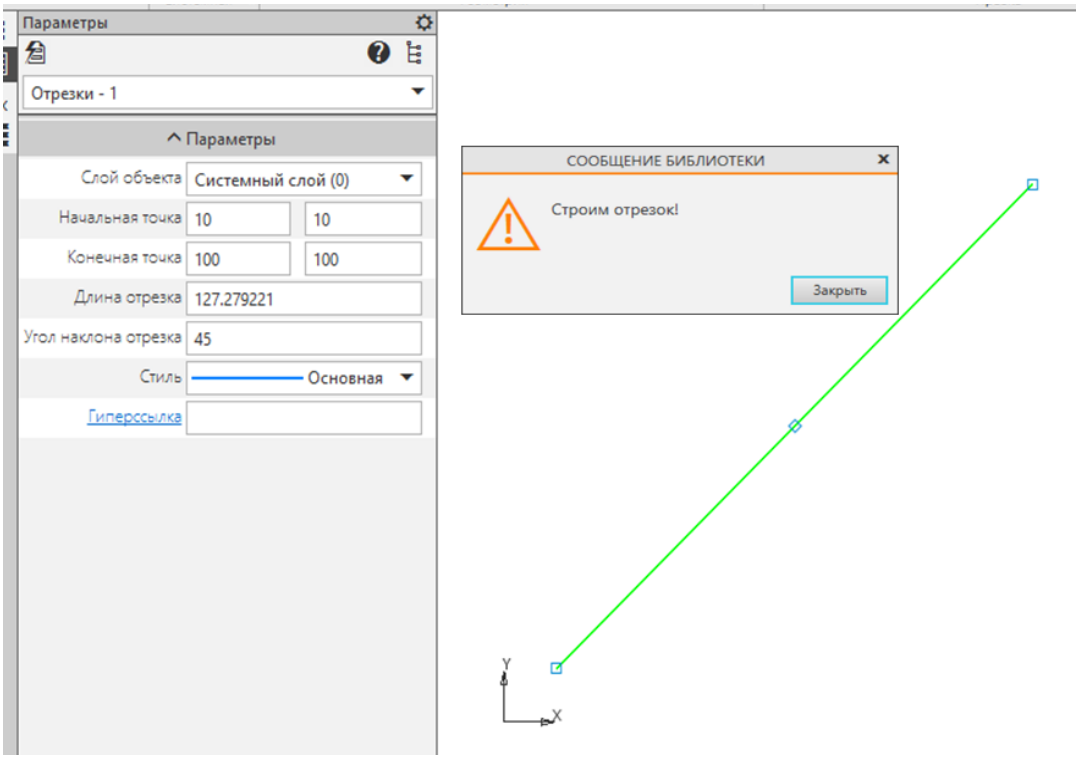



Рисунок 1.14 – Результат работы функции из библиотеки

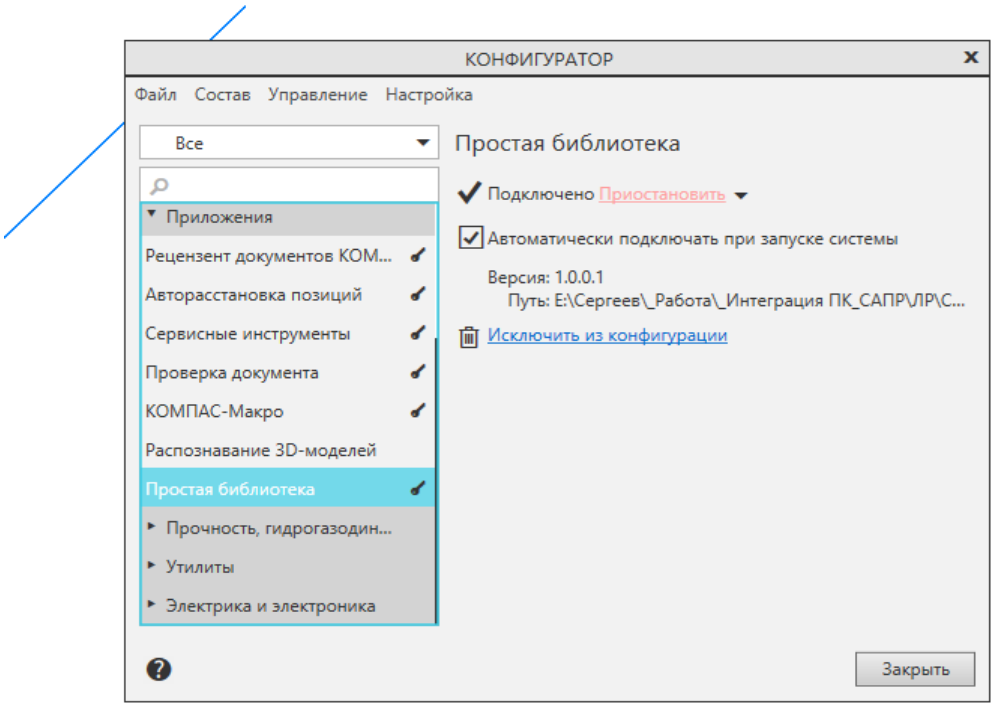



Рисунок 1.15 – Отключение библиотеки в КОМПАС

После того как библиотека откомпилирована, можно удалить лишний программный код, который был создан автоматически системой Visual Studio.

Для этого сначала зайдем внутрь функции

`CMFCLibrary1App::CMFCLibrary1App()`

(нажатие клавиши Ctrl + Левая кнопка мыши на название функции).

Система откроет файл с расширением *.h (MFCLibrary1.h), в котором нужно закомментировать эту функцию с помощью символов /* … */ (листинг 1.3).

Листинг 1.3 – Модификация кода библиотеки

```
// MFCLibrary1.h: основной файл заголовка для библиотеки DLL MFCLibrary1
#pragma once
#ifndef __AFXWIN_H__
#error "include 'pch.h' before including this file for PCH"
#endif
#include "resource.h"// основные символы
// CMFCLibrary1App
// Реализацию этого класса см. в файле MFCLibrary1.cpp
/*
```

```
class CMFCLibrary1App : public CWinApp
{
public:
	CMFCLibrary1App();
	// Переопределение
public:
	virtual BOOL InitInstance();
	DECLARE_MESSAGE_MAP()
};
*/
```

После этого в файле с исходным кодом можно закомментировать или удалить весь неиспользуемый код (листинг 1.4).

Листинг 1.4 – Модификация кода библиотеки

```
/*
BEGIN_MESSAGE_MAP(CMFCLibrary1App, CWinApp)
END_MESSAGE_MAP()
// Создание CMFCLibrary1App
CMFCLibrary1App::CMFCLibrary1App()
{
	// TODO: добавьте код создания,
	// Размещает весь важный код инициализации в InitInstance
}
// Единственный объект CMFCLibrary1App
CMFCLibrary1App theApp;
// Инициализация CMFCLibrary1App
BOOL CMFCLibrary1App::InitInstance()
{
	CWinApp::InitInstance();
	return TRUE;
}
*/
```

Основные функции создания 2D-геометрии приведены в таблице 1.1.

Таблица 1.1 – Основные функции создания 2D-геометрии

| Примитив | Функция | Параметры |
|---|---|---|
| Точка | Point(x, y, style) | Ставит точку с координатами x, y и стилем style |
| Отрезок | LineSeg (x1, y1, x2, y2, type) | Проводит отрезок стилем линии type из точки (x1, y1) в точку (x2, y2), |
| Прямая | Line (x, y, angle) | Проводит бесконечную прямую через точку x, y под углом в градусах angle к положительному направлению оси OX |
| Дуга | ArcBy3Points (x1, y1, x2, y2, x3, y3, type) | Строит дугу по трем точкам стилем линии type |
| Дуга | ArcByAngle (xc, yc, rad, fl, f2, direction, type) | Строит дугу с центром (xc, yc), радиусом rad, из начального fl в конечный f2 угол дуги в градусах, в направлении direction (1 – против часовой стрелки, --1 – по часовой стрелке) и стилем линии type |
| Дуга | ArcByPoint (xc, yc, rad, x1, y1, x2, y2, direction, type); | Строит дугу с центром (xc, yc), радиусом rad, из начальной точки дуги (x1, y1) в конечную точку (x2, y2) в направлении direction (1 – против часовой стрелки, -1 – по часовой стрелке) и стилем линии type |
| Окружность | Circle (xc, yc, rad, type) | Строит окружность с центром в точке xc, yc, радиусом rad и стилем линии type |

В чертежах часто встречается штриховка, которая строится следующим образом: сначала дается команда начала штриховки, затем строятся объекты, образующие ее контур, а затем еще одна команда завершает построение. Начинается штриховка функцией `Hatch(style, Angle, step, width, x0, y0)` которая задает штрихование контура стилем `style` под углом `angle` с шагом `step` и толщиной `width` (0 – штриховать всю область). Начальная точка штриховки `x0`, `y0`.

Коды стилей штриховок приведены в справке по SDK (Software Development Kit - комплект средств разработки). Затем в программе идут команды отрисовки самого контура, а завершается его построение командой `EndObj()`. Пример построения заштрихованной окружности (рисунок 1.16):

```
Hatch(1, 0, 1, 10, 100, 100);
Circle(100, 100, 50, 1); // контур
EndObj();
```

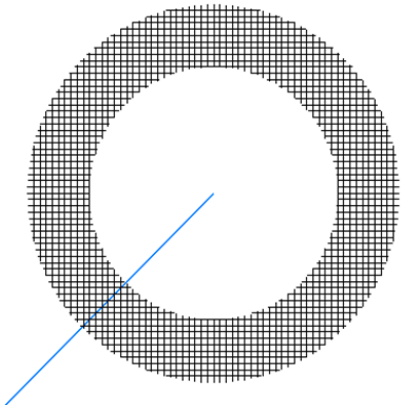


Рисунок 1.16 – Пример программного построения штриховки

### 1.4 Задание для лабораторной работы

1 Изучить теоретический материал.

2 Настроить проект в MS Visual Studio для корректной компиляции прикладной библиотеки САПР КОМПАС.

3 Разработать прикладную библиотеку КОМПАС которая строит чертеж детали по заданному варианту, приведенному в таблице А.1, приложения А.

4 Подготовить ответы на контрольные вопросы.

### 1.5 Содержание отчета

В отчете по лабораторной работе согласно СТО 02069024.101–2015 РАБОТЫ СТУДЕНЧЕСКИЕ [3] должны содержаться следующие пункты:

– название лабораторной работы;

– цель работы;

– задание на лабораторную работу;

– код разработанной программы с комментариями;

– экранные формы с отображением результата выполнения программы в системе КОМПАС;

– выводы;

– список использованных источников.

### 1.6 Контрольные вопросы

1 Опишите принцип расширения возможностей программ с помощью макросов.

2 Опишите принцип расширения возможностей программ с помощью специфических языков программирования.

3 Опишите принцип расширения возможностей программ с помощью дополнительных модулей.

4 В чем отличие обычной прикладной библиотеки Windows и библиотеки для КОМПАС?

5 Как настроить проект в MS Visual Studio для корректного создания библиотеки для САПР КОМПАС?

6 Какие обязательные процедуры и функции должна экспортировать библиотека КОМПАС?

7 Как создается отрезок?

8 Как создается дуга окружности? Какие для этого разработаны функции?

9 Как создается штриховка?

# 2 Создание графических примитивов и вспомогательные построения

## 2.1 Создание графических примитивов

Графическими примитивами называются простейшие элементы чертежа – прямые, точки, окружности и другие геометрические объекты. Примитивы можно изображать в графических документах – на листах чертежей и во фрагментах. Графические примитивы делятся на вспомогательные, простые и составные.

К вспомогательным относятся точки и вспомогательные прямые. Они применяются при построении чертежа (например, в качестве точек привязки), но в окончательном виде удаляются из него специальной командой КОМПАС-График.

Простые примитивы – это линии, окружности, прямоугольники, текст.

Составные примитивы содержат несколько элементов более низкого уровня, например, ломаная задается произвольным количеством вершин.

В таблице 2.1 перечислены основные функции для отображения примитивов.

Таблица 2.1 – Функции для отображения графических примитивов

| № | Имя функции | Назначение |
|---|---|---|
| 1 | 2 | 3 |
| Вспомогательные примитивы | | |
| 1 | Point | Точка |
| 2 | Line | Вспомогательная прямая |
| Простые примитивы | | |
| 3 | LineSeg | Отрезок прямой |
| 4 | Rectangle | Прямоугольник |
| 5 | Circle | Окружность |
| 6 | ArcByAngle | Дуга окружности (по центру и двум углам) |
| 7 | ArcBy3Points | Дуга окружности (по трем точкам) |

Продолжение таблицы 2.1

| 1 | 2 | 3 |
|---|---|---|
| 8 | ArcByPoint | Дуга окружности (по центру и конечным точкам) |
| 9 | Ellipse | Эллипс |
| 10 | EllipseArc | Дуга эллипса |
| 11 | ParEllipseArc | Дуга эллипс (задается параметрически) |
| 12 | ConicArc | Коническое сечение |
| 13 | Equidistant | Эквидистанта (кривая на постоянном расстоянии от |
| 14 | Hatch | Штриховка |
| 15 | ksRegularPolygon | Правильный многоугольник |
| 16 | PointArraw | Условный значок (например, стрелка) |
| 17 | Text | Текстовая строка |
| Составные примитивы | | |
| 18 | ksColouring | Фоновая заливка, включает в себя любые замкнутые элементы |
| 19 | ksPolyline | Ломаная, состоит из точек Point |
| 20 | Bezier | Кривая Безье, задается опорными точками BezierPoint или Point |
| 21 | Nurbs | Сплайновая NURBS-кривая, задается опорными точками NurbsPoint и узлами ksNurbsKnot |
| 22 | Paragraph | Параграф, состоит из строк TextLine |
| 23 | Macro | Макроэлемент. Может содержать любые элементы, объединяемые в единый элемент чертежа. Допускается вложенность макрообъектов |
| 24 | Contour | Контур |
| Геометрические преобразования примитивов | | |
| 25 | RotateObj | Поворот элемента чертежа вокруг заданного центра |
| 26 | ksSymmetryObj | Осевая симметрия элемента чертежа |
| 27 | MoveObj | Переместить элемент чертежа |

Продолжение таблицы 2.1

| 1 | 2 | 3 |
|---|---|---|
| 28 | TransformObj | Преобразовать элемент по текущей матрице преобразований |
| Манипуляции с примитивами | | |
| 29 | ksCopyObj | Копирование элемента чертежа |
| 30 | DeleteObj | Удаление элемента чертежа |
| 31 | DecomposeObj | Разбить составной элемент на составляющие |

Для всех примитивов, за исключением точек и вспомогательных прямых, можно указать тип линии – основная, тонкая, осевая и др.

Некоторые примитивы задаются с помощью довольно большого количества параметров, например, для эллипса требуется указать центр, полуоси и угол наклона одной из полуосей относительно оси OX. Для хранения параметров ряда примитивов в КОМПАС-МАСТЕР определены специальные структуры (и соответствующие интерфейсы).

Функции рисования вспомогательных и простых примитивов возвращают значение типа `Reference` (типа `Longint`) – дескриптор этого примитива (в справочной системе по КОМПАС-МАСТЕР называется указателем на объект чертежа). Эти дескрипторы можно использовать для работы с элементами чертежа с помощью функций геометрических преобразований и функций манипуляций с примитивами.

Для рисования составного элемента требуется организовать последовательность вызовов функций. Рисование начинается вызовом функции отображения необходимого составного элемента. Если эта функция возвратила в качестве кода успешного завершения 1, то далее вызываются функции для рисования элементов, входящих в составной элемент. Для завершения рисования требуется вызвать функцию `EndObj()`. Она возвращает дескриптор сформированного составного элемента. Ниже приведен пример рисования ломаной по четырем вершинам (листинг 2.1, рисунок 2.1).

Листинг 2.1 – Пример рисования ломанной линии

```
/*
ksPolyline - Создать ломаную
Синтаксис : int ksPolyline(unsigned short style);
Входной параметр :
style стиль линии
*/
if (ksPolyline(1))
{
	// При рисовании точки задается x, y и стиль отрисовки точки
	Point(10, 10, 1);
	Point(20, 20, 1);
	Point(30, 10, 1);
	Point(40, 20, 1);
	EndObj();
}
```

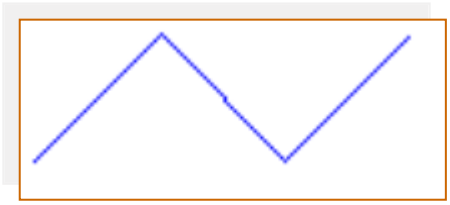


Рисунок 2.1 – Результат рисования ломанной линии

Дескриптор элемента действителен до завершения выполнения библиотечной команды, в которой он был создан.

Если надо использовать элемент в различных командах библиотеки (например, запомнить значение его дескриптора в глобальной переменной и многократно обращаться к элементу из разных команд), то обязательно нужно вызвать для этого дескриптора функцию `KeepReference`, запрещающую удалять временный объект после завершения команды библиотеки. Тогда КОМПАС-График перенесет дескриптор из динамической таблицы в глобальную. Она сохраняется в течение всего сеанса работы с графическим документом, содержащим элемент с данным дескриптором.

### 2.1.1 Применение матриц геометрических преобразований

Составной элемент чертежа удобно задавать в системе координат, связанной с этим элементом. Но в таком случае возникает необходимость задавать местоположение и ориентацию системы координат элемента в системе координат графического документа.

Допустим, на чертеже несколько раз встречается составной элемент из 3-х отрезков, окружности и дуги окружности, который удобно описать в системе координат, связанной с центром окружности – будем считать эту точку «центром элемента» (рисунок 2.2, а). На чертеже требуется построить несколько подобных элементов, отличающихся ориентацией и размером (рисунок 2.2, б, в).

а б в

а – исходный составной элемент; б – элемент уменьшен в 2 раза; в – элемент повернут вокруг центра на 30 градусов

Рисунок 2.2 – Исходный составной элемент и его преобразования

В таком случае удобно пользоваться матрицами геометрических преобразований. Сначала в листинге 2.2 определяется функция рисования составного элемента – макроэлемента, показанного на рисунке 2.2, а с помощью команды Macro:

Листинг 2.2 – Рисование макроэлемента

```
reference DrawElem()
{
    if (Macro(0))
```

```
    {     LineSeg(-10, 10, 10, 10, 1);
          LineSeg(-10, 0, -10, 10, 1);
          LineSeg(10, 0, 10, 10, 1);
          Circle(0, 0, 5, 1);
          ArcByPoint(0, 0, 10, -10, 0, 0, 0, 1, 1);
          return EndObj();
    }
    return NULL;
}
```

Синтаксис команды Macro для создания нового макроэлемента:

```
int Macro(unsigned char type);
```

Входной параметр: `type` – тип макроэлемента: 0 – объединяет объекты текущего слоя, 1– включаемые объекты могут принадлежать разным слоям.

Возвращаемое значение: 1 – в случае успешного завершения, 0 – в случае неудачи.

Функция `EndObj()` возвращает дескриптор созданного макроэлемента или `NULL` при неудаче. На чертеже пользователь может работать с макроэлементом как с единым целым, а не как с отдельными отрезками, окружностями и т.п. (но при необходимости макроэлемент можно разрушить для выделения отдельных составляющих). В программе для манипуляций с макроэлементом также требуется всего один дескриптор.

Для изменения расположения, ориентации или масштаба элемента можно применить два способа:

– сначала задать матрицу преобразований системы координат `ksMtr`, а затем вызывать функции рисования элемента;

– нарисовать составной элемент как макроэлемент, а потом применить к нему матрицу преобразований с помощью функции `TransformObj`.

Рассмотрим необходимые функции.

Для создания матрицы преобразования координат используется функция `int ksMtr(double x, double y, double angle, double scaleX, double scaleY)`.

Входные параметры: `x, y` – координаты начала локальной системы координат, `angle` – угол наклона системы координат в градусах, `scaleX, scaleY` – масштаб локальной системы координат по осям X и Y.

Возвращаемое значение: 1 – в случае успешного завершения, 0 – в случае неудачи.

Для отмены матрицы преобразования координат используется функция

```
int DeleteMtr(void);
```

Возвращаемое значение: 1 - в случае успешного завершения, 0 - в случае неудачи.

Для преобразования объекта по установленной матрице используется фонация

```
int TransformObj(reference ref);
```

Входной параметр: `ref` - указатель на объект.

Возвращаемое значение: 1 - в случае успешного завершения, 0 - в случае неудачи.

Ниже приведен исходный текст (листинги 2.3 и Листинг 2.4) для рисования элементов, показанных на рисунке 2.2 обоими способами.

Листинг 2.3 – Исходный код для рисования элементов

```
// Нарисовать исходный макроэлемент
DrawElem();
// Перенос системы координат на 30 единиц и уменьшение в 2 раза по обеим осям
ksMtr(30, 0, 0, 0.5, 0.5);
// Рисование макроэлемента с учетом переноса и масштабирования
DrawElem();
// Удаление матрицы преобразований, т.е. восстановление параметров
// системы координат
DeleteMtr();
// Перенос системы координат на 60 единиц и поворот на 30
// градусов против часовой стрелки
ksMtr(60, 0, 30, 1.0, 1.0);
// Рисование повернутого макроэлемента
DrawElem();
DeleteMtr(); // Удаление матрицы преобразований
```

Листинг 2.4 – Исходный код для рисования элементов

```
// Нарисовать три одинаковых макроэлемента
DrawElem();
reference ref1 = DrawElem();
reference ref2 = DrawElem();
// Перенести и уменьшить макроэлемент ref1
ksMtr(30, 0, 0, 0.5, 0.5);
TransformObj(ref1);
DeleteMtr();
// Перенести и оставить масштаб макроэлемента ref2
ksMtr(60, 0, 30, 1.0, 1.0);
TransformObj(ref2);
DeleteMtr();
```

Во втором способе (листинг Листинг 2.4) операции рисования и геометрических преобразований не смешиваются друг с другом в исходном тексте программы.

### 2.2 Вспомогательные построения

Вспомогательные построения (таблица Таблица 2.2) выполняют вычисления и в основном возвращают результат либо в виде числа (например, при расчете расстояний), либо в виде динамического массива (расчет точек пересечений). В редких случаях для возврата результатов применяются специальные структуры (интерфейсы).

Визуальных изменений на чертеже вспомогательные построения не выполняют, чтобы отобразить результаты вычислений, необходимо пользоваться функциями создания графических элементов.

Таблица 2.2 – Функции вспомогательных построений

| № | Имя функции | Назначение |
|---|---|---|
| 1 | 2 | 3 |
| Вычисление расстояний | | |
| 1 | ksDistancePntArc | Между точкой и дугой окружности |
| 2 | ksDistancePntCircle | Между точкой и окружностью |
| 3 | ksDistancePntLine | Между точкой и прямой |
| 4 | ksDistancePntLineForPoint | Между точкой и прямой, заданной двумя точками |
| 5 | ksDistancePntLineSeg | Между точкой и отрезком |
| 6 | DistancePntPnt | Между двумя точками |
| Вычисление точек пересечения | | |
| 7 | IntersectArcArc | Две дуги окружностей |
| 8 | IntersectArcLin | Дуга окружности и прямая |
| 9 | IntersectCirArc | Окружность и дуга окружности |
| 10 | IntersectCirCir | Две окружности |
| 11 | IntersectCirLin | Окружность и прямая |
| 12 | IntersectCurvCurv | Две кривые |
| 13 | IntersectLinLin | Две прямые |
| 14 | IntersectLinSArc | Отрезок и дуга окружности |
| 15 | IntersectLinSCir | Отрезок и окружность |
| 16 | IntersectLinSLine | Отрезок и прямая |
| 17 | IntersectLinSLinS | Два отрезка |
| Вычисление касательных | | |
| 18 | TanCircleCircle | Прямая, касательная к двум окружностям |
| 19 | TanLineAngCircle | Окружность и прямая, проходящая под заданным углом |

Продолжение таблицы 2.2

| 1 | 2 | 3 |
|---|---|---|
| 20 | TanLinePointCircle | Окружность и прямая, проходящая через заданную точку |
| 21 | ksTanLinePointCurve | Кривая и прямая, проходящая через заданную точку |
| Вычисление сопрягающих окружностей | | |
| 22 | ksCouplingCircleCircle | Сопряжение двух окружностей |
| 23 | ksCouplingLineCircle | Сопряжение окружности и прямой |
| 24 | ksCouplingLineLine | Сопряжение двух прямых |
| 25 | Вычисления с кривыми | |
| 26 | ksCalcInertiaProperties | Плоские массо-центровочные характеристики кривой |
| 27 | ksCalcMassInertiaProperties | Объемные массо-центровочные характеристики тел вращения или выдавливания, заданных кривой или группой кривых. |
| 28 | ksGetCurvePerimeter | Периметр кривой |
| 29 | ksGetCurvePerpendicular | Угол нормали к кривой в заданной точке |
| 30 | ksGetCurvePointProjection | Координаты проекции точки на кривую |
| 31 | ksMovePointOnCurve | Координаты точки, находящейся на указанном вдоль кривой расстоянии от заданной точки |
| 32 | ksPointsOnCurve | Вычисление точек, равномерно расположенных на кривой |
| Геометрические преобразования | | |
| 33 | Rotate | Поворот точки на угол (в градусах) вокруг заданного центра |
| 34 | Symmetry | Осевая симметрия точки |

Продолжение таблицы 2.2

| 1 | 2 | 3 |
|---|---|---|
| Тригонометрические функции (аргумент в градусах) | | |
| 35 | AtanD | Арктангенс |
| 36 | CosD | Косинус |
| 37 | SinD | Синус |
| 38 | TanD | Тангенс |
| Разное | | |
| 39 | Angle | Угол между вектором и осью OX |
| 40 | ksEqualPoints | Проверить совпадение двух точек |
| 41 | Perpendicular | Вычислить точку пересечения отрезка и перпендикуляра к нему, проходящего через заданную точку |

Ниже описана процедура (листинг 2.5), вычисляющая пересечение двух прямых и отображающая эти прямые и точку пересечения на текущем листе КОМПАС-График (рисунок 2.3). Эта процедура выполняет действия в следующей последовательности:

– вычисление точки пересечения прямых;

– рисование прямых;

– рисование точки.

Вычислительные операции выполняет математический интерфейс, операции отображения – интерфейс графического документа.

Особо следует отметить вопрос о передаче параметров функциями. В ряде случаев количество параметров функции может быть переменным. Точка пересечения двух прямых, конечно, либо одна, либо ни одной (или прямые совпадают), но, например, точек пересечения произвольных кривых может быть произвольное количество.

Листинг 2.5 – Вычисление пересечения двух прямых линий

```
// Пересечь прямые Отрисовка прямых
```

```
Line(10, 10, 0);
Line(15, 5, 90);
int count1; // Количество пересечений
double x, y; // Точка пересечения
// Получить координаты точки пересечения двух прямых
IntersectLinLin(10, 10, // Точка на первой прямой
      0, // Угол первой прямой
      15, 5, // Точка на второй прямой
      90, // Угол второй прямой
      &count1, // Количество точек пересечения
      &x, &y); // Точка пересечения
      // Отрисовка точки пересечения
if (count1)
      Point(x, y, 3);
```

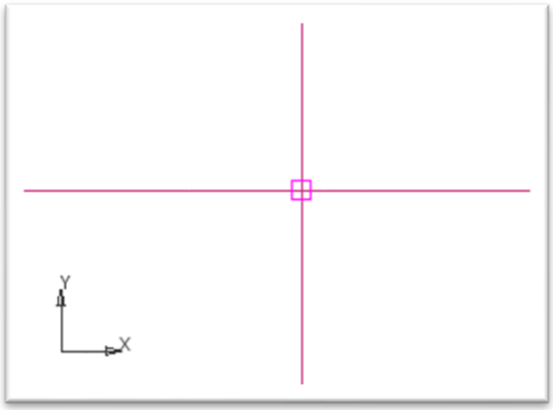



Рисунок 2.3 – Результат выполнения программы из листинга 2.5

Ниже приведен пример (листинг 2.6) вычисления сопрягающих окружностей для двух пересекающихся прямых. Радиус окружностей задается константой, затем вычисляются их центры и точки касания прямых и окружностей. Результаты вычислений отображаются на листе чертежа (рисунок 2.4).

Листинг 2.6 – Вычисление сопрягающих окружностей

```
// Сопрягающие окружности к двум прямым Отрисовка прямых
Line(100, 100, 45);
```

```
Line(100, 100, -45);
double rad = 20; // Радиус сопряжения
int count; // Число сопряжений
CON con[4]; // Массив данных по каждому сопряжению ( структура CON включает:
// центр сопрягающей окружности, координаты точки сопряжения )
// Получить параметры окружностей, касательной к двум прямым
CouplingLineLine(100, 100, 45, // Первая прямая
      100, 100, -45, // Вторая прямая
      rad, // Радиус сопряжения
      &count, // Число сопряжений
      con); // Массив данных по каждому сопряжению
      // Отрисовка сопрягающихся окружностей и точек касания
for (int i = 0; i < count; i++) {
      Circle(con[i].xc, con[i].yc, rad, 1);
      Point(con[i].x1, con[i].y1, i);
      Point(con[i].x2, con[i].y2, i);
}
// Результат сопряжения
TCHAR buf[255];
_stprintf_s(buf, _T("count = %d, con[0].x1 = %4.2f, con[0].y1 =
%4.2f,\ncon[0].x2 = %4.2f, con[0].y2 = %4.2f ..."), count, con[0].x1,
con[0].y1, con[0].x2, con[0].y2);
MessageT(buf);
```

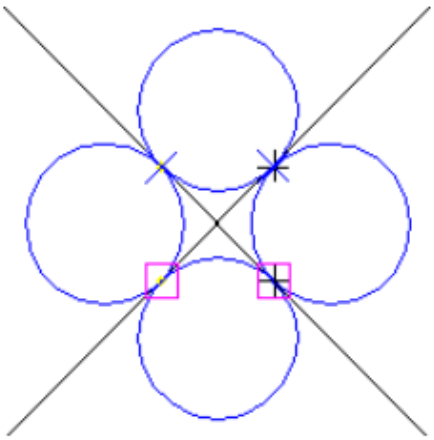


Рисунок 2.4 – Результат выполнения программы из листинга 2.6

### 2.3 Определение параметров сопрягающих окружностей определенного радиуса и точек сопряжения для двух окружностей

Разработаем программу, которая формирует изображение в САПР КОМПАС в соответствии с рисунком 2.5.

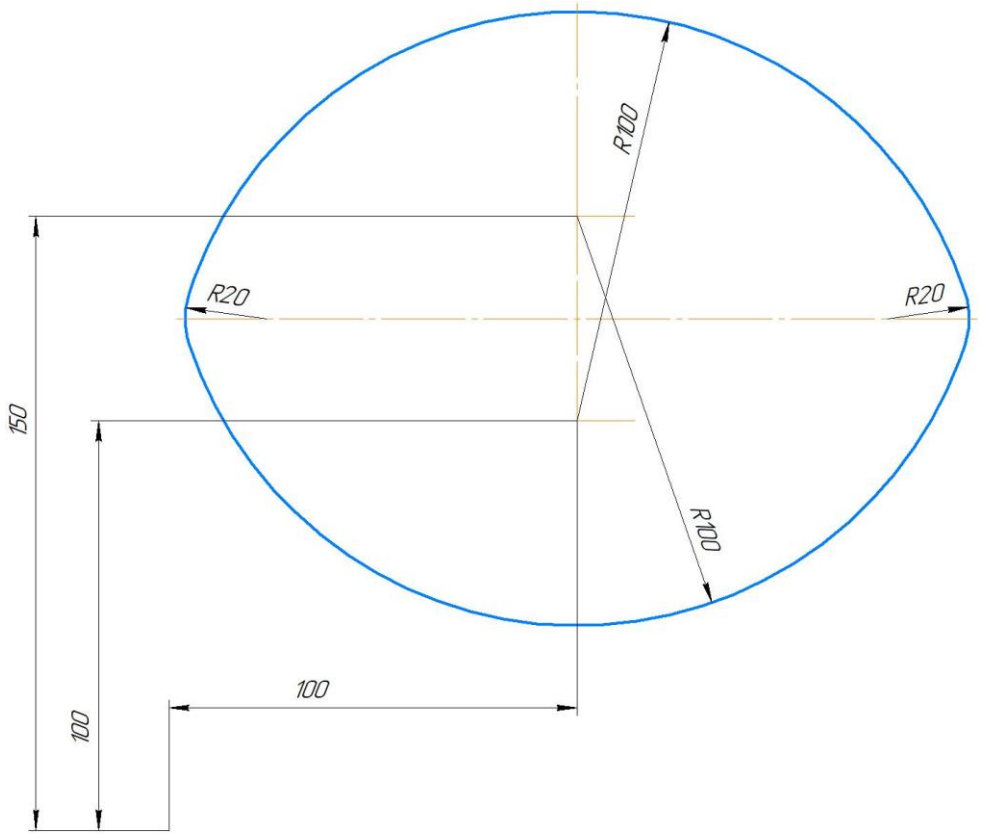



Рисунок 2.5 – Изображения для формирования в САПР КОМПАС

В начале необходимо построить все сопряжения для дуг окружностей радиусом 100 мм, дугами радиусом 20 мм (листинг 2.7).

Листинг 2.7 – Построение сопряжений для дуг

```
double rad = 20;
int kp;
CON con[8];
Circle(100, 100, 100, 1);
Circle(100, 150, 100, 1);
```

```
ksCouplingCircleCircle(100, 100, 100, //параметры первой окружности
      100, 150, 100, //параметры второй окружности
      rad, &kp, con);
//отрисуем окружности и точки сопряжения
for (short i = 0; i < 8; i++) {
      Circle(con[i].xc, con[i].yc, rad, i + 1);
      Point(con[i].x1, con[i].y1, i);
      Point(con[i].x2, con[i].y2, i);
}
```

Нас интересуют 4 дуги, отмеченные стрелками (рисунок 2.6).

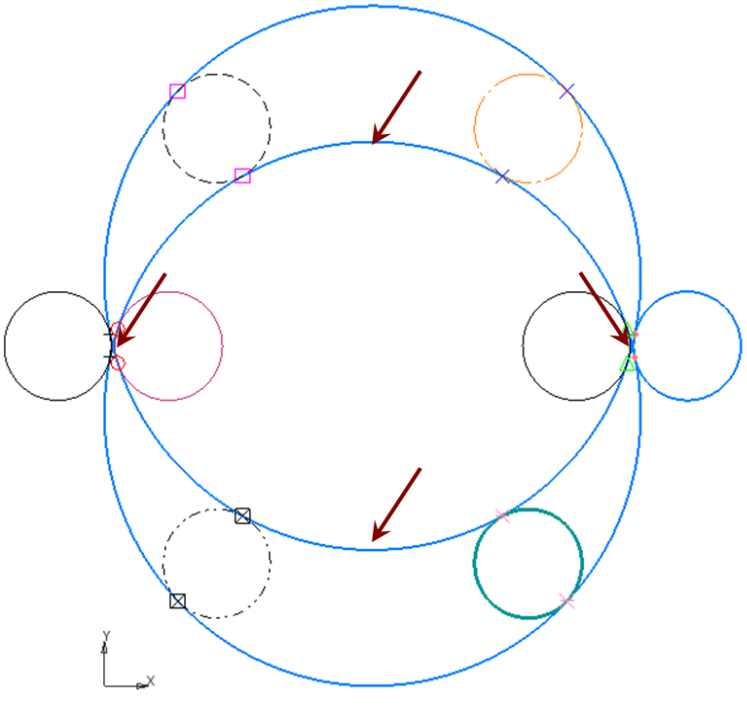
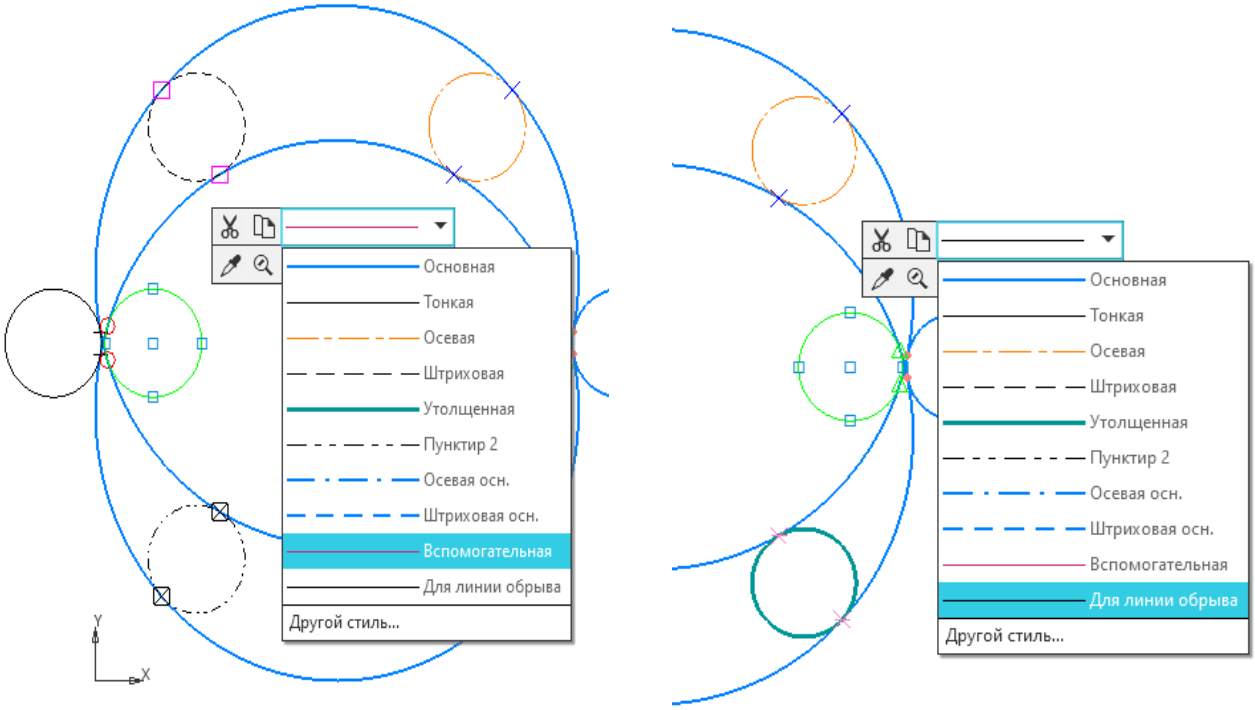


Рисунок 2.6 – Результат построения с указанием точек сопряжения

В САПР КОМПАС определяем стиль линии интересующих окружностей.

Из рисунков видно, что окружность слева имеет стиль линии «Вспомогательная» (рисунок 2.7, а), а справа – «Для линии обрыва» (рисунок 2.7, б).




а б

а – стиль линий «Вспомогательная»; б – стиль линий «Для линии обрыва»

Рисунок 2.7 – Определение стилей линий

Определим в SDK КОМПАС коды интересующих нас стилей линий. «Вспомогательная» – 6, «Для линии обрыва» – 5 (таблица 2.3).

Таблица 2.3 – Коды стилей линий

| № | Код | Стиль линии |
|---|---|---|
| 1 | 2 | 3 |
| 1 | 1 | основная |
| 2 | 2 | тонкая |
| 3 | 3 | осевая |
| 4 | 4 | штриховая |
| 5 | 5 | для линии обрыва |
| 6 | 6 | вспомогательная |
| 7 | 7 | утолщенная |

Продолжение таблицы 2.3

| 1 | 2 | 3 |
|---|---|---|
| 8 | 8 | пунктир 2 |
| 9 | 9 | штриховая основная |
| 10 | 10 | осевая основная |
| 11 | 11 | тонкая линия, включаемая в штриховку |

Введем в программу условие (листинг 2.8), ограничивающее формирование окружностей только стилями 5 и 6 (отмечены ранее стрелками).

Листинг 2.8 – Ограничение формирования окружностей стилями 5 и 6

```
double rad = 20;
int kp;
CON con[8];
Circle(100, 100, 100, 1);
Circle(100, 150, 100, 1);
ksCouplingCircleCircle(100, 100, 100, //параметры первой окружности
      100, 150, 100, //параметры второй окружности
      rad, &kp, con);
//отрисуем окружности и точки сопряжения
for (short i = 0; i < 8; i++) {
      //Условие, ограничивающее формирование окружностей
      if ((i == 4) || (i == 5)) {//так как i начинается с 0,
      //а стили линий с 1, то 5 и 6 уменьшаем на 1
            Circle(con[i].xc, con[i].yc, rad, i + 1);
            Point(con[i].x1, con[i].y1, i);
            Point(con[i].x2, con[i].y2, i);
      }
}
```

В результате выполнения программы видно, что сопрягающие окружности определены верно (рисунок 2.8).

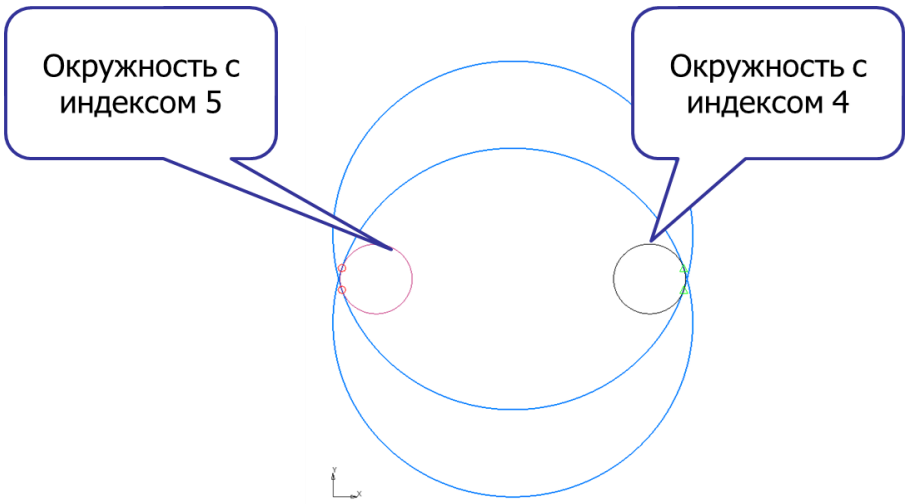



Рисунок 2.8 – Выполнение условия сопряжения окружностей по стилям линий

Сформируем верхнюю и нижнюю дуги окружности стилем линии «Утолщенная» (7) с помощью функции ArcByPoint, которая строит дугу окружности по центру, радиусу и двум точкам (рисунок 2.9).

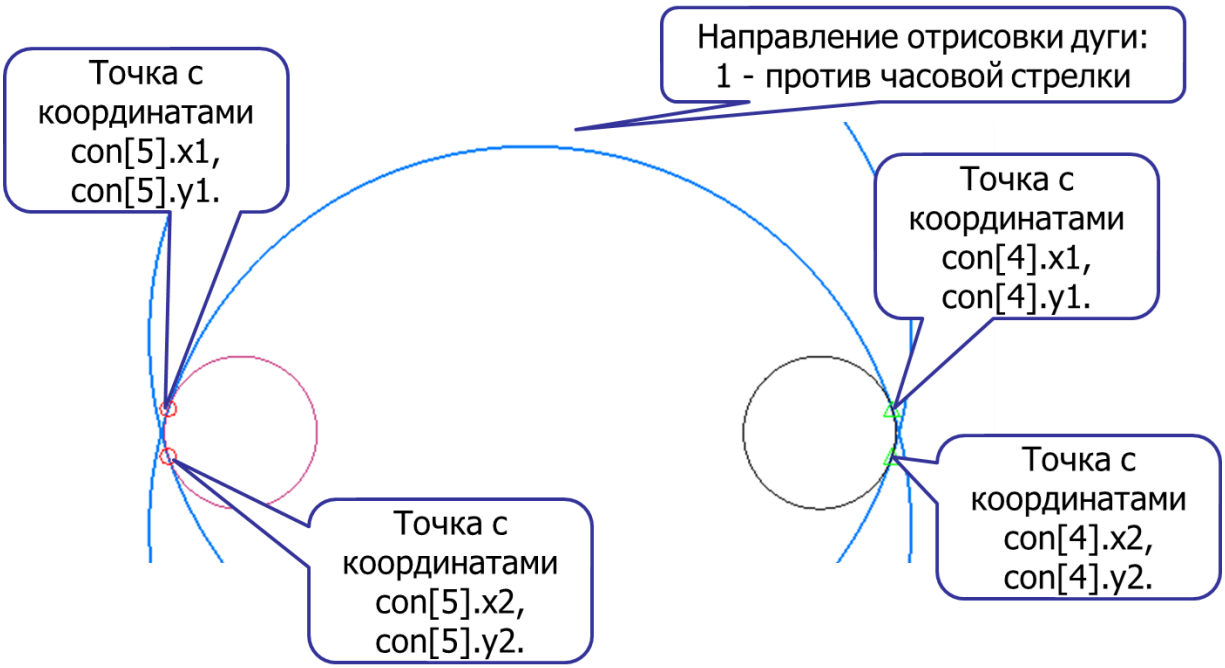



Рисунок 2.9 – Условие формирования дуг окружности стилем линии 7

В программу добавим соответствующие строки (листинг 2.9).

Листинг 2.9 – Формирование дуг окружности стилем линии 7

```
double rad = 20;
int kp;
CON con[8];
Circle(100, 100, 100, 1);
Circle(100, 150, 100, 1);
ksCouplingCircleCircle(100, 100, 100, //параметры первой окружности
      100, 150, 100, //параметры второй окружности
      rad, &kp, con);
//отрисуем окружности и точки сопряжения
for (short i = 0; i < 8; i++) {
      //Условие, ограничивающее формирование окружностей
      if ((i == 4) || (i == 5)) {//так как i начинается с 0,
      //а стили линий с 1, то 5 и 6 уменьшаем на 1
            Circle(con[i].xc, con[i].yc, rad, i + 1);
            Point(con[i].x1, con[i].y1, i);
            Point(con[i].x2, con[i].y2, i);
      }
}
//дуга верхняя
ArcByPoint(100, 100, 100, con[4].x1, con[4].y1, con[5].x1, con[5].y1, 1, 7);
//дуга нижняя
ArcByPoint(100, 150, 100, con[4].x2, con[4].y2, con[5].x2, con[5].y2, -1, 7);
```

В результате построение выполнено верно, построенные дуги выделены (рисунок 2.10). Для нижней дуги задано направление отрисовки дуги -1 – по часовой стрелке.

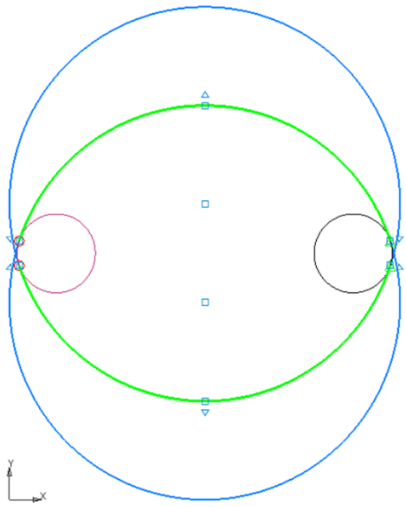


Рисунок 2.10 – Результат формирования верхней и нижней дуг окружности

Сформируем левую и правую дуги окружности, а вспомогательный код программы удалим (листинг 2.10). Стили отрисовки зададим последовательно от 1 до 4 для того, чтобы видеть какой фрагмент кода, что строит.

Листинг 2.10 – Формирование левой и правой дуги окружности

```
double rad = 20;
int kp;
CON con[8];
ksCouplingCircleCircle(100, 100, 100, //параметры первой окружности
      100, 150, 100, //параметры второй окружности
      rad, &kp, con);
//дуга верхняя
ArcByPoint(100, 100, 100, con[4].x1, con[4].y1, con[5].x1, con[5].y1, 1, 1);
//дуга нижняя
ArcByPoint(100, 150, 100, con[4].x2, con[4].y2, con[5].x2, con[5].y2, -1, 2);
//дуга правая
ArcByPoint(con[4].xc, con[4].yc, rad, con[4].x1, con[4].y1, con[4].x2,
con[4].y2, -1, 3);
//дуга левая
ArcByPoint(con[5].xc, con[5].yc, rad, con[5].x1, con[5].y1, con[5].x2,
con[5].y2, 1, 4);
```

В результате будет получено изображение, соответствующее рисунку 2.11.

Изменим стиль линии для всех построений на «Основная», получим требуемую фигуру.

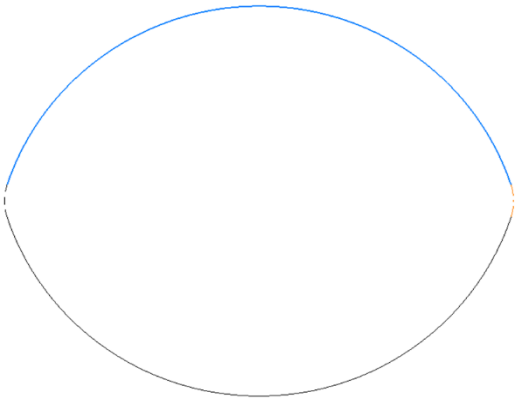


Рисунок 2.11 – Левая и правая дуги окружности

## 2.4 Определение параметров сопрягающих окружностей определенного радиуса и точек касания при сопряжении окружности и прямой

Разработать программу, которая формирует изображение в САПР КОМПАС в соответствии с рисунком 2.12.

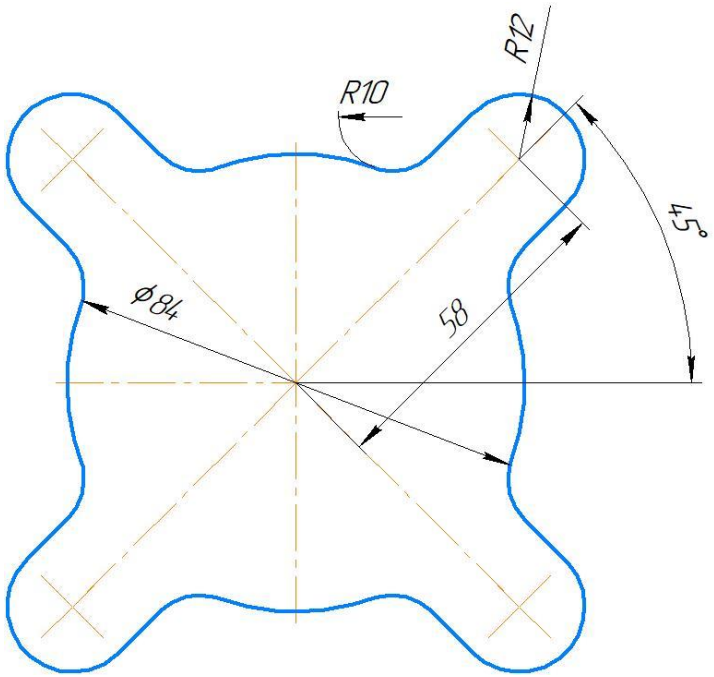



Рисунок 2.12 – Изображения для формирования в САПР КОМПАС

Из рисунка видно, что изображение симметрично относительно всех указанных осей. Поэтому построим только один фрагмент, который затем можно будет использовать в операциях симметрии и копирования.

Чтобы воспользоваться операцией `ksCouplingLineCircle` необходимо указать координаты любой точки на линии на расстоянии 12 мм, параллельной осевой, расположенной под углом 45°.

Исходя из представленной схемы (рисунок 2.13) легко можно определить расстояние c, которое будет соответствовать координате Y искомой прямой. $b = a$, так как треугольник равнобедренный. $c = \sqrt{a^2 + b^2} = \sqrt{288} = 16.970563$ мм.

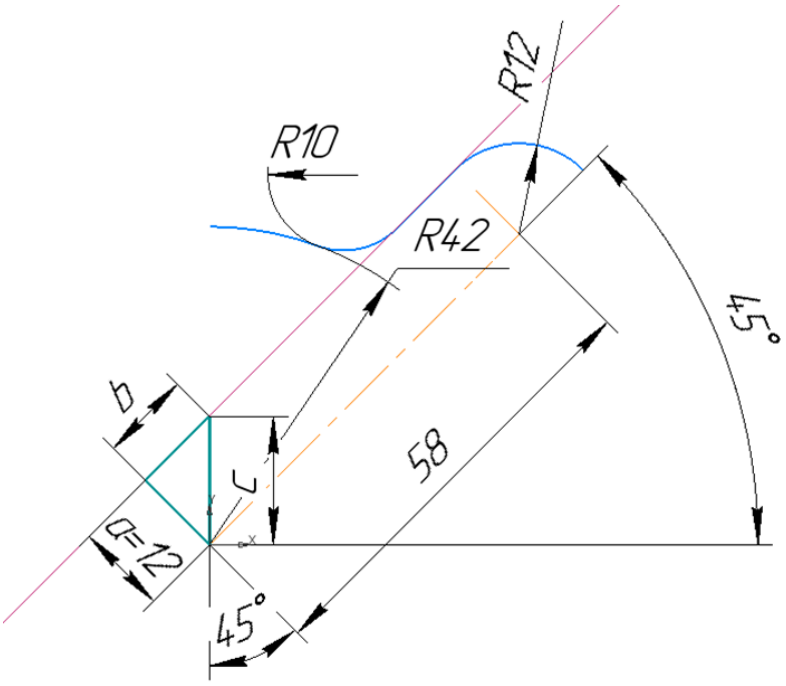



Рисунок 2.13 – Фрагмент формируемого изображения

Тогда код программы для поиска сопряжений окружности и дуги будет выглядеть следующим образом (листинг 2.11).

Листинг 2.11 – Код программы для поиска сопряжений окружности и дуги

```
double rad = 10;
int kp;
CON con[8];
```

```
Circle(0, 0, 42, 1);
Line(0, 16.970563, 45);
ksCouplingLineCircle(0, 0, 42, //параметры окружности
      0, 16.970563, 45, //параметры линии
      rad, &kp, con);
//отрисуем окружности и точки сопряжения
for (short i = 0; i < 8; i++) {
      Circle(con[i].xc, con[i].yc, rad, i + 1);
      Point(con[i].x1, con[i].y1, i);
      Point(con[i].x2, con[i].y2, i);
}
```

В результате будет построено изображение, представленное на рисунке 2.14, из которого следует, что необходимые параметры сопряжения хранятся в массиве под номером 0, так как окружность, построенная по этим параметрам имеет стиль линии 1.

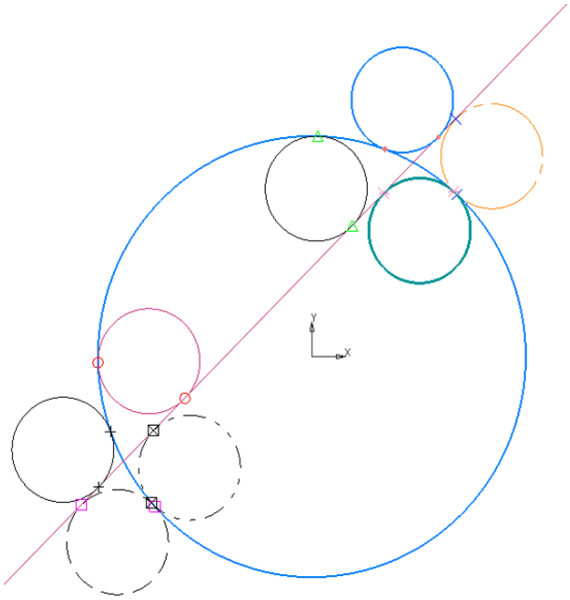


Рисунок 2.14 – Результат работы программы 2.11

Построим требуемые дуги окружностей. Код для вспомогательных построений удалим (листинг 2.12).

Листинг 2.12 – Построение дуг окружностей

```
double rad = 10;
int kp;
CON con[8];
ksCouplingLineCircle(0, 0, 42, //параметры окружности
      0, 16.970563, 45, //параметры линии
      rad, &kp, con);
//Построение
ArcByPoint(0, 0, 42, con[0].x2, con[0].y2, 0, 42, 1, 1);
ArcByPoint(con[0].xc, con[0].yc, rad, con[0].x1, con[0].y1, con[0].x2,
con[0].y2, -1, 1);
```

В результате будет построено 2 дуги (рисунок 2.15).

Рисунок 2.15 – Построение двух дуг

Для построения отрезка одна координата имеется (конечная точка дуги радиусом 10 мм), необходимо определить вторую координату, соответствующую точке касания отрезка и дуги радиусом 12 мм. Сделать это можно с помощью функции `TanLineAngCircle`, позволяющей получить точки касания окружности и прямой, проходящей под заданным углом.

Для этого нужно определить координаты центра дуги с радиусом 12 мм. Расстояние от центра системы координат до центра этой дуги по чертежу составляет 58 мм, угол расположения – 45°, что также соответствует прямоугольному треугольнику с равными катетами, которые определяются по формуле

$$a = b = c \cdot cos45 = 58 \cdot cos45 = 41.012193 \text{ мм.}$$

Фрагмент кода для построения отрезка (рисунок 2.16) будет выглядеть следующим образом

```
double x[2], y[2];
```

```
TanLineAngCircle(41.012193, 41.012193, 12, 45, x, y);
LineSeg(con[0].x1, con[0].y1, x[1], y[1], 1);
```

Рисунок 2.16 – Результат добавления отрезка

Теперь нужно построить дугу радиусом 12 мм. Первая точка дуги будет соответствовать второй координате отрезка, необходимо определить координаты второй конечной точки. Сделать это можно с помощью функции `IntersectCirLin`, позволяющей получить координаты точек пересечения окружности и прямой.

Код построения дуги по найденным координатам будет следующим. Результат построения приведен на рисунке 2.17.

```
double xx[2], yy[2];
int k;
IntersectCirLin(41.012193, 41.012193, 12, 0, 0, 45, &k, xx, yy);
ArcByPoint(41.012193, 41.012193, 12, x[1], y[1], xx[0], yy[0], -1, 1);
```

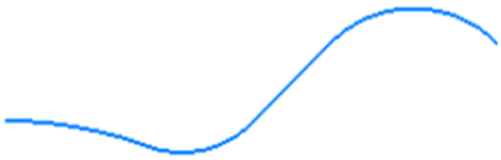


Рисунок 2.17 – Результат добавления третьей дуги

Для того, чтобы применить операции симметрии (рисунок 2.18) и копирования необходимо объединить все построения в макроэлемент (листинг 2.13).

Листинг 2.13 – Построение макроэлемента

```
reference ref;
//Построение элемента
//макроэлемент для операции симметрии относительно оси по 45 грд
```

```
Macro(0);
// большая дуга
ArcByPoint(0, 0, 42, con[0].x2, con[0].y2, 0, 42, 1, 1);
//дуга скругления
ArcByPoint(con[0].xc, con[0].yc, rad, con[0].x1, con[0].y1, con[0].x2,
con[0].y2, -1, 1);
//отрезок
LineSeg(con[0].x1, con[0].y1, x[1], y[1], 1);
//завершающая дуга радиусом 12 мм
ArcByPoint(41.012193, 41.012193, 12, x[1], y[1], xx[0], yy[0], -1, 1);
ref = EndObj();//закрытие макроэлемента
//симметрия относительно оси по 45 грд, исходный фрагмент оставляем
ksSymmetryObj(ref, 0, 0, 41.012193, 41.012193, 1);
```

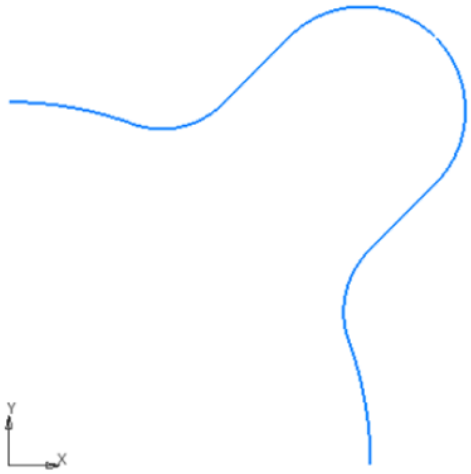



Рисунок 2.18 – Результат применения операции `ksSymmetryObj`

Получившийся фрагмент объединим в группу, чтобы иметь возможность применить операцию копирования. Тогда окончательный код построения элемента без осевых линий будет следующим (листинг 2.14).

Листинг 2.14 – Построения элемента без осевых линий

```
double rad = 10;
int kp;
CON con[8];
```

```
ksCouplingLineCircle(0, 0, 42, //параметры окружности
      0, 16.970563, 45, //параметры линии
      rad, &kp, con);
//поиск точки касания для определения второй координаты отрезка
double x[2], y[2];
TanLineAngCircle(41.012193, 41.012193, 12, 45, x, y);
//поиск точки пересечения осевой под 45 грд с дугой радиусом 12 мм
double xx[2], yy[2];
int k;
IntersectCirLin(41.012193, 41.012193, 12, 0, 0, 45, &k, xx, yy);
reference ref, gr;
//Построение элемента
//группа для хранения двух макроэлементов
gr = NewGroup(0);
//макроэлемент для операции симметрии относительно оси по 45 грд
Macro(0);
// большая дуга
ArcByPoint(0, 0, 42, con[0].x2, con[0].y2, 0, 42, 1, 1);
//дуга скругления
ArcByPoint(con[0].xc, con[0].yc, rad, con[0].x1, con[0].y1, con[0].x2,
con[0].y2, -1, 1);
//отрезок
LineSeg(con[0].x1, con[0].y1, x[1], y[1], 1);
//завершающая дуга радиусом 12 мм
ArcByPoint(41.012193, 41.012193, 12, x[1], y[1], xx[0], yy[0], -1, 1);
ref = EndObj();//закрытие макроэлемента
//симметрия относительно оси по 45 грд, исходный фрагмент оставляем
ksSymmetryObj(ref, 0, 0, 41.012193, 41.012193, 1);
EndGroup();//закрытие группы
//цикл построения остальных трех элементов с шагом 90 грд
//или копия по окружности
for (short i = 0; i < 360; i = i + 90)
      ksCopyObj(gr, 0, 0, 0, 0, 1, i);
```

В результате будет построена фигура, приведенная на рисунке 2.19.

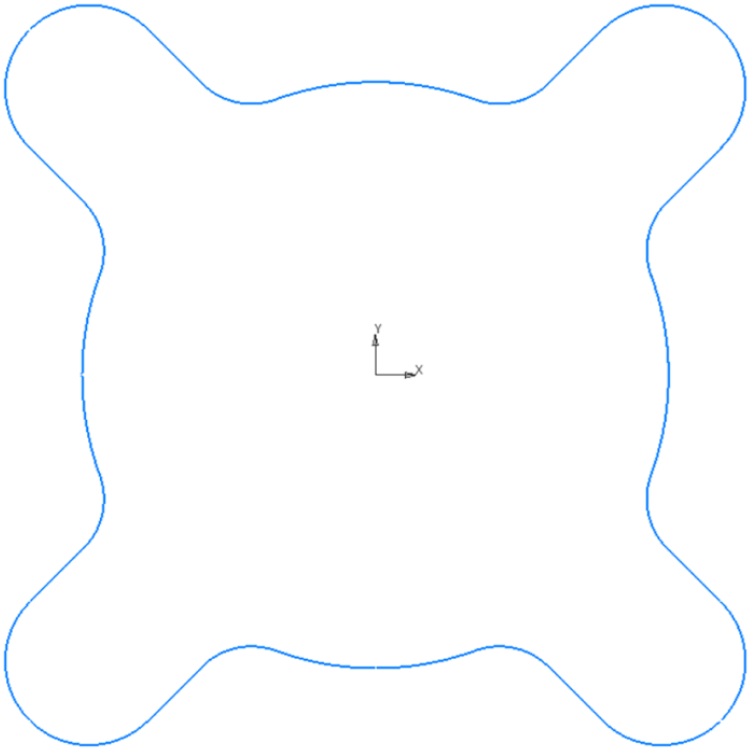


Рисунок 2.19 – Построение элемента без осевых линий

Выполним заполнение штриховкой (рисунок 2.20) внутренней области, используя функцию `ksMakeEncloseContours` (листинг 2.15)

Листинг 2.15 – Штриховка внутренней области элемента

```
HatchParam par;
memset(&par, 0, sizeof(par));
par.step = 2;
par.style = 0;
par.ang = 45;
//вычисляем границы штриховки, указав базовую точку с координатами 0, 0
par.pBoundaries = ksMakeEncloseContours(0, 0, 0); //ksMakeEncloseContours - Определить группу объектов, охватывающих заданную точку
//штриховка
ksHatch(&par);
```

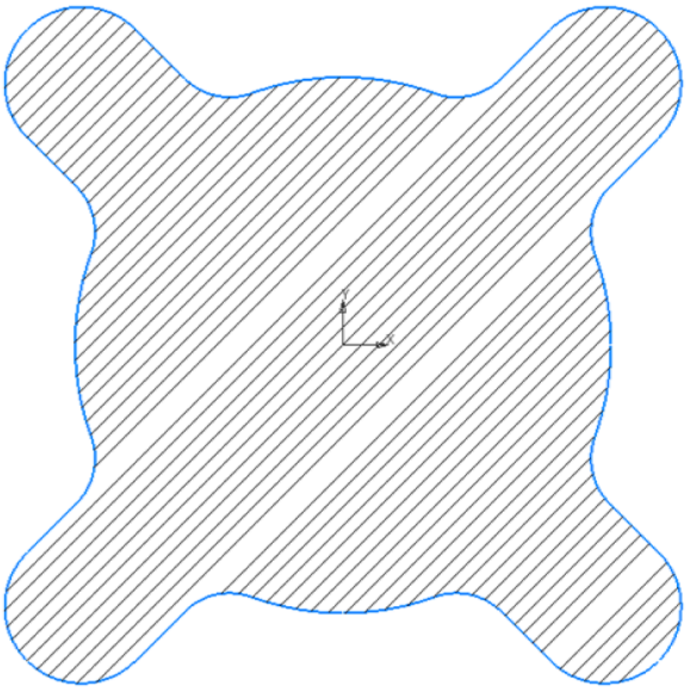


Рисунок 2.20 – Элемент со штриховкой

### 2.5 Задание для лабораторной работы

1 Изучить теоретический материал.

2 Настроить проект в MS Visual Studio для корректной компиляции прикладной библиотеки КОМПАС.

3 Разработать прикладную библиотеку КОМПАС которая строит чертеж детали по заданному варианту, приведенному в приложении Б.

4 Подготовить ответы на контрольные вопросы.

### 2.6 Содержание отчета

В отчете по лабораторной работе согласно СТО 02069024.101–2015 РАБОТЫ СТУДЕНЧЕСКИЕ [3] должны содержаться следующие пункты:

– название лабораторной работы;

– цель работы;

– задание на лабораторную работу;

– код разработанной программы с комментариями;

– экранные формы с отображением результата выполнения программы в системе КОМПАС;

– выводы;

– список использованных источников.

### 2.7 Контрольные вопросы

1 Какие примитивы можно отнести к простым, а какие к составным?

2 Опишите принцип построения составного примитива.

3 Как получить ссылку на дескриптор составного примитива?

4 Приведите пример использования функции ksPolyline.

5 Опишите функцию Macro.

6 Как задать матрицу преобразования координат?

7 Как преобразовать объект по установленной матрице?

8 Опишите функцию IntersectLinLin.

9 Опишите функцию CouplingLineLine.

10 Опишите структуру CON.

11 Как найти в массиве CON индекс заданного сопряжения?

12 Как определить точки сопряжения дуг?

13 Для чего используется условие if ((i == 4) || (i == 5)) и что оно означает?

14 Опишите команду TanLineAngCircle.

15 Что такое Группа?

16 Опишите команду ksSymmetryObj.

17 Опишите команду ksCopyObj.

18 Приведите пример построения симметрии.

19 Приведите пример копирования по линии.

20 Как выполняется копирование по окружности?

21 Опишите команду ksMakeEncloseContours.

22 Опишите структуру HatchParam.

# 3 Модель графического документа. Структура, файлы, виды, слои

## 3.1 Структура графического документа

К графическим документам в КОМПАС принято относить листы чертежа и фрагменты. Эти документы обрабатываются подсистемой КОМПАС-График – с ее помощью пользователь в интерактивном режиме добавляет в документ геометрические элементы, редактирует или удаляет их. Кроме геометрических элементов, на листе чертежа могут располагаться элементы оформления (например, штамп и технические требования).

В чертеже или фрагменте может быть любое количество геометрических элементов. На чертежах сложных изделий их может быть очень много, но конструктор обычно может выполнить логическое объединение этих элементов в более крупные совокупности (например, детали или их части). Может быть удобно рассматривать эти совокупности элементов как самостоятельные компоненты чертежа - например, чтобы перемещать изображения деталей на чертеже, включать или выключать их отображение, переносить в другие чертежи или сохранять в отдельных файлах. Для подобных задач, т.е. для удобного редактирования чертежей, в модели документа КОМПАС предусмотрены компоненты для логического объединения элементов чертежа:

– виды;

– слои;

– группы.

Основным документом в подсистеме КОМПАС-График является лист чертежа (далее – просто чертеж). Чертежи хранятся в отдельных файлах специального формата с расширением .CDW. Чертежи сложных изделий могут состоять из нескольких листов, тогда эти листы создаются и обрабатываются отдельно (в различных файлах).

Каждый чертеж (рисунок 3.1) состоит из набора визуально различимых компонент: видов, технических требований, основной надписи (штампа чертежа), спе-

цификации и обозначения шероховатости неуказанных поверхностей детали (знака неуказанной шероховатости). На конкретном чертеже часть перечисленных компонент может отсутствовать.

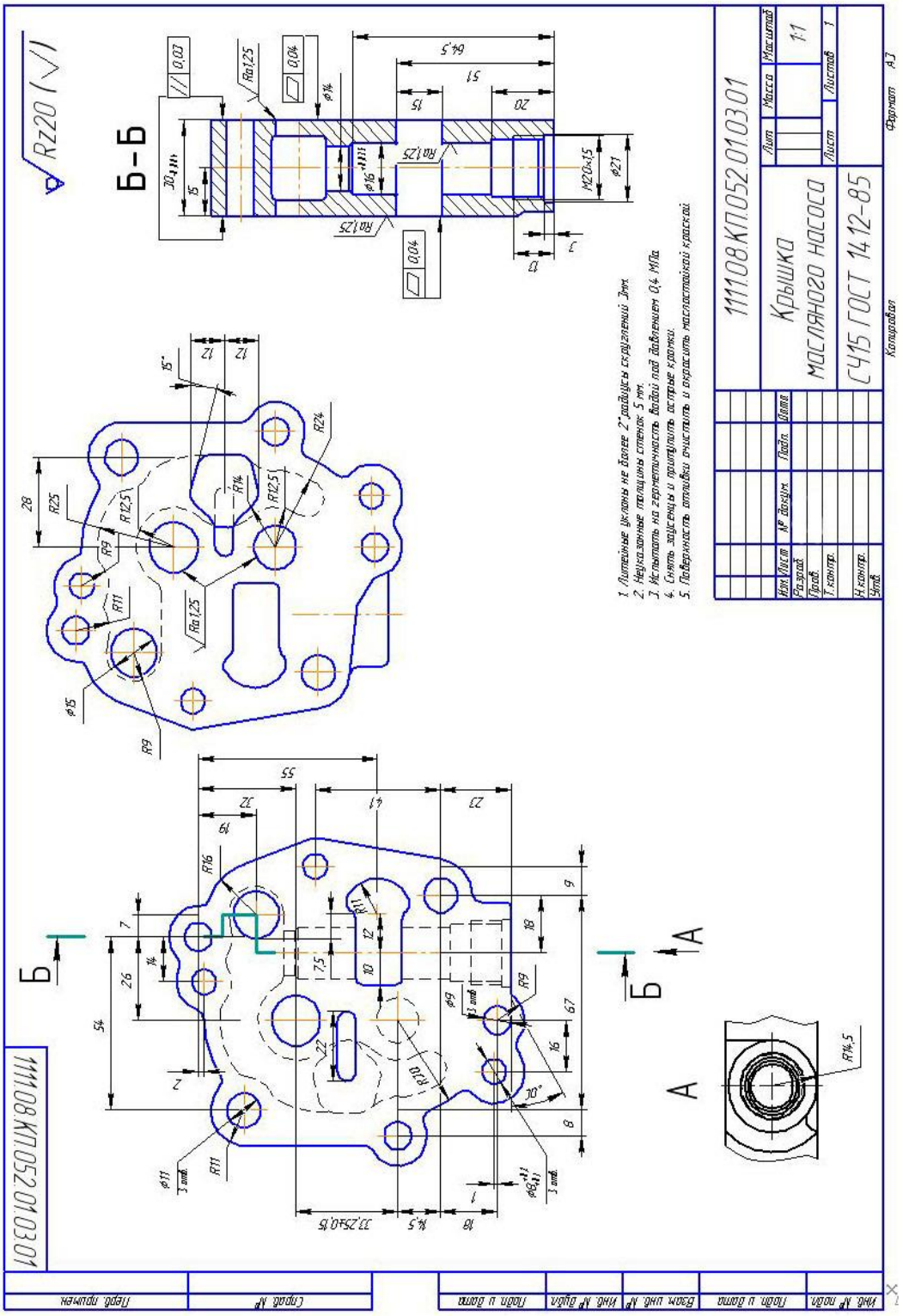



Рисунок 3.1 – Пример чертежа

Видом называется любое изолированное изображение на чертеже. Например, это могут быть различные проекции детали (вид слева, вид спереди, вид сверху). Этим случаем использование видов не ограничено. Для удобства редактирования чертежа на нем можно создать любое количество видов. Фактически, вид в КОМПАС-График – это отдельная система координат, заданная относительно системы координат чертежа («абсолютной системы координат»). Абсолютная система координат – правая система координат (ось X направлена вправо, ось Y – вверх) с началом отсчета в левом нижнем углу чертежа. Положение каждого вида в абсолютной системе координат определяется точкой привязки (координатами начала отсчета вида), углом поворота и масштабом.

В принципе, все изображение на чертеже может быть создано в одном виде. При создании нового чертежа автоматически создается специальный системный вид с номером 0, и пользователь может немедленно приступать к вычерчиванию элементов, которые будут помещаться в этот нулевой вид.

В каждом виде можно создавать до 255 слоев для удобного размещения и обработки изображения. Все слои принадлежат виду. Вид можно переместить на чертеже, и все его слои переместятся вместе с ним.

Слой можно назвать уровнем, на котором размещена часть элементов фрагмента или вида чертежа. Слои применяются во многих графических программах, по аналогии с созданием изображения путем наложения друг на друга нескольких прозрачных листов (при обычном проектировании на кульмане используются накладываемые друг на друга кальки).

Например, при рисовании чертежа здания в отдельных слоях можно разместить строительные детали, схемы газо-, водоснабжения и электропроводки. При редактировании схемы электропроводки можно отключить отображение остальных схем, чтобы не загромождать чертеж ненужными в данный момент деталями.

Каждый вид и слой может иметь уникальное название для облегчения поиска и выбора. Виды и слои могут находиться в различных состояниях (текущий, фоновый, невидимый).

Модель чертежа можно представить с помощью иерархической структурной схемы (рисунок 3.2), на которой показана принадлежность компонент. Числа около стрелки обозначают кратность отношения принадлежности (одному виду может принадлежать от 1 до 255 слоев; технические требования присутствуют на чертеже либо в единственном экземпляре, либо отсутствуют вообще; штамп на чертеже есть обязательно, и только один).

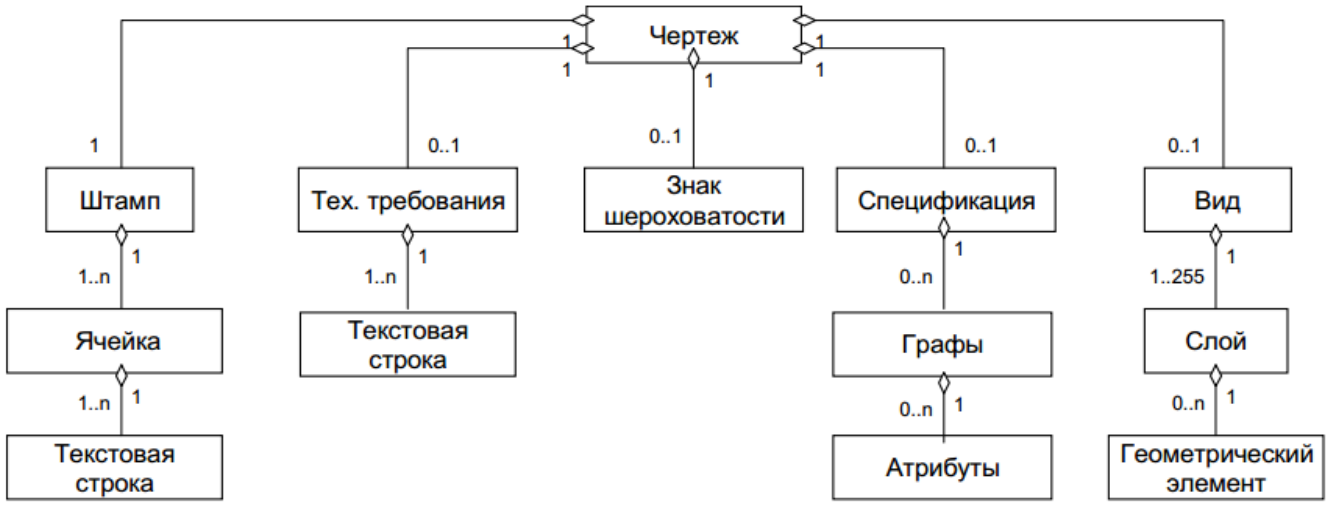



Рисунок 3.2 – Модель чертежа

Вторая разновидность графических документов КОМПАС-График – Фрагмент. Он отличается от чертежа отсутствием компонент оформления. Во фрагменте нет рамки, основной надписи, знака шероховатости и технических требований. Вид во фрагменте только один, но, как и вид чертежа, он может содержать до 255 слоев.

Начало абсолютной системы координат чертежа всегда находится в левой нижней точке габаритной рамки формата. Начало системы координат фрагмента не имеет такой четкой привязки, как в случае чертежа. Поэтому, когда открывается новый фрагмент в окне КОМПАС-График, точка начала его системы координат автоматически отображается в центре окна. Фактически фрагмент можно считать видом чертежа, сохраненным в отдельном файле (с расширением .frw).

Фрагмент идеально подходит для хранения изображений, которые не нужно оформлять как лист чертежа (эскизные прорисовки, разработки и т.д.). Кроме того,

во фрагментах удобно сохранять созданные типовые решения и конструкции для последующего использования в других документах. КОМПАС-График позволяет ссылаться на внешний фрагмент без его физического копирования в документ, при этом после редактирования внешнего фрагмента автоматически будет откорректирован и документ, в который вставлен этот фрагмент.

## 3.2 Операции с чертежом, видами и слоями

КОМПАС-МАСТЕР содержит обширные функции, предоставляющие прикладному программисту доступ к модели графического документа.

В нем также есть функция для получения дескриптора любого компонента чертежа – `ksGetReferenceDocumentPart`.

Функции для работы с компонентами чертежа эквивалентны командам меню КОМПАС-График, подобно тому, как функции для рисования геометрических элементов аналогичны командам геометрических построений КОМПАС-График.

### 3.2.1 Операции с файлами документов

Чертеж является разновидностью документа КОМПАС, а для всех документов предусмотрены одни и те же функции (таблица 3.1).

Специфика документа проявляется при его создании, поэтому функция `CreateDocument` является наиболее сложной в использовании. При создании документа требуется указать его параметры с помощью интерфейса `DocumentParam`.

Кроме перечисленных функций, к функциям работы с документами можно отнести еще одну функцию для открытия окна просмотра перед печатью - `ksPrintPreviewWindow`.

Таблица 3.1 – Функции для работы с файлами документов

| Имя функции | Назначение |
|---|---|
| OpenDocument | Открыть документ (чертеж, фрагмент, спецификацию, текстовый документ, деталь или сборку) |
| SaveDocument | Сохранить документ |
| CloseDocument | Закрыть документ |
| CreateDocument | Создать документ |

Ниже приведен пример процедуры (листинг 3.1), в которой демонстрируется создание нового чертежа, выполнение в нем геометрических построений и сохранение файла.

Листинг 3.1 – Создание нового чертежа

```
DocumentParamT parDocument; // Структура параметров документа
lstrcpy(parDocument.comment, _T("Документ, созданный программно")); // Комментарий к документу
lstrcpy(parDocument.author, _T("Сергеев А.И."));   // Автор документа
parDocument.regim = 0;    // Режим ( 0 - видимый, 1 - слепой )
parDocument.type = 1;    // Тип документа
parDocument.sheet.stPar.format = 4;    // Формат листа  0 (A0) ... 4(A4)
parDocument.sheet.stPar.multiply = 1;    // Кратность формата
parDocument.sheet.stPar.direct = 0;    // Расположение штампа ( 0 - вдоль короткой стороны, 1 - вдоль длинной )
parDocument.sheet.shtType = 1;    // Тип штампа из указанной библиотеки
parDocument.sheet.layoutName[0] = _T('\0'); // Имя библиотеки оформления, пустая строка - библиотека "Graphic.lyt"
// Создать документ КОМПАС-ГРАФИК
lstrcpy(parDocument.fileName, _T("c:\\a.cdw")); // Имя  файла документа
if (CreateDocumentW(&parDocument))
{
      // Перенос системы координат из левого нижнего угла в центр листа
      //   Размеры листа  A4   -   210x2 97   мм
      ksMtr(210.0 / 2, 297.0 / 2, 0, 1.0, 1.0);
      // Пример операции рисования
```

```
        Circle(0, 0, 25, 1);
        // Восстановление системы координат
        DeleteMtr;
        // Сохранение документа (NULL означает использование
        // имени файла из  документа)
        SaveDocument(0, NULL);
}
```

### 3.2.2 Операции с видами

Среди функций работы с видами есть функции для преобразования координат точек, функции для манипуляций с видами по номерам и дескрипторам, и др. (таблица 3.2).

Таблица 3.2 – Функции для работы с видами

| Имя функции | Назначение |
|---|---|
| CreateSheetView | Создать новый вид в чертеже |
| GetViewReference | Получить указатель на вид по номеру вида |
| GetViewNumber | Получить номер вида по указателю на вид или объект вида |
| OpenView | Сделать текущим существующий вид с указанным номером |
| NewViewNumber | Определить номер следующего вида (т.к. при создании нового вида требуется указывать его номер) |
| GetViewObjCount | Получить количество объектов вида |
| SheetToView | Пересчитать координаты точки из СК листа в СК текущего вида |
| ksViewToSheet | Пересчитать координаты точки из СК текущего вида в СК листа |

В следующем примере (листинг 3.2) показано создание нового вида в текущем документе (это обязательно должен быть лист чертежа, иначе КОМПАС-График выдаст сообщение об ошибке) и выполнение простого построения в этом виде.

Листинг 3.2 – Создание нового вида в текущем документе

```
//Работа с видами
ViewParamT par; // Структура параметров вида
par.x = 10; // Точка привязки вида
par.y = 20;
par.scale = 1;  // Масштаб вида
par.ang = 0;  // Угол поворота вида
par.color = RGB(10, 20, 10);   // Цвет вида в активном состоянии
par.state = stACTIVE;            // Состояние вида
_tcscpy_s(par.name, _T("Вид программно")); // Имя вида
// Создадим вид
int i = 1;
reference rView = CreateSheetViewT(&par, &i); // Создание нового вида черте-
жа, если вид с данным
 // номером существует, ничего не создается
//пример создания объекта
Circle(40, 20, 30, 1);
// Получить параметры вида
GetObjParam(rView, &par, sizeof(par), ALLPARAM_T);
i = GetViewNumber(rView);
TCHAR buf[255];
_stprintf_s(buf, _T("x = %4.2f, y = %4.2f, scale = %4.2f, ang = %4.2f, color
= %x, state = %d, name = %s, number = %d"), par.x, par.y, par.scale, par.ang,
par.color, par.state, par.name, i);
MessageT(buf);
// Сделать текущим системный вид ( номер 0 )
OpenView(0);
MessageT(_T("Изменить состояние вида"));
// Изменить состояние вида ( только чтение )
int state = stREADONLY;
SetObjParam(rView, &state, sizeof(state), VIEW_LAYER_STATE);
```

### 3.2.3 Операции со слоями

В таблице 3.3 описаны функции для работы со слоями.

Таблица 3.3 – Функции для работы со слоями

| Имя функции | Назначение |
|---|---|
| Layer | Сделать слой с заданным номером текущим слоем (если слоя с таким номером нет, он будет создан) |
| GetLayerReference | Получить указатель на слой по номеру слоя текущего вида |
| GetLayerNumber | Получить номер слоя по его дескриптору |
| ChangeObjectLayer | Изменить слой объекта |

В качестве примера операций со слоями далее показано выполнение следующих действий (листинг 3.3):

– создание в текущем виде слоя с номером, введенным пользователем;

– выделение слоя;

– изменение имени, цвета и состояния слоя.

Листинг 3.3 – Операции со слоями

```
//Работа со слоями
int number;
if (ReadIntT(_T("Введите номер создаваемого слоя"), 1, 0, 255, &number))
{
	// Переопределение текущего слоя, если слоя с заданным номером
	// нет, такой слой создается. Слой становится текущим
	reference rLayer = Layer(number);
	Mtr(20, 15, 0, 1); // Матрица преобразования координат (20 по оси OX, 15 по оси OY)
	LineSeg(-10, 0, 10, 0, 1);
	LineSeg(10, 0, 10, 20, 1);
	LineSeg(10, 20, -10, 20, 1);
	LineSeg(-10, 20, -10, 0, 1);
	DeleteMtr(); // Отключение матрицы преобразования координат
```

```
	LightObj(rLayer, 1);  // Подсветить слой
	// Получить номер слоя по указателю и указатель по номеру
	number = GetLayerNumber(rLayer);    // Получить номер слоя по указателю
	rLayer = GetLayerReference(number); // Получить указатель слоя по номеру
	LayerParamT parLayer; // Структура параметров слоя
	memset(&parLayer, 0, sizeof(parLayer));
	// Получить параметры слоя
	GetObjParam(rLayer, &parLayer, sizeof(parLayer), ALLPARAM_T);
	TCHAR buf[255];
	_stprintf_s(buf, _T("number = %d, rLayer = %d,\nstate = %d, color = %x,
name = %s"), number, rLayer, parLayer.state, parLayer.color, parLayer.name);
	MessageT(buf);
	// Установить параметры слоя
	parLayer.color = RGB(0, 255, 0);   // Цвет слоя в активном состоянии
	parLayer.state = stACTIVE;           // Состояние слоя
	_tcscpy_s(parLayer.name, _T("Зеленый")); // Имя слоя
	MessageT(_T("Изменим параметры слоя"));
	// Параметры текущего слоя изменить нельзя поэтому переключаемся на ну-
левой слой
	Layer(0);
	if (!SetObjParam(rLayer, &parLayer, sizeof(parLayer), ALLPARAM_T))
		MessageBoxResult(); // Выдать сообщение об ошибке на экран
	else {
		// Получить параметры слоя
		GetObjParam(rLayer, &parLayer, sizeof(parLayer), ALLPARAM_T);
		_stprintf_s(buf, _T("number = %d, rLayer = %d,\nstate = %d, color =
%x, name = %s"), number, rLayer, parLayer.state, parLayer.color, parLayer.name);
		MessageT(buf);
	}
	// Снять выделение слоя
	LightObj(rLayer, 0);
	MessageT(_T("Изменить состояние слоя"));
	// Изменить состояние слоя ( активизировать слой )
	int state = stACTIVE;
	SetObjParam(rLayer, &state, sizeof(state), VIEW_LAYER_STATE);
}
```

### 3.2.4 Группы элементов чертежа

Группа – это объединение логически связанных между собой элементов чертежа для их удобного одновременного поиска и редактирования. Группа может иметь название, по которому ее можно выбирать в списке групп.

Элементы, входящие в группу, пользователь может выделять и редактировать по отдельности. Один и тот же элемент чертежа может одновременно входить в несколько разных групп. Группа может объединять элементы, принадлежащие различным слоям и видам чертежа, а также отдельные виды, технические требования и основную надпись штампа или ее отдельные графы.

Для работы с группами в меню КОМПАС-График предназначены команды Черчение → Группы (эта команда входит и в контекстное меню выделенного элемента) и Выделить → По свойствам → Группы.

С точки зрения прикладного программиста, группа очень похожа на составной элемент чертежа. Формирование группы начинается с вызова функции `NewGroup`, которой указывается тип группы - модельный или временный. В отличие от большинства составных элементов, формирование группы завершается вызовом функции `EndGroup()`, а не `EndObj()` (т.к. в группу могут входить составные элементы). При создании новой группы необязательно закрывать предыдущую (т.е. поддерживается вложенность групп).

После вызова `EndGroup()` элементы модельной группы помещаются на чертеж, а элементы временной группы запоминаются во временном списке и на экране не отображаются. Для дальнейшей обработки группы используются те же функции редактирования, что и для отдельных элементов, так как дескриптор группы ничем не отличается от дескрипторов остальных элементов чертежа.

В КОМПАС-МАСТЕР принято соглашение, по которому дескриптор 0 имеет служебная группа селектирования, содержащая все выделенные в данный момент элементы чертежа.

Элементы временной группы сохраняются только до завершения работы создающей их библиотечной функции. Временная группа служит чаще всего для от-

рисовки фантомного изображения - например, когда пользователь добавляет на чертеж какую-либо деталь из библиотеки, то в процессе размещения этой детали на чертеже показывается ее фантомное изображение, состоящее из сплошных однотипных линий. После успешного завершения команды вставки детали вместо фантомного изображения на чертеже создается полное изображение детали. В таблице 3.4 описаны функции для работы с группами.

Таблица 3.4 – Функции для работы с группами

| Имя функции | Назначение |
|---|---|
| NewGroup | Создать новую группу элементов чертежа |
| EndGroup | Завершить создание группы |
| AddObjGroup | Добавить объект в группу |
| ksClearGroup | Очистить группу |
| ksDrawKompasGroup | Нарисовать группу как слайд в заданном окне Windows |
| ExcludeObjGroup | Исключить элемент из группы |
| ExistGroupObj | Проверить, не является ли группа пустой |
| GetGroup | Получить указатель на группу по ее имени |
| ksGetGroupName | Получить имя группы по ее дескриптору |
| ksReadGroupFromClip | Получить элементы чертежа из буфера обмена и поместить их во временную группе |
| SaveGroup | Сохранить модельную группу в документе (преобразовать ее в именованную группу) |
| SelectGroup | Сформировать группу из элементов, попадающих в заданный габаритный прямоугольник |
| StoreTmpGroup | Вставить временную группу в документ |
| ksViewGetObjectArea | Сформировать временную группу из элементов, определяющих область выделения, в интерактивном режиме |
| ksWriteGroupToClip | Поместить группу в буфер обмена (скопировать или вырезать) |

3.2.5 Работа с именованной группой

На уровне функций КОМПАС-МАСТЕР для создания именованной группы приходится выполнять два действия:

– создать модельную группу (модельные группы удаляются после завершения работы функции RTW-библиотеки);

– сохранить модельную группу в документе – при этом указывается имя группы, и она становится именованной и будет включена в список групп (например, вызываемый командой Выделить → По свойствам →Группы).

Далее показан пример (листинг 3.4), в котором демонстрируется создание именованной группы, добавление элемента в уже созданную группу и геометрическое преобразование группы как обычного элемента чертежа.

Листинг 3.4 – Работа с группами

```
//Работа с группой
// Создание группы объектов, type - тип группы ( 0 - определяет модельный, 1
- временный )
ksMtr(210.0 / 2, 297.0 / 2, 0, 1.0, 1.0);
reference rGroup = NewGroup(0);
LineSeg(20, 20, 40, 20, 1);
LineSeg(40, 20, 40, 40, 1);
LineSeg(40, 40, 20, 40, 1);
LineSeg(20, 40, 20, 20, 1);
EndGroup(); // Завершить создание группы объектов
DeleteMtr();
// Сохранить группу объектов в модели с указанным именем
// Группа автоматически сохраняется в чертеже при его записи
// В противном случае группа действительна только в текущем сеансе работы
// Если указатель группы равен нулю, то сохраняется группа селектирования
// КОМПАС-ГРАФИК (выделенные объекты чертежа).
if (!SaveGroupT(rGroup, _T("Программная группа")))
      exit(EXIT_SUCCESS);
rGroup = GetGroupT(_T("Программная группа"));
if (!rGroup)
      exit(EXIT_SUCCESS);
// Создание окружности
```

```
reference rCircle = Circle(30, 30, 10, 1);
// Добавить новый объект в группу
AddObjGroup(rGroup, rCircle);
LightObj(rGroup, 1);
MessageT(_T("Был добавлен объект в именную группу"));
LightObj(rGroup, 0);
MoveObj(rGroup, 10, 0);
MessageT(_T("Сдвинули группу на 10 мм"));
RotateObj(rGroup, 20, 10, 45);
MessageT(_T("Повернули группу на 45 гр"));
// Исключить объект из группы
ExcludeObjGroup(rGroup, rCircle);
LightObj(rGroup, 1);
MessageT(_T("Объект был исключен из именной группы"));
LightObj(rGroup, 0);
```

### 3.3 Перебор элементов чертежа. Итератор

В модели графического документа предполагается, что чертеж состоит из элементов различных типов - видов, геометрических элементов, компонент оформления и др. Может быть несколько разновидностей элементов одного типа, например, существует много разных геометрических элементов. При создании элементов чертежа информация о них запоминается в модели документа.

В прикладной библиотеке может возникнуть необходимость обработки всех элементов заданного типа, или поиск одного из элементов заданного типа. Для хранения информации произвольного объема могут применяться динамические массивы. В принципе, можно было бы предположить наличие функций для загрузки информации об элементах заданного типа в динамический массив. Но это связано с копированием информации, а элементов может быть много – например, отрезков на сложном чертеже может оказаться несколько тысяч. Поэтому в КОМПАС-МАСТЕР есть средства для перебора элементов непосредственно в модели чертежа, без копи-

рования информации в буферные промежуточные массивы. Этими средствами являются итераторы.

Итератором называется вспомогательный объект КОМПАС-МАСТЕР, предназначенный для программного перебора списка однотипных элементов чертежа.

Стиль работы с итераторами напоминает работу с динамическими наборами в базах данных: итератор можно создать, перемещаться по нему в одном направлении, а после использования – закрыть. При создании итератора указывается, для какого типа элементов он создается или код `ALL_OBJ` для перебора элементов любого типа. Итератор привязывается к одному из компонент чертежа – виду, штампу, документу. Один и тот же итератор для перемещения по элементам различных компонент чертежа использовать нельзя. При каждом смещении итератора он возвращает либо дескриптор очередного элемента, либо 0 в случае, если таких элементов больше нет. Итератор можно сбросить в начальную позицию и снова перемещаться по нему в одном направлении, поместить в начало.

Перебирать можно документы, виды, различные геометрические элементы и вообще любые элементы, у которых есть идентификатор.

Функции для работы с итераторами приведены в таблице 3.5. Для большинства элементов чертежа (в том числе геометрических элементов, видов, фрагментов, всех элементов, для которых в КОМПАС-МАСТЕР определен код типа элемента) предназначены итераторы, создаваемые функцией `CreateIterator`. Для перебора атрибутов (программно создаваемых характеристик элементов), квалитетов и спецификации предназначены итераторы особого типа.

Рассмотрим пример перебора всех элементов чертежа, содержащихся в текущем виде текущего документа (листинг 3.5). Процедура `Demo_AllObjsIterator` создает итератор для перебора элементов любого типа (код `ALL_OBJ`). В цикле она выполняет проверку существования очередного элемента, дескриптор которого возвратил итератор, и выделение этого элемента цветом. В функцию перемещения итератора `MoveIterator` передается либо строка 'F' для перемещения итератора в начало списка, либо 'N' для перемещения к следующему элементу списка.

Таблица 3.5 – Функции навигации по объектам документа

| Имя функции | Назначение |
|---|---|
| CreateIterator | Создание итератора |
| MoveIterator | Перемещение итератора |
| CreateAttrIterator | Создать итератор для перебора атрибутов объекта |
| MoveAttrIterator | Перемещение итератора по атрибутам объекта |
| CreateSpcIterator | Создать итератор для перебора объектов спецификации |
| ksCreateQualityIterator | Создать итератор для перебора квалитетам (параметрам, указывающим качество обработки поверхностей) |
| ksMoveQualityIterator | Перемещение итератора квалитетов |
| DeleteIterator | Удаление итератора |

Листинг 3.5 – Перебор всех элементов чертежа

```
void Demo_AllObjsIterator()
{
	reference itAllObj;
	reference obj;
	int count = 0;
	char buf[128];
	//Создать итератор
	itAllObj = CreateIterator(ALL_OBJ, 0);
	if (itAllObj)
	{
		if (ExistObj(obj = MoveIterator(itAllObj, 'F')))
		{
			// поиск первого объекта и следующего в цикле
			do {
				LightObj(obj, 1);
				count++;
				sprintf_s(buf, "номер= %d", count);
				Message(buf);
				LightObj(obj, 0);
```

```
            } while (ExistObj(obj = MoveIterator(itAllObj, 'N')));
        }
    }
    //Удалить итератор
    DeleteIterator(itAllObj);
}
```

Элементы могут быть контейнерными, т.е. содержать внутри себя другие элементы. Примерами контейнеров являются виды, слои, группы, макроэлементы.

При переборе элементов внутри их контейнера, например, элементов, входящих в группу, при создании итератора требуется указать и тип перебираемых элементов, и дескриптор элемента-контейнера. Ниже показана процедура `WalkFromGroup()`, выполняющая создание модельной группы и перебор элементов, входящих в эту группу (листинг 3.6). В процессе перебора элементы подсвечиваются и в окне сообщения выводится их порядковый номер.

Листинг 3.6 – Перебор групп

```
// Хождение по группе
// ---
void WalkFromGroup()
{
    // Создание группы объектов, type - тип группы (0 - определяет модельный, 1 - временный)
    reference rGroup = NewGroup(0);
    LineSeg(10, 50, 50, 50, 1);
    LineSeg(10, 10, 50, 10, 1);
    LineSeg(10, 10, 10, 50, 1);
    LineSeg(50, 10, 50, 50, 1);
    Circle(30, 30, 20, 1);
    Circle(30, 30, 10, 1);
    // Штриховка окружностей
    Hatch(0, 45, 2, 0, 0, 0);
    Circle(30, 30, 20, 1);
    Circle(30, 30, 10, 1);
```

```
	EndObj();
	EndGroup(); // Завершить создание группы объектов
	// Создать итератор для хождения по группе
	reference rItObject = CreateIterator(ALL_OBJ,  // Тип поиска объекта
		rGroup); // Указатель на объект (для движения по группе и внутри макро)
	if (rItObject)
	{
		int count = 0;
		reference rObject = MoveIterator(rItObject, 'F'); // Переместить
итератор ( F - на первый объект заданного типа )
		TCHAR buf[255];
		while (rObject)
		{
			LightObj(rObject, 1); // Подсветить полученный объект
			MessageT(_T("Снять выделение объекта"));
			LightObj(rObject, 0);
			count++;
			rObject = MoveIterator(rItObject, 'N'); // Переместить итера-
тор ( N - на следущий объект заданного типа )
			_stprintf_s(buf, _T("Номер объекта группы = %d"), count);
			MessageT(buf);
		}
		DeleteIterator(rItObject); // Удалить блок параметров навигации по
модели
	}
}
```

Часто перебор элементов, расположенных в графическом документе, связан с определением типа элемента и его свойств. В некоторых случаях необходимо для заданных элементов изменить параметры. Процедура `Demo_TypeObjsIterator()` определяет тип элемента: отрезок или окружность, выводит сообщения об их параметрах, затем для отрезков изменяет координаты начальной точки, для окружностей

– радиус, для всех элементов стиль линии устанавливается «Основная» (листинг 3.7).

Листинг 3.7 – Работа с типами элементов

```
// Работа с параметрами объектов в зависимости от их типа
void Demo_TypeObjsIterator()
{
	reference itTypeObj;
	reference obj;//указатель на объект
	int count = 0;
	char buf[128];
	reference objType = 0; //тип объекта
	LineSegParam par;
	reference t;
	CircleParam parCircle;
	//Создать итератор
	itTypeObj = CreateIterator(SELECT_GROUP_OBJ, 0);
	if (itTypeObj) {
		if (ExistObj(obj = MoveIterator(itTypeObj, 'F')) == 0) {
			MessageT(_T("Нет выделенного объекта!"));
		}
		else
		{// поиск первого объекта и следующего в цикле
			do {
//Вызов функции с нулевыми значениями параметров param, parSize и parType
			//позволяет получить тип объекта по его reference.
				objType = GetObjParam(obj, 0, 0, 0);
				LightObj(obj, 1); // подсветка объекта
				// если объект отрезок
				if (objType == LINESEG_OBJ)
				{
			//получаем его параметры в структуру параметров отрезка par
					t = GetObjParam(obj, &par, sizeof(par), ALLPARAM);
					//формируем строку вывода сообщения для вывода
					sprintf_s(buf, "Тип объекта - Отрезок, x1=%4.1f
y1=%4.1f x2=%4.1f y2=%4.1f Стиль=%d", par.x1, par.y1, par.x2, par.y2,
par.style);
```

```
                        Message(buf);
                        //изменяем параметры отрезка
                        par.x1 = 0; par.y1 = 0; par.style = 1;
                        //задаем новые параметры
                        if (SetObjParam(obj, &par, sizeof(par), ALLPARAM))
                              Message(" Изменили отрезок");
                        else MessageBoxResult();
                    } //если объект окружность
                    if (objType == CIRCLE_OBJ)
                    {//получаем ее параметры в структуру параметров окруж-
ности parCircle
                        t = GetObjParam(obj, &parCircle, sizeof (parCircle),
ALLPARAM);
                        sprintf_s(buf, "Тип объекта - Окружность, xc=%4.1f
yc=%4.1f rad=%4.1f Стиль=%d",parCircle.xc, parCircle.yc, parCircle.rad,
parCircle.style);
                        Message(buf);
                        //изменяем параметры окружности
                        parCircle.rad = 30; parCircle.style = 1;
                        //задаем новые параметры
                if (SetObjParam(obj, &parCircle, sizeof(parCircle), ALLPARAM))
                              Message(" Изменили окружность");
                        else MessageBoxResult();
                    }
                    count++;
                    sprintf_s(buf, "номер= %d", count);
                    Message(buf);
                    LightObj(obj, 0);
                } while (ExistObj(obj = MoveIterator(itTypeObj, 'N')));
            }
        }
        //Удалить итератор
        DeleteIterator(itTypeObj);
}
```

Результат работы программы показан на рисунках 3.3, 3.4.

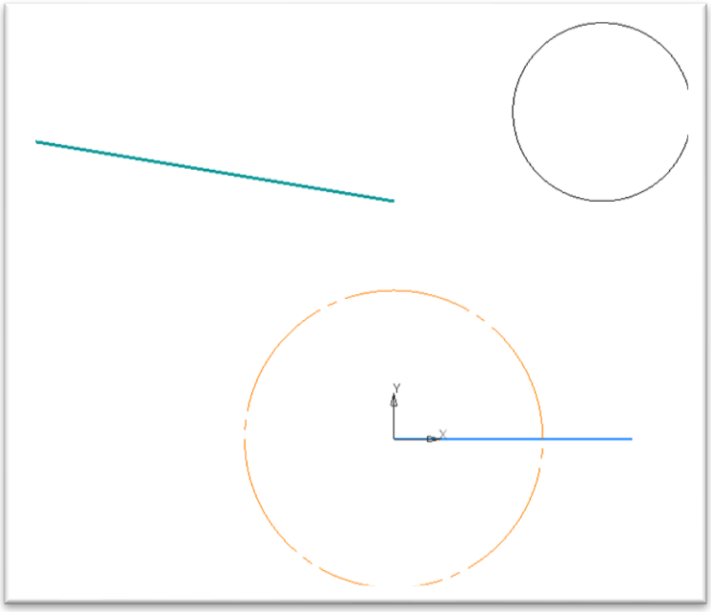


Рисунок 3.3 – Исходное положение объектов документа

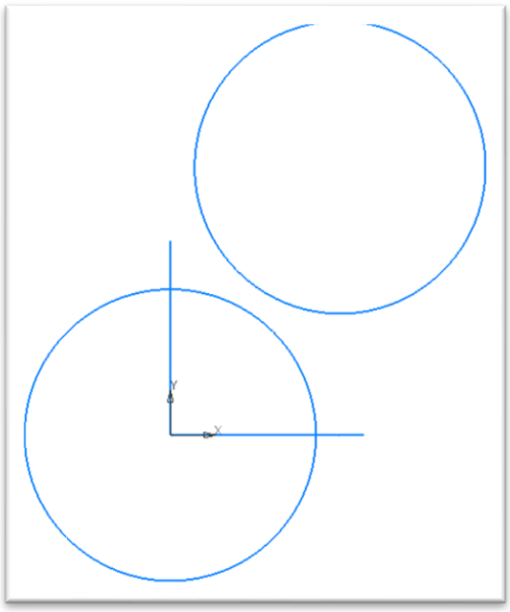


Рисунок 3.4 – Положение объектов после изменения параметров

В процессе работы программы выводятся сообщения с информацией об элементах (рисунок 3.5).

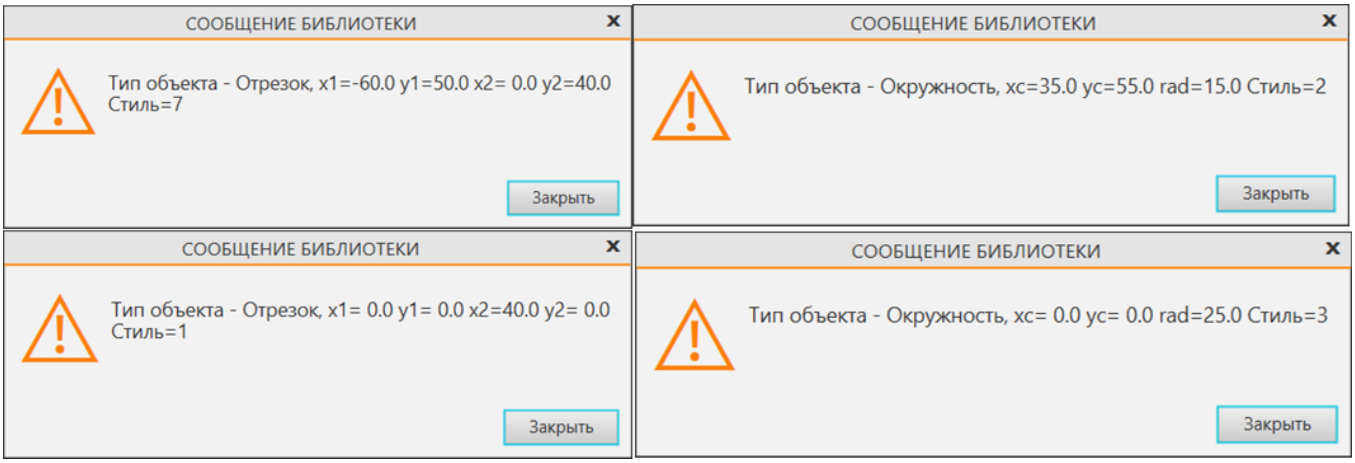



Рисунок 3.5 – Сообщения с параметрами графических примитивов

## 3.4 Компоненты оформления чертежа

К компонентам оформления чертежа можно отнести штамп, технические требования, знак неуказанной шероховатости и спецификацию.

Спецификация будет рассматриваться отдельно.

Все эти компоненты содержат текстовую и числовую информацию, необходимую для однозначной интерпретации данного чертежа и изготовления изображенного на нем изделия.

### 3.4.1 Штамп

В штампе (основной надписи) размещаются общие сведения о чертеже – название; фамилия конструктора, подготовившего чертеж; дата изготовления чертежа; количество листов, если их несколько, и т.п. Визуально штамп похож на таблицу, в столбцах которой размещено неодинаковое количество ячеек. К штампу относится также рамка вокруг листа чертежа и ячейки, размещенные на этой рамке.

Пользователь в КОМПАС-График может ввести текст в любую ячейку штампа. Сначала штамп надо открыть вызовом `OpenStamp()`. После этого можно выбирать ячейку штампа по номеру вызовом `ColumnNumber()`. Номера ячеек определены в соответствии с ГОСТом на данный штамп и не обязательно являются последовательными числами. После выбора текущей ячейки в нее можно поместить текст вызовом функции `TextLine()`.

Ниже приведен пример (листинг 3.8), в котором выполняется открытие штампа, перебор и заполнение ячеек с номерами от 0 до 200, а затем закрытие штампа.

Листинг 3.8 – Открытие и заполнение штампа

```
// Пример заполнения граф штампа
void EditStamp()
{
	if (OpenStamp()) // Открыть штамп чертежа/текстового документа
	{
		// Цикл перебора ячеек штампа
```

```
            for (int i = 0; i < 200; i++)
            {
                  // Заменим i-ю графу
                  ColumnNumber(i);
                  TCHAR buf[255];
                  _stprintf_s(buf, _T("%d"), i);
                  TextLineT(NEW_LINE, 0, 0, buf);
            }
            CloseStamp(); // Закрыть штамп чертежа/текстового документа
      }
      else
            ErrorT(_T("Штамп не найден"));
}
```

Графы, заполняемые автоматически, например нумерация листов, игнорируются (рисунок 3.6).

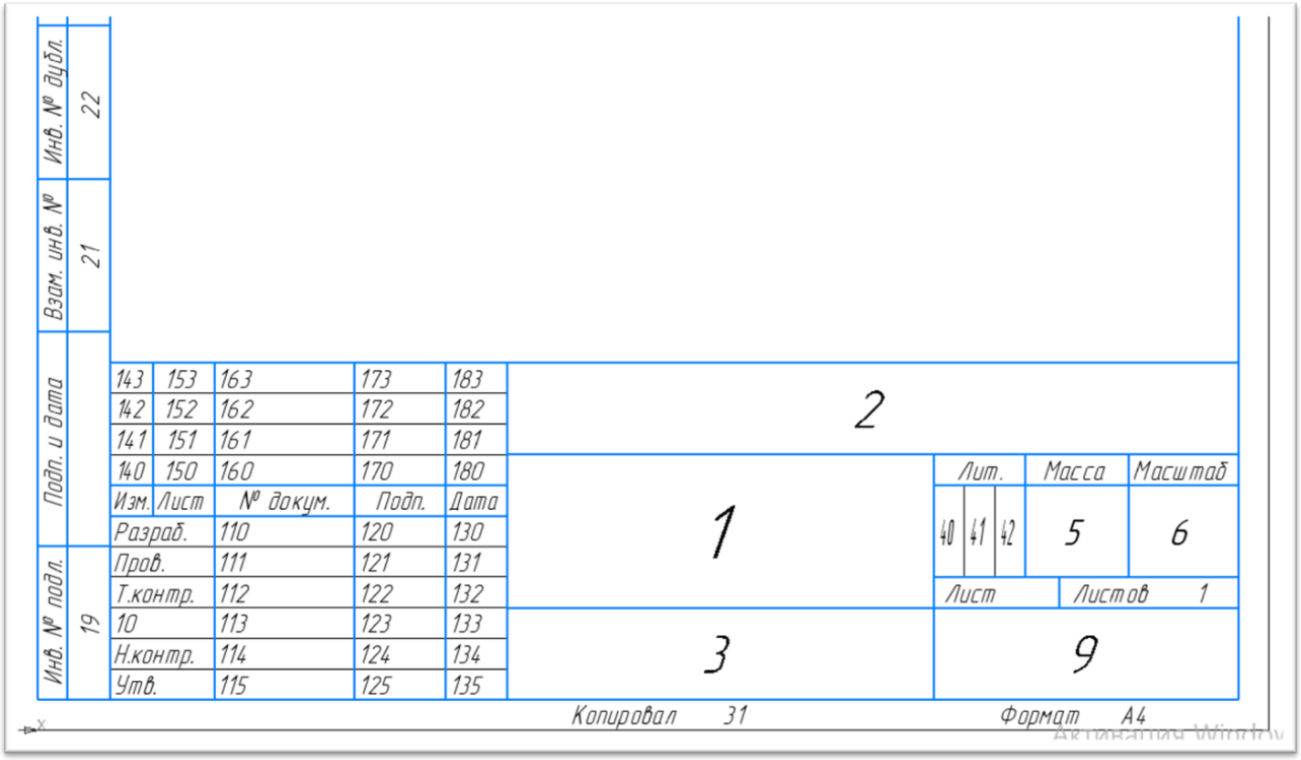



Рисунок 3.6 – Заполненные графы штампа

### 3.4.2 Технические требования

Технические требования – это текст с пояснениями тех особенностей изделия, которые непосредственно из чертежа не видны. Например, это могут быть требования к изготовлению детали - что поверхность надо хромировать, что следует использовать такой-то клей, что все кромки надо скруглить и т.п.

Технические требования (ТТ) в КОМПАС-МАСТЕР являются составным элементом чертежа. Сначала их надо «открыть», затем вызовами `TextLine()` добавить в них текст, а затем – «закрыть». При открытии ТТ указывается динамический массив габаритных прямоугольников, внутри которых будет размещаться текст. Координаты этих прямоугольников задаются в абсолютной системе координат.

В листинге 3.9 показано, как заполняются ТТ, состоящие из 6 текстовых строк и размещенные в двух габаритных прямоугольниках. В примере также показано как можно удалить ТТ, эти строки закомментированы.

Листинг 3.9 – Заполнение и удаление технических требований

```
// Заполнение технических требований
void TDemWork()
{
	// Динамический массив габаритных прямоугольников
	reference pArRect = CreateArray(RECT_ARR, 0);
	// Поместим техтребования в двух габаритных окнах
	RectParam par; // Структура параметров прямоугольника по диагональным
точкам
	// Заполним структуру параметров прямоугольника
	par.pBot.x = 230; // Параметры левой нижней точки прямоугольника
	par.pBot.y = 65;
	par.pTop.x = 415; // Параметры правой верхней точки прямоугольника
	par.pTop.y = 80;
	// Добавим прямоугольник в массив, элемент добавляется в конец массива
	AddArrayItem(pArRect, -1, &par, sizeof(par));
	par.pBot.x = 45;  // Параметры левой нижней точки прямоугольника
	par.pBot.y = 15;
```

```
	par.pTop.x = 230; // Параметры правой верхней точки прямоугольника
	par.pTop.y = 60;
	// Добавим прямоугольник в массив, элемент добавляется в конец массива
	AddArrayItem(pArRect, -1, &par, sizeof(par));
	// Открыть технические требования
	OpenTechnicalDemand(pArRect, 0); // Динамический массив габаритных прямоугольников, если 0, то технические требования размещаются на одной странице автоматически
	  // Задание подстроки технических требований
	for (int i = 1; i < 7; i++)
	{
		TCHAR buf[255];
		_stprintf_s(buf, _T("Строка номер - %d"), i);
		TextLineT(NEW_LINE,    // Значения свойств, устанавливаемые двоичными флагами
			0,           // Тип свойства
			0,           // Указатель на значение свойства
			buf);// Строка
	}
	// Закрыть технические требования
	reference rTechnicalDemand = CloseTechnicalDemand();
	//MessageT(_T("Удалим технические требования"));
	// Удалим технические требования
	//DeleteObj(rTechnicalDemand);
}
```

В зависимости от размера прямоугольников, в которых располагается текст, строки по заполнении первой области начинают заполнять следующую (рисунок 3.7).

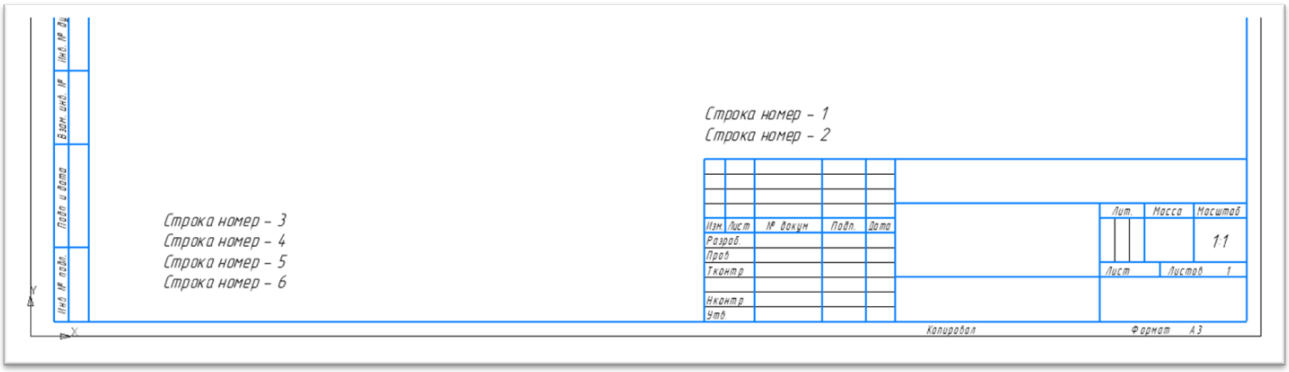



Рисунок 3.7 – Заполненные технические требования

### 3.4.3 Знак неуказанной шероховатости

Если к какой-либо поверхности есть особые требования, например, что она должна быть полированной с таким-то качеством, то на чертеже около этой поверхности может быть поставлен специальный размерный знак. Для большинства поверхностей, качество которых не принципиально важно, на чертеже никаких обозначений не ставится. Но общие требования к качеству этих поверхностей могут быть выражены в числовом виде в виде знака неуказанной шероховатости, который располагается в правом верхнем углу чертежа. В общем, это некое число, которое по ГОСТу обозначает величину допустимых «впадин» и «выпуклостей» на поверхности.

В КОМПАС-МАСТЕР для задания знака неуказанной шероховатости надо заполнить свойствами функцию SpecRoughParam() и затем создать знак на чертеже вызовом ksSpecRough(). В листинге 3.10 приведен пример задания шероховатости.

Листинг 3.10 – Оформление шероховатости

```
// Шероховатость
void DrawSpecRough()
{
	SpecRoughParamT param;
	param.style = 0;	// стиль текста
	param.sign = 2;	// тип знака 0-вид обработка не устанавливается,
	//1 - обработка удалением слоя материала,
```

```
        //2 - обработка без удаления слоя материала
        param.t = 1;          // 1- есть знак в скобках 0 -нет
        _tcscpy_s(param.s, _T("Rz40"));
        // Задать шероховатость неуказанных поверхностей
        ksSpecRoughT(&param);
}
```

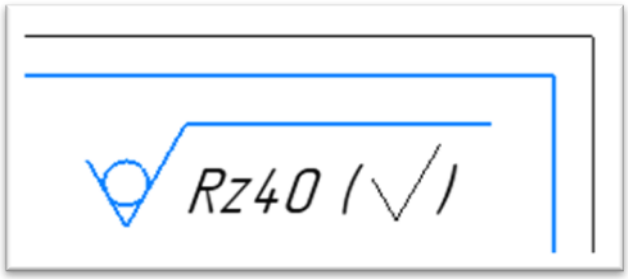



Рисунок 3.8 – Оформление шероховатости

### 3.5 Задание для лабораторной работы

1 Изучить теоретический материал.

2 Разработать прикладную библиотеку КОМПАС, которая выполняет операции с графическим документом по заданному варианту, приведенному в приложении В.

3 Подготовить ответы на контрольные вопросы.

### 3.6 Содержание отчета

В отчете по лабораторной работе согласно СТО 02069024.101–2015 РАБОТЫ СТУДЕНЧЕСКИЕ [3] должны содержаться следующие пункты:

– название лабораторной работы;

– цель работы;

– задание на лабораторную работу;

– код разработанной программы с комментариями;

– экранные формы с отображением результата выполнения программы в системе КОМПАС;

– выводы;

– список использованных источников.

### 3.7 Контрольные вопросы

1 Назовите основные три компонента чертежа.

2 Опишите структуру графического документа.

3 Какие операции с файлами документов вам известны?

4 Какие операции с видами вам известны?

5 Какие операции со слоями вам известны?

6 Какие операции с группами вам известны?

7 Как создать итератор?

8 Как переместить итератор на первый объект?

9 Как создать итератор, перемещающийся только по выделенным объектам чертежа?

10 Как переместить итератор на следующий объект?

11 Как удалить итератор?

12 Как создать итератор, перемещающийся только по группе объектов чертежа?

13 Как определить тип объекта?

14 Как определить параметры объекта?

15 Как установить параметры объекта?

16 Каким образом осуществляется работа со штампом документа?

17 Каким образом осуществляется работа с техническими требованиями?

18 Каким образом осуществляется работа с неуказанной шероховатостью?

# 4 Динамические массивы и изменение параметров элементов чертежа

## 4.1 Динамические массивы

Динамическим массивом обычно называется одномерный массив, размер которого жестко не задается при создании массива и может изменяться в процессе работы по мере добавления или удаления элементов массива.

Динамические массивы в КОМПАС-МАСТЕР применяются для получения или приема данных от функций интерфейсов, которые могут возвращать различное количество элементов данных. Например, у ломаной линии может быть произвольное количество вершин, а у текстового параграфа – произвольное количество строк. Каждая вершина ломаной или каждая строка параграфа описывается специальной структурой данных и для получения всех данных подобных составных элементов удобно применять динамические массивы.

При получении массивов от функций КОМПАС-МАСТЕР эти массивы создаются самими функциями, возвращающими запрошенные данные.

Новый динамический массив можно создать вызовом функции `CreateArray`. В качестве параметра в функцию нужно передать код типа элементов, которые будут храниться в массиве (например, константа `POINT_ARR` для массива точек, `RECT_ARR` для массива прямоугольников и др.).

Допустимые типы массивов и имена соответствующих структур перечислены в таблице 4.1.

КОМПАС-МАСТЕР позволяет создавать массивы для хранения элементов произвольного типа, определяемого программистом RTW-библиотеки (тип массива `USER_ARR`). В случаях, когда не предполагается передавать эти данные функциям КОМПАС-МАСТЕР, особого смысла пользоваться динамическими массивами нет - быстродействие COM-автоматизации слишком мало, и контейнерные классы, обычно имеющиеся в библиотеках конкретной среды программирования, будут гораздо эффективнее.

Таблица 4.1 – Предопределенные типы динамических массивов и структуры элементов

| № | Константа типа массива | Структура | Объекты, параметры которых хранятся в элементах массива |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| Графические примитивы | | | |
| 1 | POINT_ARR | MathPointParam | Точка |
| 2 | POLYLINE_ARR | - | Динамический массив полилиний (указателей массивов POINT_ARR), данный динамический массив является вложенным |
| 3 | RECT_ARR | RectParam | Прямоугольник |
| 4 | NURBS_POINT_ARR | NurbsPointParam | Динамический массив структур точек кривой NURBS |
| 5 | CURVE_PATTERN_ARR | CurvePattern | Участок штриховой кривой |
| 6 | CURVE_PATTERN_ARR_EX | CurvePatternEx | Участок штриховой кривой («расширенные» параметры) |
| 7 | CORNER_ARR | CornerParam | Скругленный угол прямоугольника или правильного многоугольника |
| 8 | HATCHLINE_ARR | HatchLineParam | Линия штриховки |
| 9 | CHAR_STR_ARR | - | Символьная строка (до 255 символов) |

Продолжение таблицы 4.1

| 1 | 2 | 3 | 4 |
|---|---|---|---|
| 10 | TEXT_LINE_ARR | TextLineParam | Текстовая строка |
| 11 | TEXT_ITEM_ARR | TextItemParam | Компонента текстовой строки |
| Атрибуты элементов чертежа и компоненты спецификации | | | |
| 12 | ATTR_COLUMN_ARR | ColumnInfo | Колонка табличного атрибута |
| 13 | LIBRARY_ATTR_TYPE_ARR | LibraryAttrTypeParam | Тип атрибута |
| 14 | LIBRARY_ STYLE_ARR | LibraryStyleParam | Стиль из библиотеки стилей |
| 15 | DOC_SPCOBJ_ARR | DocAttachedSpcParam | Документ, подключенный к объекту спецификации |
| 16 | SPCSUBSECTION_ARR | SpcSubSectionParam | Подраздел спецификации |
| 17 | SPCTUNINGSEC_ARR | SpcTuningSectionParam | Настройка раздела спецификации |
| 18 | SPCSTYLECOLUMN_ARR | SpcStyleColumnParam | Стиль колонки спецификации |
| 19 | SPCSTYLESEC_ARR | SpcStyleSectionParam | Стиль раздела спецификации |
| Размеры | | | |
| 20 | QUALITYITEM_ARR | QualityItemParam | Интервал квалитета |
| 21 | TOLERANCEBRANCH_ARR | ToleranceBranch | «Опора» допуска формы |

Продолжение таблицы 4.1

| 1 | 2 | 3 | 4 |
|---|---|---|---|
| Разное | | | |
| 22 | CONSTRAINT_ARR | ConstraintParam | Параметрическая связь или ограничение |
| 23 | VARIABLE_ARR | VariableParam | Внешняя параметрическая переменная модели графического документа |
| 24 | DOUBLE_ARR | - | Вещественное число типа |
| 25 | LTVARIANT_ARR | LtVariant | Значение variant-совместимого типа |
| 26 | USER_ARR | - | Пользовательский тип данных |

Динамический массив в КОМПАС-МАСТЕР представлен функциями, приведенными в таблице 4.2.

Таблица 4.2 – Функции работы с динамическими массивами

| № | Имя | Назначение |
|---|---|---|
| 1 | 2 | 3 |
| 1 | AddArrayItem | Добавление элемента в массив |
| 2 | ClearArray | Очистка содержимого массива |
| 3 | DeleteArray | Удаление массива и всех его элементов |
| 4 | ExcludeArrayItem | Удаление элемента из массива |
| 5 | GetArrayCount | Получение количества элементов массива |
| 6 | GetArrayItem | Получение элемента массива |
| 7 | GetArrayType | Получение типа элементов массива |
| 8 | GetUserArrayItem | Получить указатель на элемент пользовательского динамического массива |
| 9 | SetArrayItem | Задание значения элемента массива |

Элементы массива индексируются целыми числами, начиная с 0. Последний элемент имеет специальное обозначение -1. Это удобно, например, для удаления последнего элемента – не требуется вызывать функцию для вычисления количества элементов массива.

Добавление/удаление элементов в конец массива приводит к росту/уменьшению массива, но все элементы внутри него остаются действительными и корректными. Допустим, в массиве есть 5 элементов, с индексами от 0 до 4. Если попытаться вставить новый элемент перед 10-м, несуществующим, то вызов `AddArrayItem` вернет 0 в качестве признака неудачи (в случае успеха возвращается 1). Значение 10 в данном случае - неверное значение индекса массива. Добавить элемент в массив можно либо в конец массива, либо перед существующим элементом. Такое поведение динамического массива напоминает связный список с индексацией узлов.

### 4.1.1 Пример использования динамического массива

Рассмотрим способы вызова функций динамического массива на примере, в котором выполняются следующие операции:

– создается пустой массив для хранения строк;

– в массив добавляются три элемента-строки;

– элементы из массива по очереди выводятся в окне сообщения;

– один элемент массива удаляется, а значение одного элемента изменяется;

– содержимое массива снова выводится в диалоговое окно;

– массив и все его элементы удаляются.

В конце процедуры `StrIndefiniteArray()` выполняется удаление динамического массива. Удаление массива надо выполнять в случае, если массив был создан программистом. Массивы, полученные от функций КОМПАС-МАСТЕР, удалять не обязательно. Система КОМПАС управляет распределением памяти для динамических массивов и освобождает память автоматически.

В листинге 4.1 реализованы данные действия.

Листинг 4.1 – Пример работы с динамическими массивами

```
// Массив предназначен для хранения строк
void StrIndefiniteArray()
{ // Создать массив для хранения строк
      reference rArString = CreateArray(CHAR_STR_ARR_T, 0);
      // Наполним массив строк
      TCHAR buf[255];
      _tcscpy_s(buf, _T("12345"));
      // Добавим 1-ю строку, элемент добавляется в конец массива
      AddArrayItem(rArString,       // Указатель на массив
            -1,              // Индекс в массиве
            &buf,            // Указатель на структуру элемента
            sizeof(buf)); // Размер структуры элемента
      _tcscpy_s(buf, _T("67890"));
      // Добавим 2-ю строку, элемент добавляется в конец массива
      AddArrayItem(rArString, -1, &buf, sizeof(buf));
      _tcscpy_s(buf, _T("qwerty"));
      // Добавим 3-ю строку, элемент добавляется в конец массива
      AddArrayItem(rArString, -1, &buf, sizeof(buf));
      // Количество элементов в массиве строк
      int count = GetArrayCount(rArString);
      _stprintf_s(buf, _T("count = %d"), count);
      MessageT(buf);
      // Просмотрим массив строк
      for (int i = 0; i < count; i++)
      {// Получить значение элемента массива
            GetArrayItem(rArString,       // Указатель на массив
                  i,               // Индекс в массиве
                  buf,             // Указатель на структуру элемента
                  sizeof(buf)); // Размер структуры элемента
            MessageT(buf);
      }
      // Исключить из массива элемент с индексом 1
      ExcludeArrayItem(rArString, 1);
      // Количество элементов в массиве строк
      count = GetArrayCount(rArString);
      //Изменение элемента массива с индексом 1
```

```
	_tcscpy_s(buf, _T("фыва"));
	SetArrayItem(rArString, 1, buf, sizeof(buf));
	// Просмотрим массив строк
	for (int i = 0; i < count; i++)
	{// Получить значение элемента массива
		GetArrayItem(rArString, i, buf, sizeof(buf));
		MessageT(buf);
	}
	DeleteArray(rArString); // Удалить динамический массив строк
}
```

### 4.1.2 Вложенные динамические массивы

Динамические массивы в КОМПАС-МАСТЕР одномерные. В ряде случаев для представления данных требуются массивы большей размерности. Предположим, что в RTW-библиотеке требуется работать с массивом ломаных линий. Каждая ломаная описывается массивом точек. В подобной ситуации можно применить вложенные динамические массивы – т.е. массив элементов типа «Ломаная», каждый из которых является массивом типа «Точка».

В приведенной ниже функции `PolyLineArray()`, реализованной в листинге 4.1 выполняются следующие действия:

– создается массив из трех ломаных;

– массив ломаных отображается на чертеже;

– удаляются два массива - массив ломаных и вспомогательный массив точек, используемый для задания каждой ломаной.

Листинг 4.2 – Работа с вложенными динамическими массивами

```
// Массив полилиний это массив массивов математических точек
void PolyLineArray()
{
MathPointParam parMathPoint; // Структура параметров математической точки
	reference rArMathPoint = CreateArray(POINT_ARR, 0);	// Массив для
хранения математических точек
```

```
    reference rArPolyLine = CreateArray(POLYLINE_ARR, 0); // Массив полили-
ний - массивы математических точек
    // Наполнить массив математических точек
    parMathPoint.x = 10;
    parMathPoint.y = 10;
    // Добавим 1-ю точку, элемент добавляется в конец массива
    AddArrayItem(rArMathPoint,              // Указатель на массив
        -1,                         // Индекс в массиве
        &parMathPoint,              // Указатель на структуру элемента
        sizeof(parMathPoint)); // Размер структуры элемента
    parMathPoint.x = 100;
    parMathPoint.y = 100;
    // Добавим 2-ю точку, элемент добавляется в конец массива
    AddArrayItem(rArMathPoint, -1, &parMathPoint, sizeof(parMathPoint));
    parMathPoint.x = 1000;
    parMathPoint.y = 1000;
    // Добавим 3-ю точку, элемент добавляется в конец массива
    AddArrayItem(rArMathPoint, -1, &parMathPoint, sizeof(parMathPoint));
    // Добавим 1-й массив математических точек в массив полилиний, элемент
добавляется в конец массива
    AddArrayItem(rArPolyLine, -1, &rArMathPoint, sizeof(rArMathPoint));
    // Очистили массив математических точек, чтобы использовать для созда-
ния 2-й полилинии
    ClearArray(rArMathPoint);
    // Наполнить массив математических точек
    parMathPoint.x = 20;
    parMathPoint.y = 20;
    // Добавим 1-ю точку, элемент добавляется в конец массива
    AddArrayItem(rArMathPoint, -1, &parMathPoint, sizeof(parMathPoint));
    parMathPoint.x = 200;
    parMathPoint.y = 200;
    // Добавим 2-ю точку, элемент добавляется в конец массива
    AddArrayItem(rArMathPoint, -1, &parMathPoint, sizeof(parMathPoint));
    parMathPoint.x = 2000;
```

```
	parMathPoint.y = 2000;
	// Добавим 3-ю точку, элемент добавляется в конец массива
	AddArrayItem(rArMathPoint, -1, &parMathPoint, sizeof(parMathPoint));
	// Добавим 2-й массив математических точек в массив полилиний, элемент добавляется в конец массива
	AddArrayItem(rArPolyLine, -1, &rArMathPoint, sizeof(rArMathPoint));
	// Очистили массив математических точек, чтобы использовать для создания 3-й полилинии
	ClearArray(rArMathPoint);
	// Наполнить массив математических точек
	parMathPoint.x = 30;
	parMathPoint.y = 30;
	// Добавим 1-ю точку, элемент добавляется в конец массива
	AddArrayItem(rArMathPoint, -1, &parMathPoint, sizeof(parMathPoint));
	parMathPoint.x = 300;
	parMathPoint.y = 300;
	// Добавим 2-ю точку, элемент добавляется в конец массива
	AddArrayItem(rArMathPoint, -1, &parMathPoint, sizeof(parMathPoint));
	parMathPoint.x = 3000;
	parMathPoint.y = 3000;
	// Добавим 3-ю точку, элемент добавляется в конец массива
	AddArrayItem(rArMathPoint, -1, &parMathPoint, sizeof(parMathPoint));
	// Добавим 3-й массив математических точек в массив полилиний, элемент добавляется в конец массива
	AddArrayItem(rArPolyLine, -1, &rArMathPoint, sizeof(rArMathPoint));
	// Количество элементов в массиве полилиний
	int count = GetArrayCount(rArPolyLine);
	TCHAR buf[255];
	_stprintf_s(buf, _T("count = %d"), count);
	MessageT(buf);
	// Просмотрим массив полилиний
	// Цикл по полилиниям - массивы математических точек
	for (int i = 0; i < count; i++)
	{ // Получить значение элемента массива
```

```
            GetArrayItem(rArPolyLine,              // Указатель на массив
                  i,                         // Индекс в массиве
                  &rArMathPoint,             // Указатель на структуру элемента
                  sizeof(rArMathPoint)); // Размер структуры элемента
                  // Количество элементов в массиве математических точек
            int count1 = GetArrayCount(rArMathPoint);
            // Цикл по полилинии - массиву математических точек
            for (int j = 0; j < count1; j++)
            { // Получить значение элемента массива
                  GetArrayItem(rArMathPoint,             // Указатель на массив
                        j,                         // Индекс в массиве
                        &parMathPoint,    // Указатель на структуру элемента
                        sizeof(parMathPoint)); // Размер структуры элемента
                  _stprintf_s(buf, _T("i = %d, x = %4.2f, y = %4.2f"), i, par-
MathPoint.x, parMathPoint.y);
                  MessageT(buf);
            }
      }
      // Просмотрим массив полилиний
      // Цикл по полилиниям - массивы математических точек
      for (int i = 0; i < count; i++)
      { // Получить значение элемента массива
            GetArrayItem(rArPolyLine,              // Указатель на массив
                  i,                         // Индекс в массиве
                  &rArMathPoint,             // Указатель на структуру элемента
                  sizeof(rArMathPoint)); // Размер структуры элемента
                  // Количество элементов в массиве математических точек
            int count1 = GetArrayCount(rArMathPoint);
            // Цикл по полилинии - массиву математических точек
            for (int j = 0; j < count1; j++)
            { // Получить значение элемента массива
                  GetArrayItem(rArMathPoint,             // Указатель на массив
                        j,                         // Индекс в массиве
```

```
                    &parMathPoint,            // Указатель на структуру
элемента
                    sizeof(parMathPoint)); // Размер структуры элемента
                _stprintf_s(buf, _T("i = %d, x = %4.2f, y = %4.2f"), i, par-
MathPoint.x, parMathPoint.y);
                MessageT(buf);
            }
        }
        // Просмотрим массив полилиний
        // Цикл по полилиниям - массивы математических точек
        for (int i = 0; i < count; i++)
        { // Получить значение элемента массива
            GetArrayItem(rArPolyLine, i, &rArMathPoint, sizeof(rArMathPoint));
            // Количество элементов в массиве математических точек
            int count1 = GetArrayCount(rArMathPoint);
            // Цикл по полилинии - массиву математических точек
            for (int j = 0; j < count1; j++)
            {
                // Получить значение элемента массива
                GetArrayItem(rArMathPoint, j, &parMathPoint,
sizeof(parMathPoint));
                _stprintf_s(buf, _T("i = %d, x = %4.2f, y = %4.2f"), i, par-
MathPoint.x, parMathPoint.y);
                MessageT(buf);
            }
        }
        // Удалить динамические массивы
        DeleteArray(rArMathPoint); // Массив для хранения математических точек
        DeleteArray(rArPolyLine);  // Массив полилиний - массивы математических
точек
}
```

## 4.2 Изменение параметров элементов чертежа

### 4.2.1 Использование структур параметров

При вызове функций рисования геометрических элементов этим функциям передаются параметры элементов - либо явным перечислением значений, либо в виде структур данных (интерфейсов). Впоследствии параметры элемента чертежа может потребоваться изменить, не удаляя перерисовывая элемент заново.

Для этого в есть две универсальных функции `GetObjParam()` и `SetObjParam()`, позволяющих получить или изменить параметры любого элемента по его дескриптору. Обмен значениями параметров выполняется только посредством структур - для каждого элемента чертежа в КОМПАС-МАСТЕР есть специальная структура и соответствующий интерфейс автоматизации.

Допустим, имеется дескриптор окружности и требуется изменить ее радиус и стиль линии. Для этого надо:

– задать структуру `CircleParam`;

– загрузить в созданную структуру параметры окружности;

– изменить значения радиуса и стиля линии;

– записать значения параметров в элемент – окружность.

В листинге 4.3 приведена функция `ChangeCircle()`, в которой выполняются перечисленные действия.

Листинг 4.3 – Использование структур параметров

```
void ChangeCircle()
{
	reference obj;
	CircleParam parCircle;
	char buf[128];
	obj = Circle(0, 0, 50, 1);
	//получаем ее параметры в структуру параметров окружности parCircle
	GetObjParam(obj, &parCircle, sizeof(parCircle), ALLPARAM);
```

```
        sprintf_s(buf, "Тип объекта - Окружность, xc=%4.1f yc=%4.1f rad=%4.1f
Стиль=%d", parCircle.xc, parCircle.yc, parCircle.rad, parCircle.style);
        Message(buf);
        //изменяем параметры окружности
        parCircle.rad = 30; parCircle.style = 7;
        //задаем новые параметры
        if (SetObjParam(obj, &parCircle, sizeof(parCircle), ALLPARAM))
              Message(" Изменили окружность");
        else MessageBoxResult();
}
```

В приведенном примере третьим параметром при вызове функций `GetObjParam()` и `SetObjParam()` является константа `ALLPARAM`, означающая, что при вызове учитываются все параметры элемента.

Для некоторых, наиболее крупных, структур параметров, в КОМПАС-МАСТЕР определен набор констант для частичного изменения/получения параметров. Например, для вида или слоя можно получить только код состояния, для размера – отдельно текстовую часть, отдельно параметры привязки или параметры отрисовки. Небольшие по размеру структуры, такие, как параметры отрезка или окружности, можно получить только целиком.

### 4.2.2 Интерактивное редактирование элементов

Существует три функции, с помощью которых RTW-библиотека может предоставить пользователю возможность интерактивного создания или изменения элементов чертежа:

– `ksCreateViewObject` – интерактивное создание элемента заданного типа;

– `ksEditViewObject` – запуск процесса редактирования элемента;

– `ksViewGetObjectArea` – интерактивное формирование группы элементов.

При вызове этих функций управление передается пользователю, и он может в обычном режиме работы с окном КОМПАС-График выполнить соответствующие

действия. В случае неудачи (например, если пользователь отменил команду нажатием `Esc`) эти функции возвращают значение 0.

Кроме трех перечисленных функций, к организации взаимодействия RTW-библиотеки с пользователем имеют отношение функции `Cursor` и `Placement`.

В простейшем случае их можно применить для получения координат точки на чертеже, например, чтобы пользователь указал на какой-нибудь элемент для обработки.

В более сложном варианте использования `Cursor`/ `Placement` позволяют зарегистрировать функцию обратной связи, которая будет вызываться в процессе перемещения указателя пользователем, реализовать отображение варианта детали (фантомного изображения) в процессе перемещения указателя, предъявлять пользователю меню для выбора команды и другие.

Эти функции можно использовать для организации разветвлений в библиотеке. Для этого им передается перечень команд в виде строки или меню. Функция выдает командное окно, где отражается присланное меню и в случае выбора пользователем команды, функция возвращает номер команды.

В приведенной в листинге 4.4 функции `DrawEquidistant()` показано построение эквидистанты – кривой, лежащей на заданном расстоянии от некоторого геометрического объекта. С помощью `CursorEx` выполняется поиск элемента, для которого надо строить эквидистанту (базового элемента).

Функция `CursorEx` возвращает координаты точки, выбранной пользователем, а затем функция `FindObj` позволяет найти элемент чертежа, ближайший к этой точке. Необычно соглашение о коде возврата - признаком того, что функция `CursorEx` успешно возвратила координаты точки, служит значение -1. Значение 0 означает отмену команды, а положительные значения возникают в случае, когда `CursorEx` применяется для работы с меню.

Листинг 4.4 – Построение эквидистанты

```
// Построить эквидистанту
void DrawEquidistant()
{
```

```
      EquidistantParam parEquidistant; // Структура параметров эквидистанты
      parEquidistant.side = 2; // Признак, с какой стороны строить эквиди-
станту ( 0 - слева по направлению,
       // 1 - справа по направлению, 2 - с двух сторон )
      parEquidistant.cutMode = 0; //Тип обхода углов контура (0 - обход сре-
зом, 1 - обход дугой)
      parEquidistant.degState = 0; // Флаг разрешения вырожденных сегментов
эквидистанты ( 0 - вырожденные
       // сегменты запрещены, 1 - вырожденные сегменты разрешены )
      parEquidistant.radRight = 5; // Радиус эквидистанты
      parEquidistant.radLeft = 3; // Радиус эквидистанты
      parEquidistant.style = 1; // Стиль линии ( 1 - основная, 2 - тонкая, 3
- осевая, 4 - штриховая,
       // 5 - волнистая, 6 - утолщенная, 7 - штрихпунктирная с двумя точками,
       // 8 - осевая основная, 9 - штриховая основная, 10 - осевая толстая,
       // 11 - тонкая, включаемая в штриховку )
      // Структура параметров запроса к системе
      RequestInfoT info;
      memset(&info, 0, sizeof(info));
      // Строка или идентификатор меню состава команд
      info.prompt = _T("Укажите объект");
      // Координаты точки ввода
      double x, y;
      // Интерактивный ввод точки или команды
      while (CursorExT(&info, &x, &y, 0, NULL))
      {// Найти ближайший к заданной точке объект вида
            parEquidistant.geoObj = FindObj(x, y,  // Координаты точки
                  1); // Размер стороны квадрата-ловушки с центром в точке x,y
                  // Проверить существование объекта
            if (ExistObj(parEquidistant.geoObj))
            {// Создание эквидистанты
                  reference rEquidistant = Equidistant(&parEquidistant);
                  // Подсветить эквидистанту
                  LightObj(rEquidistant, 1);
```

```
                MessageT(_T("Снять выделение эквидистанты"));
                // Снять выделение эквидистанты
                LightObj(rEquidistant, 0);
            }
            else
                ErrorT(_T("Объект не найден"));
        }
}
```

### 4.3 Отображение текста

Отображение текста выполняется двумя способами:

– отдельными символьными строками;

– параграфами.

Рассмотрим эти способы подробнее.

#### 4.3.1 Вывод отдельных символьных строк

Вывод отдельных строк выполняется функцией `Text`. В качестве параметров указывается точка привязки строки, угол наклона строки, высота символов, сужение, слово флагов и собственно символьная строка. Точкой привязки считается координата левого нижнего угла первого символа строки. В слове флагов можно скомпоновать константы, определяющие начертание текста – например, сделать его наклонным или полужирным.

С выводом текста по смыслу связаны еще две вспомогательных функции – `GetTextLength()` и `GetTextLengthFromReference()`, которые возвращают длину текстовой строки в мм.

Для демонстрации использования `Text` ниже приведена функция (листинг

Листинг 4.5), рисующая в графическом документе таблицу с четырьмя ячейками (2 строки, 2 столбца) и текст внутри ячеек этой таблицы. Текст выводится горизонтально. Отдельно от таблицы приведены примеры различных начертаний текста.

Листинг 4.5 – Демонстрация работы с функцией Text

```
void TableWork()
{
	Table(); // Определение таблицы, возвращает указатель на графический объект таблица  // Разлиновка таблицы
// Рамку таблицы можно формировать из горизонтальных и вертикальных отрезков
	LineSeg(50, 50, 90, 50, 1);
	LineSeg(50, 40, 90, 40, 1);
	LineSeg(50, 30, 90, 30, 1);
	LineSeg(50, 50, 50, 30, 1);
	LineSeg(70, 50, 70, 30, 1);
	LineSeg(90, 50, 90, 30, 1);
	TextT(52, 48, 0, 5, 1, 0, _T("Строка 1"));
	TextT(72, 48, 0, 5, 1, 0, _T("Строка 2"));
	TextT(52, 38, 0, 5, 1, 0, _T("Строка 3"));
	TextT(72, 38, 0, 5, 1, 0, _T("Строка 4"));
	EndObj(); // Описание таблицы заканчивается функцией EndObj
	TextT(50, 15, 0, 10, 1, ITALIC_OFF, _T("Отдельная строка"));
	TextT(30, 90, 0, 5, 1, 0, _T("Простой текст"));
	TextT(30, 80, 0, 5, 1, 0, _T("Пример... дроби $dЧислитель; Знаменатель$"));
	TextT(30, 70, 0, 5, 1, 0, _T("Пример... отклонений 20$0.5; -0.3$"));
	TextT(30, 60, 0, 5, 1, 0, _T("Пример... спецсимвола &32"));
}
```

В данном примере показано, как текст задается внутри таблицы – вызовы Text являются вложенными вызовами для формирования составного элемента «Таблица».

При попытке редактирования этого текста из окна КОМПАС-График оказывается, что область вывода текста ограничена ячейками таблицы.

В общем случае, отдельными вызовами Text создаются самостоятельные элементы «Текстовая строка», область вывода которых не ограничена (например, как в последних вызовах в функции DrawTable()).

Результат работы функции показан на рисунке 4.1.

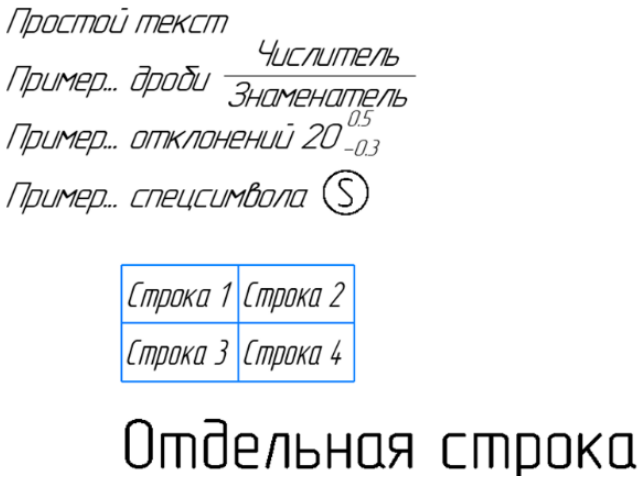



Рисунок 4.1 – Результат работы функции `Text`

### 4.3.2 Текстовые параграфы

Параграф – составной элемент чертежа. При создании параграфа указывается стиль, который можно выбирать из набора системных стилей – стиль по умолчанию, обычный текст, текст размерной надписи, текст технических требований и др. Стиль влияет на параметры шрифта внутри параграфа «по умолчанию».

Параграф состоит из строк, а строки – из компонент. У каждой компоненты есть собственный текст, тип (спецзнак, символ шрифта, дробь, выражение с под- или надстрокой) и параметры начертания шрифта (задаются в виде битовых флагов).

Таким образом, параграф можно представить с помощью вложенных динамических массивов – массива строк, в котором хранятся массивы компонент строк.

В КОМПАС-МАСТЕР параграфы являются основным механизмом представления текстовой информации. Рассмотренная выше функция `Text` в действительности является способом для упрощенного создания однострочных текстовых параграфов.

При рисовании таблицы между вызовами функций `Table()/EndObj()` можно вставлять вызовы для создания произвольных текстовых параграфов, а не только однострочных.

На рисунке 4.2 показан текстовый параграф, состоящий из 4 строк.

Вторая строка состоит из нескольких компонент, не все из которых являются видимыми: компонента с прямым текстом «Дробь и отклонения», компонента с признаком «числитель дроби» и текстом «111 222 333», компонента с признаком «верхний индекс» и текстом «Верх», компонента с признаком «нижний индекс» и текстом «Низ», компонента без текста с признаком «конец индекса», компонента с текстом «444» и признаком «знаменатель дроби», компонента без текста с признаком «конец дроби», компонента с прямым текстом «555».

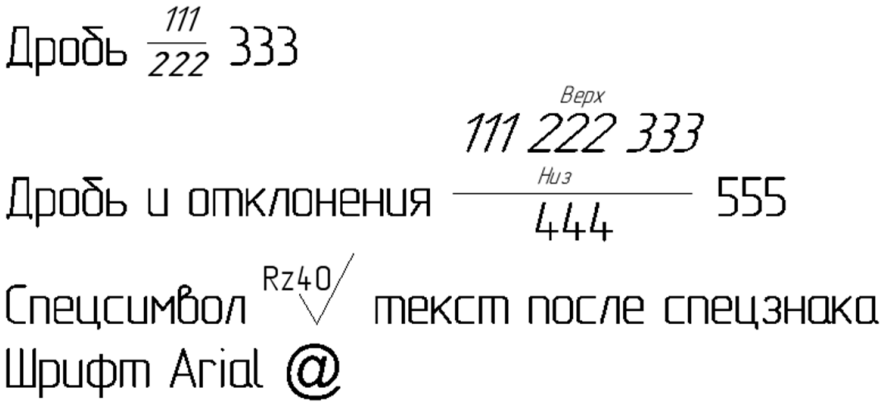



Рисунок 4.2 – Текстовый параграф

В целом создание текстового параграфа выполняется согласно схеме:

– начать создание параграфа заданного стиля;
– создать компоненту строки;
– задать тип компоненты;
– задать текст компоненты;
– задать параметры шрифта компоненты;
– добавить компоненту в параграф;
– повторить действия 2)-6) для всех компонент всех строк параграфа;
– завершить создание параграфа.

В данной схеме разбиение параграфа на строки выполняется неявно. Необязательно в тексте программы собирать отдельные компоненты в строки, а затем до-

бавлять их в параграф. Вместо этого достаточно во флагах начертания первой компоненты каждой строки возводить флаг «Новая строка».

Каждая компонента хранит в себе текст, значение типа компоненты, строку и уточняющий параметр `isNumb` – в зависимости от типа компоненты в нем может быть указан номер спецсимвола, номер символа из произвольного шрифта Windows или тип отрисовки дроби или выражения с под- или надстрокой. Многие параметры, влияющие на вид компоненты, хранятся в слове флагов, определяющих начертание шрифта этой компоненты. В этом слове флагов указываются не только очевидные признаки курсива, полужирного или подчеркнутого начертания, но и признаки «знаменатель дроби», «верхний индекс» и др.

При работе с текстовыми параграфами приходится использовать 4 различных интерфейса: графический документ `Document 2D`, параметры параграфа `ParagraphParam`, параметры компоненты строки `TextItemParam` и шрифт компоненты строки `TextItemFont`.

В приведенной в листинге 4.6 функции `DrawParagraph()`выполняется вывод строк согласно рисунку 4.2. Среди компонент, формирующих вторую строку, встречаются компоненты, не содержащие текста, но имеющие служебные признаки «конец индекса» (`END_DEVIAT`) и конец дроби (`END_FRACTION`) в слове флагов шрифта компоненты.

Листинг 4.6 – Вывод текста

```
// Вывод параметров строки
void PrintTextLine(TextLineParam parTextLine)
{
	TCHAR buf[255];
	_stprintf_s(buf, _T("style = %d"), parTextLine.style);
	MessageT(buf);
TextItemParamT parTextItem; // Структура параметров компоненты строки текста
	// Количество элементов в массиве
	int count = GetArrayCount(parTextLine.pTextItem);
	// Цикл по компонентам строки
	for (int i = 0; i < count; i++)
	{		// Получить значение элемента массива
```

```
            GetArrayItem(parTextLine.pTextItem, i, &parTextItem,
sizeof(parTextItem));
            _stprintf_s(buf, _T("i = %d, font.height = %4.2f, s =
%s,\nfont.fontName = %s"), i, parTextItem.font.height, parTextItem.s, parTex-
tItem.font.fontName);
            MessageT(buf);
      }
}
void DrawParagraph()// Текст
{     ParagraphParam parParagraph; // Структура параметров параграфа
      // Задать параметры параграфа
      parParagraph.style = 0;  // Номер стиля текста (0 - стиль по умолчанию,
/* 1 - обычный текст, 2 - текст для технических требований, 3 - текст размер-
ной надписи, 4 - текст в обозначении шероховатости,5 - текст на позиционной
линии-выноске, 6 - текст над\под полкой линии-выноски, 7 - текст на ответвле-
нии линии-выноски, 8 - текст в обозначении допуска формы, 9 - текст для заго-
ловка таблицы, 10 - текст для ячейки таблицы,11 - текст для линии разреза, 12
- текст для стрелки направления взгляда, 13 - текст в обозначении неуказанной
шероховатости, 14 - текст в обозначении изменения )*/
      parParagraph.x = 30; // Координаты точки привязки текста
      parParagraph.y = 30;
      parParagraph.ang = 0;  // Угол наклона текста
      parParagraph.hFormat = 0;  // Признак горизонтального форматирования (
0 - нет форматирования, 1 - сужение текста, 2 - перенос на другую строку )
      parParagraph.vFormat = 0;  // признак вертикального форматирования ( 0
- нет форматирования,  1-изменение шага строк )
      parParagraph.height = 25; // Высота блока форматирования
      parParagraph.width = 20; // Ширина блока форматирования
      // Параграфом называется автоматически форматируемый блок текста
      Paragraph(&parParagraph);
      // Снятие наклона, по умолчанию включен
      TextLineT(ITALIC_OFF, 0, 0, _T(""));
      // Пример задания дроби
      // Задание подстроки параграфа текста
TextLineT(NEW_LINE, // Значения свойств, устанавливаемые двоичными флагами
```

```
            0,           // Тип свойства
            0,           // Указатель на значение свойства
            _T("Дробь "));  // Строка
// Числитель, наклон, высота дроби в 1.5 раза меньше высоты текста
      int fraction = 2; // Значение свойства FRACTION_TYPE
      TextLineT(NUMERATOR | ITALIC_ON, FRACTION_TYPE, &fraction, _T("111"));
      // Знаменатель, утолщение
      TextLineT(DENOMINATOR | BOLD_ON, 0, 0, _T("222"));
      // Конец дроби, снятие утолщения, снятие наклона и текст после дроби
            TextLineT(END_FRACTION | BOLD_OFF | ITALIC_OFF, 0, 0, _T(" 333"));
      // Пример задания дроби, нижнего и верхнего отклонения
      TextLineT(NEW_LINE, 0, 0, _T("Дробь и отклонения "));
      // Числитель, наклон
      TextLineT(NUMERATOR | ITALIC_ON, 0, 0, _T("111 "));
            // Базвая строка для отклонения
      TextLineT(S_BASE, 0, 0, _T("222"));
      // Верхнее отклонение
      TextLineT(S_UPPER_INDEX, 0, 0, _T("Верх"));
      // Нижнее отклонение
      TextLineT(S_LOWER_INDEX, 0, 0, _T("Низ"));
      // Конец отклонения
      TextLineT(S_END, 0, 0, _T(" 333"));
      // Знаменатель, утолщение
      TextLineT(DENOMINATOR, 0, 0, _T("444"));
      // Конец дроби, снятие утолщения, снятие наклона и текст после дроби
      TextLineT(END_FRACTION | BOLD_OFF | ITALIC_OFF, 0, 0, _T(" 555"));
      // Пример задания спецсимвола
      TextLineT(NEW_LINE, 0, 0, _T("Спецсимвол "));
      int symbol = 65; // Номер спецзнака - шероховатость
      // Спецзнак
      TextLineT(SPECIAL_SYMBOL, SPECIAL, &symbol, _T(""));
      // Конец спецсимвола и текст после спецзнака
      TextLineT(SPECIAL_SYMBOL_END, 0, 0, _T(" текст после спецзнака"));
      // Пример задания символа из существующего шрифта
```

```
	TextLineT(NEW_LINE, 0, 0, _T("Шрифт Arial "));
	TCHAR name[10];
	lstrcpy(name, _T("Arial")); // Название шрифта - Arial
	unsigned int ch = 64; // Номер символа из шрифта - @
	// Задания символа из Arial
	TextLineT(FONT_SYMBOL, FONT_NAME, name, (TCHAR *)(size_t)ch);
	// Описание параграфа заканчивается функцией EndObj, возвращающей ука-
затель на созданный объект
	reference rParagraph = EndObj();
	TextLineParam parTextLine; // Структура параметров строки текста
	// Возьмем параметры 1-ой строки ( индекс 0 )
	GetObjParam(rParagraph, &parTextLine, sizeof(parTextLine), 0);
	// Вывод параметров строки
	PrintTextLine(parTextLine);
	if (YesNoT(_T("Изменять параметры текста ?")) == 1)
	{TextItemParamT parTextItem; // Структура параметров компоненты строки
текста
		// У первой строки включим ITALIC и BOLD и меняем цвет
GetArrayItem(parTextLine.pTextItem, 0, &parTextItem, sizeof(parTextItem));
parTextItem.font.bitVector = parTextItem.font.bitVector | ITALIC_ON |
BOLD_ON;
		// Значения свойств, устанавливаемые двоичными флагами
		parTextItem.font.color = RGB(255, 0, 0); // Цвет
SetArrayItem(parTextLine.pTextItem, 0, &parTextItem, sizeof(parTextItem));
		// Заменим у текста первую строку
		SetObjParam(rParagraph, &parTextLine, sizeof(parTextLine), 0);
		// Возьмем параметры 1-ой строки ( индекс 0 )
		GetObjParam(rParagraph, &parTextLine, sizeof(parTextLine), 0);
		// Вывод параметров строки
		PrintTextLine(parTextLine);
	}
}
```

В последнем условии реализовано изменение параметров текста.

Состояние признаков начертания, которые можно включить/выключить, запоминается и продолжает действовать на все вновь добавляемые компоненты, в том числе и в новых строках. Можно выключить курсив, и все последующие компоненты будут выводиться прямым шрифтом.

При создании параграфа его можно рассматривать состоящим из компонент строк, не выделяя явно отдельные строки. Для изменения содержимого существующего параграфа, напротив, доступ организуется именно к динамическому массиву строк, каждая из которых содержит динамический массив компонент.

### 4.4 Создание размеров

Размеры необходимы для однозначной интерпретации чертежа, и их добавляют столько, сколько необходимо для изготовления детали с требуемыми геометрическими свойствами и качеством.

На рисунке 4.3 приведен пример чертежа с нанесенными размерами. На чертеже есть размеры нескольких типов – линейные, угловые, радиальные, диаметральные. В надписи около размерной линии может присутствовать, помимо значения размера, некоторый текст до значения размера (префикс) и текст после значения размера.

Например, до значения размера может присутствовать символ диаметра или радиуса, а после – допустимые отклонения и квалитет (качество изготовления). Под размерной линией тоже может быть некоторый текст, например, указание количества одинаковых отверстий.

Для нанесения на чертеж размеров и технологических обозначений в КОМПАС-График есть отдельная инструментальная панель. Все средства, доступные в интерактивном режиме пользователю, доступны также и программисту КОМПАС-МАСТЕР. Размеры являются самостоятельными элементами чертежа.

Размеры перемещаются и пересчитываются автоматически только если они связаны с геометрическими элементами параметрическими связями. В макроэлементе размеры только перемещаются (но не пересчитываются) вместе с макроэле-

ментом, а в группе они будут перемещаться (но не пересчитываться), только если выделить и переместить всю группу.

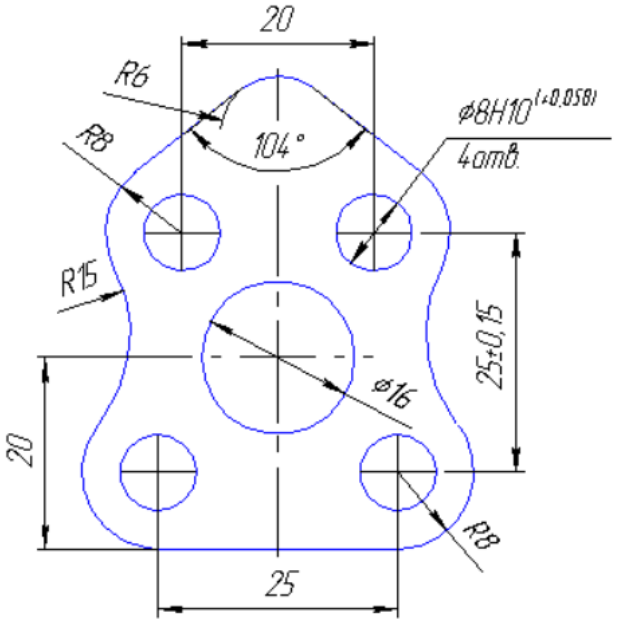



Рисунок 4.3 – Фрагмент чертежа с проставленными размерами

При работе с размерами приходится пользоваться большим количеством различных структур. Во-первых, это объясняется тем, что размер - составной элемент, включает в себя стрелки, полку с текстом, текст надписи. Кроме того, размер привязан к некоторым точкам - характерным точкам геометрического элемента, и эти характеристики привязки у размера тоже надо задавать с помощью специального интерфейса.

Основной структурой для работы с размерами является структура параметров размера (для размера каждого типа есть отдельная структура), например:

– `ABreakDimParam` - угловой размер с обрывом;

– `ADimParam` - угловой размер;

– `LBreakDimParam` - линейный размер с обрывом;

– `LDimParam` - линейный размер;

– `RBreakDimParam` - радиальный размер с изломом;

– `RDimParam` - диаметральный и обычный радиальный размер.

Из большинства перечисленных структур можно получить указатели на подчиненные структуры, предназначенные для задания параметров конкретного размера: параметры размерной надписи, параметры привязки размера и параметры отрисовки размера. В процессе создания размера надо настроить параметры каждого из перечисленных типов.

Определение текста размерной надписи тоже требует взаимодействия с рядом структур. Структура размерной надписи предоставляет доступ к тексту надписи размера, которая хранится в виде динамического массива компонент текстовых строк.

Для примера рассмотрим функцию `DrawLinDim()`, приведенную в листинге 4.7, которая демонстрирует создание линейного размера. После размещения размера, выводится соответствующее сообщение, затем выполняется редактирование размера.

Листинг 4.7 – Создание линейного размера

```
// Линейный размер
void DrawLinDim()
{     LDimParam parLDim; // Структура параметров линейного размера
      memset(&parLDim, 0, sizeof(parLDim));
      // Параметры изображения размера
      parLDim.dPar.textPos = 10;  // Положение текста ( 0 - автоматическое
размещение текста,
      //>0 - на указанное расстояние в направлении от первой точки ко второй,
      //<0 - на указанное расстояние в направлении от второй точки к первой )
      parLDim.dPar.textBase = 2;   // Параметр отрисовки текста ( 0 - в цен-
тре, 1 - textPos относительно 1 точки,
      // 2 - textPos относительно 2 точки, 3 - общая размерная линия )
      parLDim.dPar.pl1 = 0;   // Признак отрисовки первой выносной линии ( 0
- включена, 1 - выключена )
      parLDim.dPar.pl2 = 0;   // Признак отрисовки второй выносной линии ( 0
- включена, 1 - выключена )
      parLDim.dPar.pt1 = 2;   // Тип стрелки у первой выносной линии ( 0 -
стрелки нет, 1 - внутри, 2 - снаружи, 3 - засечка, 4 - точка )
      parLDim.dPar.pt2 = 2;   // Тип стрелки у второй выносной линии ( 0 -
стрелки нет, 1 - внутри, 2 - снаружи, 3 - засечка, 4 - точка )
      parLDim.dPar.shelfDir = 0;   // Наличие выносной полки ( 0 - нет вынос-
ной полки, -1 - полка направлена влево,
```

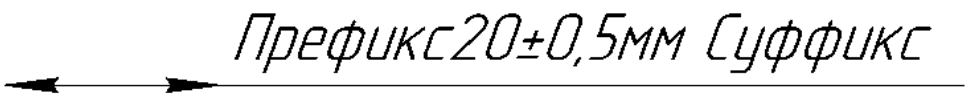
```
	// 1 - полка направлена вправо, 2 - полка направлена вверх, 3 - полка
направлена вниз )
	parLDim.dPar.ang = -30; // Угол наклона ножки выносной полки
	parLDim.dPar.length = 20;  // Длина ножки выносной полки
	// Параметры размерной надписи
	parLDim.tPar.style = 0; // Стиль текста размера, ( 0 - стиль по умолча-
нию )
	parLDim.tPar.sign = 0; // Номер условного значка перед номиналом ( 0 -
нет значка, 1 - диаметр, 2 - квадрат,
	// 3 - радиус, > 3 - номер значка из шрифта Symbol type A )
	//Признаки размерной надписи (набор битовых полей, 0 - ручное задание)
	// Должны быть строки PREFIX, номинал ставится auto (ставить не нужно),
TOLERANCE, отклонения
	// ставятся auto ( ставить не нужно ), UNIT, SUFFIX ( 4 строки )
	parLDim.tPar.bitFlag = _AUTONOMINAL | _PREFIX | _DEVIATION | _UNIT |
_SUFFIX;
	parLDim.tPar.pText = CreateArray(CHAR_STR_ARR_T, 0); // Динамический
массив строк
	// Добавить элементы динамического массива
	AddArrayItem(parLDim.tPar.pText, -1, _T("Префикс"), 8);
	AddArrayItem(parLDim.tPar.pText, -1, _T("+0.5"), 5);
	AddArrayItem(parLDim.tPar.pText, -1, _T("-0.5"), 5);
	AddArrayItem(parLDim.tPar.pText, -1, _T("мм"), 5);
	AddArrayItem(parLDim.tPar.pText, -1, _T(" Суффикс"), 13);
	// Привязка линейного размера
	parLDim.sPar.ps = 0;   // Признак ориентации размерной линии ( 0 - го-
ризонтально, 1 - вертикально,
	// 2 - параллельно отрезку, соединяющему точки привязки, 3 - по dx, dy,
	// 4 - параллельно отрезку с выносными линиями по dx, dy.
	parLDim.sPar.x1 = 50;  //Координаты первой точки привязки
	parLDim.sPar.y1 = 50;
	parLDim.sPar.x2 = 70;  //Координаты второй точки привязки
	parLDim.sPar.y2 = 60;
	parLDim.sPar.dy = -20; //Вектор, определяющий положение размерной линии
	parLDim.sPar.dx = 0;
	parLDim.sPar.basePoint = 1;  // Признак, указывающий, от какой точки
откладывать dx, dy
	// ( 1 - от первой точки, 2 - от второй точки )
	// Создание линейного размера
	reference rLinDimension = LinDimension(&parLDim);
	GetObjParam(rLinDimension,     // Указатель на графический объект
```

```
        &parLDim,          // Указатель на структуру параметров
        sizeof(parLDim), // Размер структуры параметров
        ALLPARAM_T);        // Тип считывания параметров
    TCHAR buf[255];
    _stprintf_s(buf, _T("Создание размера"));
    MessageT(buf);
    // Изменение параметров изображения размера
    parLDim.dPar.textPos = 1; // Положение текста
    parLDim.dPar.textBase = 2; // Параметр отрисовки текста
    parLDim.dPar.pl1 = 1; // Признак отрисовки первой выносной линии
    parLDim.dPar.pl2 = 1; // Признак отрисовки второй выносной линии
    parLDim.dPar.pt1 = 1; // Тип стрелки у первой выносной линии
    parLDim.dPar.pt2 = 1; // Тип стрелки у второй выносной линии
    parLDim.dPar.shelfDir = 0; // Наличие выносной полки
    SetObjParam(rLinDimension,          // Указатель на графический объект
        &parLDim.dPar,          // Указатель на структуру параметров
        sizeof(parLDim.dPar), // Размер структуры параметров
        DIM_DRAW_PARAM);       // Тип считывания параметров
    GetObjParam(rLinDimension,          // Указатель на графический объект
        &parLDim.dPar,          // Указатель на структуру параметров
        sizeof(parLDim.dPar), // Размер структуры параметров
        DIM_DRAW_PARAM);       // Тип считывания параметров
    _stprintf_s(buf, _T("Редактирование размера"));
    MessageT(buf);
    // Удалить динамический массив строк
    DeleteArray(parLDim.tPar.pText);
}
```

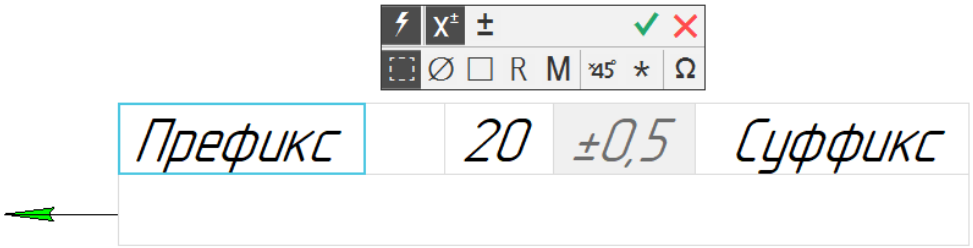
Результат построения до изменения размера и после приведен на рисунке 4.4. На рисунке 4.4, в видно, что текст на размерной линии представляет собой не просто строку, а определенные поля и параметры размерной надписи как объекта КОМПАС.

Создание диаметрального размера рассмотрим на примере функции `DrawDiamDim()`, приведенной в листинге 4.8. Как и в предыдущем примере после размещения размера, выводится соответствующее сообщение, затем выполняется редактирование размера.


а




б




в

а – размер до изменения; б – размер после изменения; в – размер в процессе редактирования

Рисунок 4.4 – Построение линейного размера

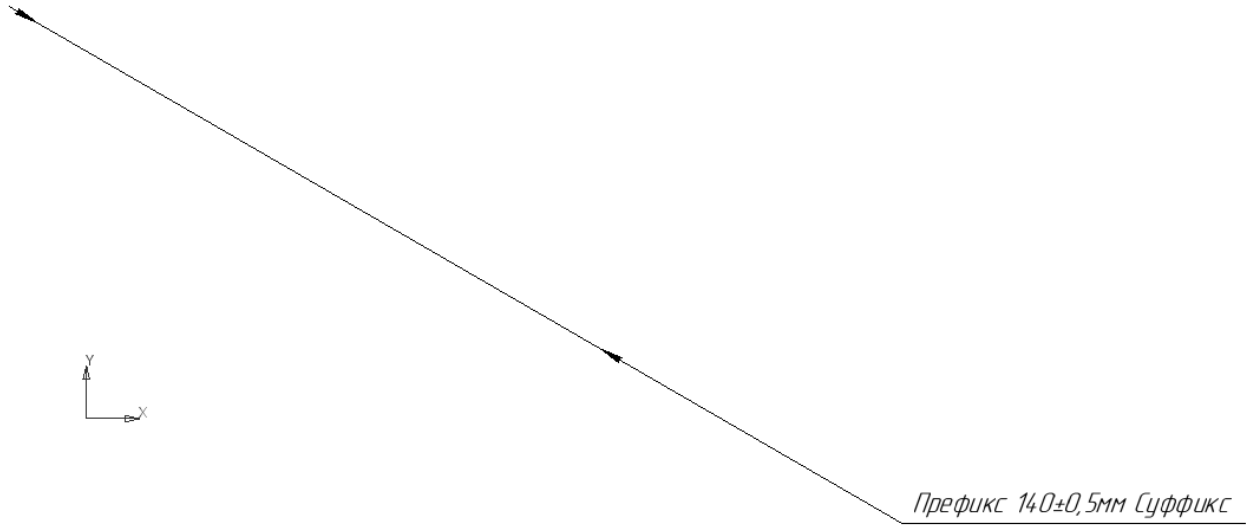
Листинг 4.8 – Создание диаметрального размера

```
void DrawDiamDim()
{
    RDimParam parRDim; // Структура параметров диаметрального и обычного радиального размера
    memset(&parRDim, 0, sizeof(parRDim));
    // Параметры изображения размера
    parRDim.dPar.textPos = 75;  // Параметр отрисовки текста или длина ножки - аннотационный ( не зависит от масштаба )
```

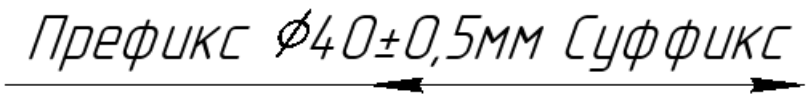
```
    parRDim.dPar.pt1 = 2;   // Тип стрелки у первой выносной линии ( 0 -
    стрелки нет, 1 - внутри, 2 - снаружи, 3 - засечка, 4 - точка )
    parRDim.dPar.pt2 = 2;   // Тип стрелки у второй выносной линии ( 0 -
    стрелки нет, 1 - внутри, 2 - снаружи, 3 - засечка, 4 - точка )
    parRDim.dPar.shelfDir = 1;   // Наличие выносной полки ( 0 - нет выносной
    полки, -1 - полка направлена влево,
    // 1 - полка направлена вправо, 2 - полка направлена вверх, 3 - полка
    направлена вниз )
    parRDim.dPar.ang = -30; // Угол наклона ножки выносной полки
     // Параметры размерной надписи
     parRDim.tPar.style = 0; //Стиль текста размера, (0 стиль по умолчанию)
     parRDim.tPar.sign = 0; // Номер условного значка перед номиналом ( 0 -
нет значка, 1 - диаметр, 2 - квадрат,
     // 3 - радиус, > 3 - номер значка из шрифта Symbol type A )
     // Признаки размерной надписи (набор битовых полей, 0 - ручное задание)
     // Должны быть строки PREFIX, номинал ставится auto (ставить не нужно),
TOLERANCE, отклонения
     // ставятся auto ( ставить не нужно ), UNIT, SUFFIX ( 4 строки )
     parRDim.tPar.bitFlag = _AUTONOMINAL | _PREFIX | _DEVIATION | _UNIT |
_SUFFIX;
     parRDim.tPar.pText = CreateArray(CHAR_STR_ARR_T, 0); // Динамический
массив строк
     // Добавить элементы динамического массива
     AddArrayItem(parRDim.tPar.pText, -1, _T("Префикс "), 8);
     AddArrayItem(parRDim.tPar.pText, -1, _T("+0.5"), 5);
     AddArrayItem(parRDim.tPar.pText, -1, _T("-0.5"), 5);
     AddArrayItem(parRDim.tPar.pText, -1, _T("мм"), 5);
     AddArrayItem(parRDim.tPar.pText, -1, _T(" Суффикс"), 13);
     // Привязка диаметрального размера
     parRDim.sPar.xc = 50; // Координаты центра
     parRDim.sPar.yc = 50;
     parRDim.sPar.rad = 70; // Радиус
     // Создание диаметрального размера
     reference rDiamDimension = DiamDimension(&parRDim);
     GetObjParam(rDiamDimension,    // Указатель на графический объект
          &parRDim,          // Указатель на структуру параметров
          sizeof(parRDim), // Размер структуры параметров
          ALLPARAM_T);        // Тип считывания параметров
     TCHAR buf[255];
     _stprintf_s(buf, _T("Создание размера"));
     MessageT(buf);
```

```
	// Параметры изображения размера
	parRDim.dPar.textPos = 0; // Параметр отрисовки текста или длина ножки - аннотационный ( не зависит от масштаба )
	parRDim.dPar.pt1 = 1; // Тип стрелки у первой выносной линии ( 0 - стрелки нет, 1 - внутри, 2 - снаружи, 3 - засечка, 4 - точка )
	parRDim.dPar.pt2 = 1; // Тип стрелки у второй выносной линии ( 0 - стрелки нет, 1 - внутри, 2 - снаружи, 3 - засечка, 4 - точка )
	parRDim.dPar.shelfDir = 0; // Наличие выносной полки ( 0 - нет выносной полки, -1 - полка направлена влево,
	// 1 - полка направлена вправо, 2 - полка направлена вверх, 3 - полка направлена вниз )
	parRDim.dPar.ang = 0; // Угол наклона ножки выносной полки
	// Параметры размерной надписи
	parRDim.tPar.sign = 1; // Номер условного значка перед номиналом ( 0 - нет значка, 1 - диаметр, 2 - квадрат,
	// 3 - радиус, > 3 - номер значка из шрифта Symbol type A )
	// Привязка диаметрального размера
	parRDim.sPar.xc = 30; // Координаты центра
	parRDim.sPar.yc = 30;
	parRDim.sPar.rad = 20; // Радиус
	SetObjParam(rDiamDimension,    // Указатель на графический объект
		&parRDim,          // Указатель на структуру параметров
		sizeof(parRDim), // Размер структуры параметров
		ALLPARAM_T);        // Тип считывания параметров
	GetObjParam(rDiamDimension,    // Указатель на графический объект
		&parRDim,          // Указатель на структуру параметров
		sizeof(parRDim), // Размер структуры параметров
		ALLPARAM_T);        // Тип считывания параметров
	_stprintf_s(buf, _T("Редактирование размера"));
	MessageT(buf);
	// Удалить динамический массив строк
	DeleteArray(parRDim.tPar.pText);
}
```

Результат построения диаметрального размера до изменения и после приведен на рисунке 4.5.




а




б

а – размер до изменения; б – размер после изменения

Рисунок 4.5 – Построение диаметрального размера

### 4.5 Задание для лабораторной работы

1 Изучить теоретический материал.

2 Разработать прикладную библиотеку КОМПАС, которая выполняет построение размеров и вывод текстовых строк в соответствии с заданным вариантом, приведенным в приложении Б.

3 Подготовить ответы на контрольные вопросы.

### 4.6 Содержание отчета

В отчете по лабораторной работе согласно СТО 02069024.101–2015 РАБОТЫ СТУДЕНЧЕСКИЕ [3] должны содержаться следующие пункты:

– название лабораторной работы;

– цель работы;

– задание на лабораторную работу;

– код разработанной программы с комментариями;

– экранные формы с отображением результата выполнения программы в системе КОМПАС;

– выводы;

– список использованных источников.

### 4.7 Контрольные вопросы

1 Что понимается под динамическими массивами?

2 Назовите предопределенные типы динамических массивов.

3 Назовите функции работы с динамическими массивами.

4 Для чего используются вложенные динамические массивы?

5 Объясните параметры функции CursorEx.

6 Объясните параметры функции Placement.

7 Опишите структуру RequestInfo.

8 Опишите функцию FindObj.

9 Объясните параметры функции Text.

10 Объясните структуру параметров параграфа ParagraphParam.

11 Как вывести в тексте дробь?

12 Как вывести в текст отклонения?

13 Как вывести в текст спецсимвол?

14 Как вывести в текст спецзнак?

15 Как изменить цвет текста?

16 Опишите структуру  углового размера с обрывом ABreakDimParam.

17 Опишите структуру  углового размера ADimParam.

18 Опишите структуру  линейного размера с обрывом LBreakDimParam.

19 Опишите структуру линейного размера LDimParam.

20 Опишите структуру радиального размера с изломом RBreakDimParam.

# 5 Интерактивное взаимодействие пользователя с КОМПАС

## 5.1 Введение в интерактивное взаимодействие пользователя с КОМПАС

Многие команды RTW-библиотек не удается сделать полностью автоматическими и при выполнении этих команд приходится запрашивать данные у пользователя.

Операции ввода данных можно разделить на три группы:

– операции со стандартными диалоговыми окнами;

– операции с окном КОМПАС;

– операции с библиотечными диалоговыми окнами.

Примерами стандартных диалоговых окон являются окна сообщений, окна сохранения и открытия файлов. Для вызова этих окон в КОМПАС-МАСТЕР есть специальные функции.

Среди операций с окном КОМПАС выделяются операции указания точки и направления (угла) на листе чертежа. Пользователя можно попросить указать мышью на некоторую точку или задать значение угла. Эти данные удобно использовать для создания некоторой библиотечной детали в заданном месте чертежа и с заданной ориентацией. С операциями задания местоположения тесно связан вопрос вариантного (фантомного) отображения. Когда пользователь размещает деталь на чертеже, в процессе перемещения мыши желательно также перемещать на чертеже некоторое предварительное изображение детали – вариантное изображение, которое пользователь может принять или отказаться от него.

RTW-библиотеки могут выводить собственные диалоговые окна. При построении стандартных деталей может потребоваться указать параметры этой детали, если, например, ее можно строить в различных видах или с различными параметрами по ГОСТу. Подобные параметры удобно вводить в специальных диалоговых окнах, а затем учитывать при построении детали.

В диалоговом окне (рисунок 5.1) присутствуют стандартные элементы управления Windows, а также слайд с изображением детали в соответствии с текущими параметрами. На слайде пользователь сразу видит примерное изображение детали.

Для отображения слайдов детали можно пользоваться BMP-слайдами, но более удобным средством являются векторные слайды КОМПАС-МАСТЕР, предназначенные для компактного представления команд для формирования изображения слайда.

После того, как стандартная деталь нарисована и запомнена в модели чертежа, пользователю может потребоваться отредактировать ее. Чаще всего обеспечивается редактирование по двойному щелчку и редактирование с помощью характерных точек.

Двойным щелчком по детали вызывается библиотечное окно параметров детали (рисунок 5.1).

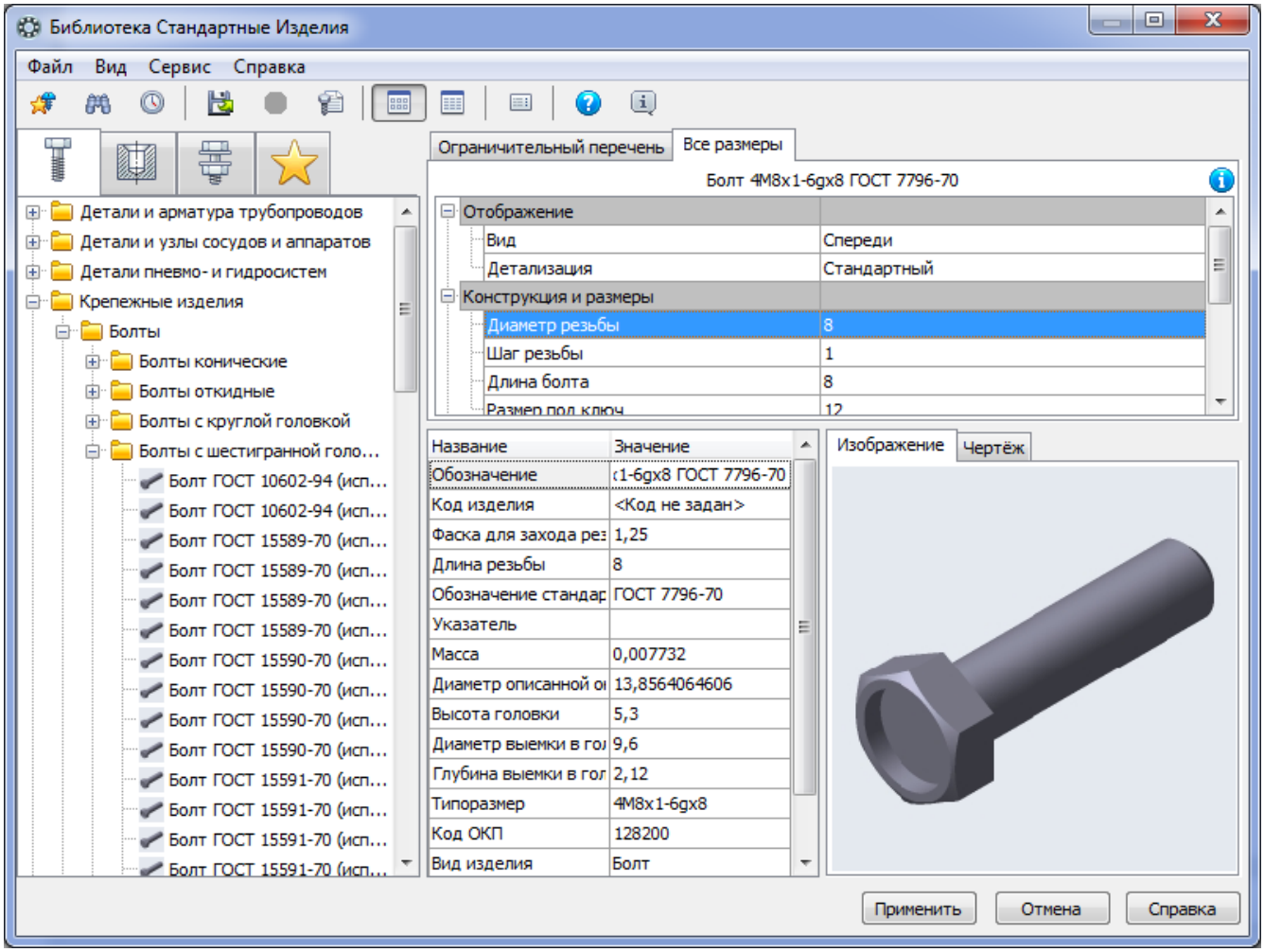



Рисунок 5.1 – Пример диалогового окна библиотеки для выбора параметров создания стандартной детали

При однократном щелчке на нарисованной детали на ней могут появиться характерные точки (если библиотека их поддерживает).

Пользователь может перетаскивать характерные точки. С каждой из них связан один или несколько параметров, которые при перетаскивании автоматически изменяются и библиотека соответствующим образом перестраивает изображение детали.

Например, у винта (рисунок 5.2) есть 4 характерные точки: точка O – местоположение детали, A – угол наклона, L – длина винта и Dr – диаметр резьбы. Важно, что при перетаскивании точек, связанных с дискретными параметрами (такими, как длина винта или диаметр – они определены ГОСТом), эти параметры меняются именно дискретно, а не с шагом указателя мыши. Таким образом, характерные точки являются удобным механизмом для изменения параметров нарисованной библиотечной детали «по месту».

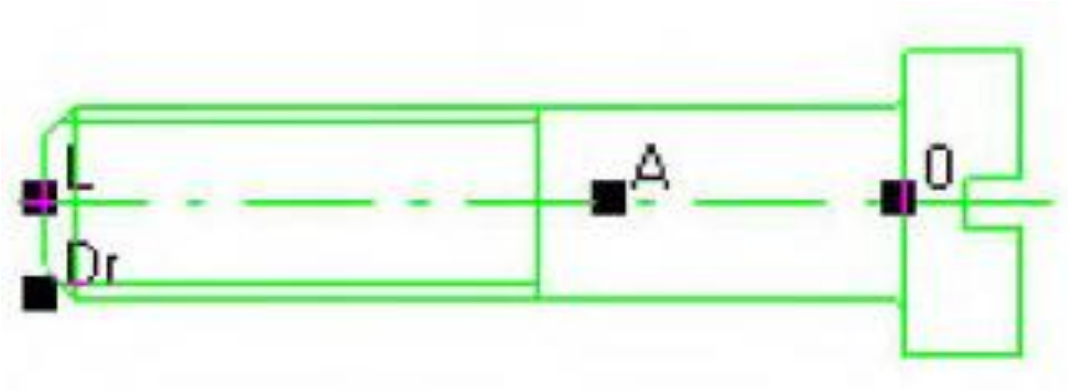



Рисунок 5.2 – Выделенное изображение библиотечной детали с 4-мя характерными точками

### 5.2 Стандартные диалоговые окна

Для выдачи информационных окон есть три функции `Message`, `MessageBoxResult` и `Error`.

Функция YesNo позволяет выдать сообщение и получить ответ как нажатие на кнопку «Да» или «Нет» – этот функция удобна для выдачи пользователю простых запросов на подтверждение некоторого действия.

Если в диалоговых окнах вводятся какие-нибудь данные, то результат ввода помещается либо в один из параметров функции, переданный по ссылке, либо передается в качестве возвращаемого значения.

При отмене окна пользователем функции выбора имен файлов возвращают пустую строку. Типичный фрагмент программы для выбора файла выглядит так:

```
void ReadFileName()// Выбрать имя файла
{
	char fileName[128];
	char buf[128];
	if (ChoiceFile("*.cdw", 0, fileName, 128))
	{
		sprintf_s(buf, "Выбран файл с именем %s\n", fileName);
		Message(buf);
	}
}
```

Часто применяются окна для ввода значения из заданного диапазона – функции с именами Read... . Например, ReadInt для ввода целого числа от 0 до 32000 со значением «по умолчанию» 100 можно вызвать так:

```
int X;
if (ReadIntT(_T("Введите координату X"), 100, 0, 32000, &X))
{
	// действия с координатой X
}
```

В таблице 5.1 приведены основные функции работы со стандартными диалоговыми окнами.

Таблица 5.1 – Функции для работы со стандартными диалоговыми окнами

| Имя функции | Назначение |
|---|---|
| Message | Вывести произвольное текстовое сообщение |
| MessageBoxResult | Вывести сообщение с объяснением результата работы (кода ошибки) последней вызванной функции КОМПАС-МАСТЕР |
| Error | Вывести текстовое сообщение об ошибке функций RTW-библиотеки |
| YesNo | Вывести строку сообщения в диалоговом окне и ожидать подтверждение или отказ от действия. |
| ksChoiceFile | Вызвать диалоговое окно выбора файла для чтения |
| ksChoiceFileAppointedDir | Вызвать диалог выбора файла для чтения с указанием каталога, с которого начинается выбор |
| ksChoiceFiles | Вызвать диалог выбора группы файлов для чтения (возвращает динамический массив интерфейсов ksChar255) |
| ksSaveFile | Вызвать диалог выбора файла для сохранения |
| ksInitFilePreviewFunc | Задать адрес пользовательской функции просмотра файла (для отображения в окне предварительного просмотра файлов, не являющихся документами КОМПАС) |
| ReadString | Ввести символьную строку |
| ReadInt | Ввести целое число с контролем попадания значения в заданный интервал |
| ReadLong | Ввести длинное целое число с контролем попадания значения в заданный интервал |
| ReadDouble | Ввести вещественное число с контролем попадания значения в заданный интервал |
| ksMaterialDlg | Получить параметры материала из Справочника материалов |

### 5.3 Отображение вариантов

Вариантом называется предварительное изображение библиотечной детали, выводимое тонкими линиями вместе с указателем мыши в процессе размещения детали пользователем в графическом документе. Получение от пользователя параметров местоположения (координат точки) выполняется с помощью функции `Cursor` или `Placement` (кроме координат точки позволяет задать значение угла наклона). Типичный вид окна КОМПАС-График в процессе размещения детали с помощью функций `Cursor`/`Placement` показан на рисунке 5.3.

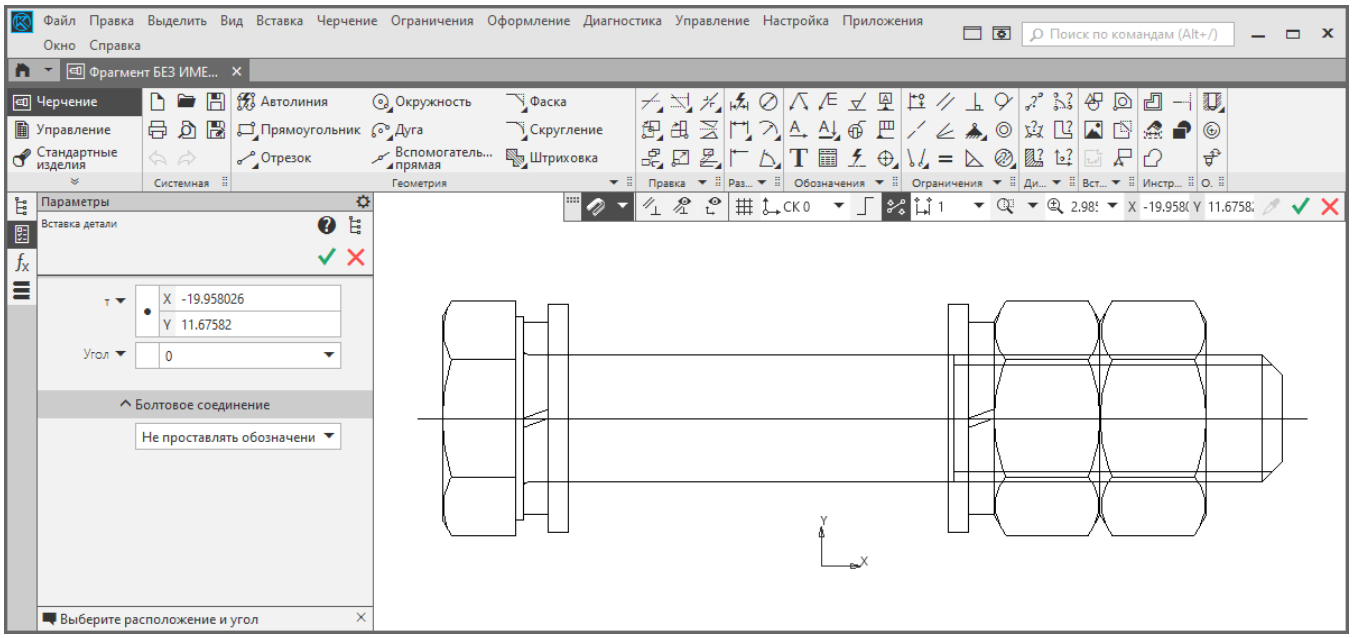



Рисунок 5.3 – Окно КОМПАС-График в процессе размещения библиотечной детали

На рисунке 5.3 присутствует изображение варианта болтового соединения, которое перемещается вместе с указателем мыши. В строке сообщений выводится подсказка пользователю о том, что он должен указать базовую точку детали.

Для выбора параметров соединения может быть создано командное окно.

Команды командного окна можно выбирать не только в окне, но и в контекстном меню в процессе размещения детали на чертеже.

Перечисленные элементы интерфейса - командное окно, содержимое строки сообщения, вариантное изображение и параметры, получаемые от пользователя, задаются свойствами интерактивного запроса. Запрос выполняется функциями `Cursor/Placement`. Эти функции имеют заголовки.

Указать положение объекта или определить вариант действия:

```
int Cursor(RequestInfo* info, double* x, double* y, void* phantom);
```

Входные параметры: `info` – указатель на структуру параметров запроса к системе; `x`, `y` – координаты введенной точки; `phantom` – указатель на структуру управления фантомом, определяющую тип движения курсора.

Возвращаемое значение: 1 – в случае удачного завершения; 0 – в случае неудачи.

Задать точку и угол:

```
int Placement(RequestInfo* info, double* x, double* y, double* angle, void* phantom);
```

Входные параметры: `info` – структуру параметров запроса к системе; `x`, `y` - координаты введенной точки; `phantom` – указатель на структуру управления фантомом, определяющую тип движения курсора; `angle` – введенный угол.

Возвращаемое значение: 1 – в случае удачного завершения; 0 – в случае неудачи.

Запрос к системе на получение точки:

```
int CursorEx(RequestInfo* info, double* x, double* y,
void* phantom, LPUNKNOWN processParam);
```

Входные параметры: `info` – указатель на структуру параметров запроса к системе `RequestInfo`; `x`, `y` – координаты введенной точки; `phantom` – указатель на структуру управления фантомом, определяющую тип движения курсора; `processParam` – указатель на интерфейс параметров процесса `IProcessParam`.

Выходные параметры: `x`, `y` – возвращаемые координаты точки.

Возвращаемое значение: -1 – если указана точка; 0 – отказ (Esc), идентификатор выбранной команды из командной строки или меню, определенный в файле ресурсов.

Запрос к системе на получение точки и угла

```
int PlacementEx(RequestInfo* info, double* x, double* y,
double* angle, void* phantom, LPUNKNOWN processParam);
```

Входные параметры: `info` – указатель на структуру параметров запроса к системе; `x`, `y` – координаты введенной точки; `angle` – введенный угол; `phantom` – указатель на структуру управления фантомом, определяющую тип движения курсора; `processParam` – указатель на интерфейс параметры процесса `ProcessParam`.

Выходные параметры: `x`, `y` – возвращаемые координаты точки; `angle` – возвращаемое значение угла.

Возвращаемое значение: 1 – в случае удачного завершения; 0 – в случае неудачи.

В интерфейсе `RequestInfo` передаются параметры, определяющие способ взаимодействия пользователя с окном КОМПАС-График - описание командного окна и содержимого строки сообщений.

В интерфейсе `phantom` указываются параметры вариантного изображения.

В переменных `x`, `y` и `angle` возвращаются координаты точки и угол наклона.

В качестве возвращаемого значения может быть возвращен код:

– 0: пользователь отменил выполнение команды;

– -1: пользователь задал местоположение и ориентацию;

– положительное значение: пользователь выбрал в командном окне команду с соответствующим порядковым номером (или код команды, если содержимое командного окна было задано с помощью ресурса меню).

Рассмотрим интерфейс параметров интерактивного запроса `RequestInfo`.

Основное назначение этого интерфейса - управление командным окном, которое показывается в немодальном диалоговом окне в процессе выполнения запроса. Пользователь может не только сдвинуть и повернуть вариантное изображение, но и выбрать двойным щелчком команду из этого окна.

Команды, помещаемые в окно, надо задать в виде строки в свойстве `RequestInfo.commands`. Даже если команд много, все они задаются одной строкой – перед именем каждой команды ставится восклицательный знак.

В свойстве RequestInfo.title задается заголовок командного окна, в свойстве prompt – содержимое строки - приглашения.

Вариантное изображение описывается интерфейсом параметров Phantom. Вариантное изображение может быть нескольких типов, который надо указать в свойстве phantom. В соответствии со значением этого свойства, внутри интерфейса Phantom надо заполнить структуру параметров варианта именно этого типа (Type1, Type2, ... или Type6). Допустимые значения phantom и соответствующие подчиненные интерфейсы перечислены в таблице 5.2.

Таблица 5.2 – Допустимые типы вариантов

| Тип варианта phantom | Назначение варианта | Подчиненный интерфейс |
|---|---|---|
| 1 | параметры сдвига группы | Type1 |
| 2 | параметры фантома-отрезка | Type2 |
| 3 | параметры фантома-прямоугольника | Type3 |
| 4 | параметры фантома-отрезка с заданным углом | Type3 |
| 5 | параметры фантома-половины прямоугольника | Type5 |
| 6 | параметры пользовательского фантома | Type6 |
| 7 | параметры фантома-окружности | Type2 |

Наиболее универсальным вариантом является вариант 1 – в интерфейсе Type1 можно задать дескриптор произвольной группы, которая будет обрисовываться с учетом смещения и поворота, которые задал пользователь с помощью мыши. Остальные типы позволяют, например, отобразить вариант в виде отрезка с зафиксированным концом или в виде прямоугольника с зафиксированным углом. Варианты типов 2-7 обычно применяют с функцией Cursor. Часто варианты этих типов – отрезок, окружность и прямоугольник – называются типами резиновой нити.

Основные приемы работы с вариантным изображением и командным окном показаны в процедуре Demo_Phantom (листинг 5.1). Эта процедура позволяет нарисовать в текущем графическом документе окружность, треугольник, квадрат или от-

резок фиксированного размера. Для треугольника, квадрата и отрезка можно указать местоположение и ориентацию, для окружности – только местоположение. Не выходя из этой процедуры, можно нарисовать произвольное количество любых фигур. Для этого выполняется анализ возвращаемого значения функции `Placement`.

Листинг 5.1 – Основные приемы работы с вариантами

```
int g_type = 1; // Глобальная переменная - тип текущей фигуры
// Обновляется не только изображение но и меню для запроса
void Update_Cursor(reference& rGroup,  // Группа
     RequestInfo& info) // Структура параметров запроса к системе
{
     // Группа для фантома должна быть временная и обновляться при изменении вида отрисовки
     if (rGroup)
          DeleteObj(rGroup);
     // Создание группы объектов, type - тип группы ( 0 - определяет модельный, 1 - временный )
     rGroup = NewGroup(1);
     switch (g_type)
     {
     case 0:
          // Состав группы - Квадрат
          LineSeg(-10, 0, 10, 0, 1);
          LineSeg(10, 0, 10, 20, 1);
          LineSeg(10, 20, -10, 20, 1);
          LineSeg(-10, 20, -10, 0, 1);
          info.commands = ("!Окружность !Треугольник !Отрезок"); // меню
          break;
     case 1:
          Circle(0, 0, 20, 1); // Состав гуппы - Окружность
          info.commands = ("!Квадрат !Треугольник !Отрезок"); // меню
          break;
     case 2:
          // Состав гуппы - Треугольник
          LineSeg(-10, 0, 10, 0, 1);
          LineSeg(10, 0, 0, 20, 1);
          LineSeg(0, 20, -10, 0, 1);
```

```
            // Отображаемое меню
            info.commands = ("!Окружность !Квадрат !Отрезок");
            break;
      case 3:
            // Состав гуппы - Отрезок
            LineSeg(0, 0, 20, 0, 1);
            // Отображаемое меню
            info.commands = ("!Окружность !Треугольник !Квадрат");
            break;
      }
      EndGroup(); // Завершить создание группы объектов
}
void Demo_Phantom()
{
      Phantom phantom; // Структура параметров фантома
      phantom.phType = 1; // Тип фантома ( type1, type2, ... type7 )
      phantom.type1.xBase = 0; // Координаты начальной точки группы
      phantom.type1.yBase = 0;
      phantom.type1.scale = 1; // Масштаб
      phantom.type1.gr = 0; // Указатель на группу
      phantom.type1.ang = 0; // Угол поворота группы
      // Структура параметров запроса к системе
      RequestInfo info;
      memset(&info, 0, sizeof(info));
      double x, y;  // Координата вставки
      int comm = 1; // Пришедшая команда
      while (comm)
      {// Обновляется не только изображение, но и меню для запроса
            Update_Cursor(phantom.type1.gr, info);
            if (g_type == 1)
            {// Пример с использованием Cursor
            // Ввод точки или команды
                  comm = Cursor(&info,            // Указатель на структуру па-
раметров запроса к системе
                        &x, &y,           // Координаты введенной точки
                        &phantom); // Указатель на структуру управления фанто-
мом, определяющую тип движения курсора
```

```
            }
            else
            {// Пример с использованием Placemant
            // Задание точки, угла или команды
                  comm = Placement(&info,              // Указатель на структу-
ру параметров запроса к системе
                        &x, &y,              // Координаты введенной точки
                        &phantom.type1.ang, // Введенный угол
                        &phantom);          // Указатель на структуру управления
фантомом, определяющую типдвижения курсора
            }
            // Выполнение команды
            if (comm == -1) // Поставить в модель
            {// Сдвинуть и повернуть группу
                  MoveObj(phantom.type1.gr, x, y);
                  if (fabs(phantom.type1.ang) > 0.001)
                        RotateObj(phantom.type1.gr, x, y, phantom.type1.ang);
                  // Поставить временную группу в вид
                  StoreTmpGroup(phantom.type1.gr);
                  // Очистить группу объектов
                  ClearGroup(phantom.type1.gr);
            }
            else // Сменить тип фигуры
            {
                  if ((g_type == 1 && comm == 1) || (g_type == 2 && comm == 2)
|| (g_type == 3 && comm == 3))
                        g_type = 0;
                  else
                        g_type = comm;
            }
      }
}
```

Результат работы программы показан на рисунке 5.4.

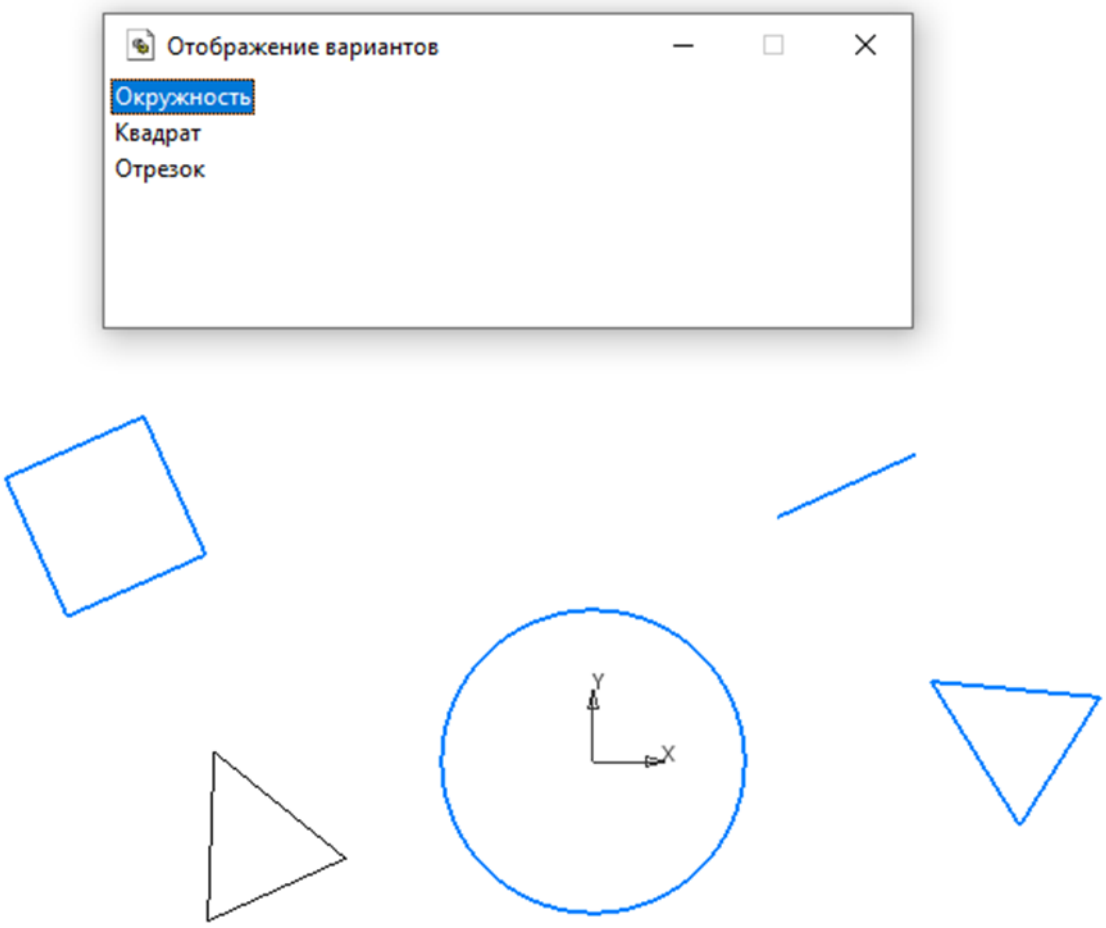



Рисунок 5.4 – Результат работы программы

Процедура `Demo_Phantom` показывает способ использования `Cursor/Placement` без функции обратной связи. Процесс указания точки функция `Cursor/Placement` – многократно запускается во внешнем (по отношению к КОМПАС) цикле прикладной библиотеки, и выбранная команда обрабатывается вне процесса.

Недостаток данного способа заключается в том, что командное окно заметно исчезает и появляется снова, так как создается при каждом вызове `Cursor/Placement`. При использовании функции обратной связи процесс указания точки запускается один раз. Обработка команды происходит в функции обратной связи. Процесс указания точки не завершается, пока функция обратной связи не вернет значение 0.

Команды в командном окне нумеруются, начиная с 1. С учетом этого обстоятельства и перечисления команд в строке `Requestlnfo: commands` в процедуре `Demo_Phantom` определена глобальная переменная – тип текущей фигуры `g_type`. Для меню команд:

```
info.commands = _T("!Окружность !Треугольник !Отрезок");
```

`g_type=1` – соответствует Окружности,

`g_type=2` – Треугольнику,

`g_type=3` – Отрезку.

Для последующих фигур перечисление команд меняется таким образом, чтобы номер команды построения квадрата соответствовал номеру текущего типа фигуры. Это требование проверяется в соответствующем условии и при равенстве типа текущей фигуры и номера выбранной команды, переменной `g_type` присваивается 0, что соответствует фигуре Квадрат:

```
if ((g_type == 1 && comm == 1) || (g_type == 2 && comm == 2) ||
(g_type == 3 && comm == 3))
    g_type = 0;
else
    g_type = comm;
```

В `Demo_Phantom` группа, описывающая вариантное изображение, создается и уничтожается на каждой итерации цикла. Если пользователь успешно разместил вариант, то перед уничтожением группа запоминается в документе.

### 5.4 Задание для лабораторной работы

1 Изучить теоретический материал.

2 Разработать прикладную библиотеку КОМПАС, которая будет формировать меню с использованием функции обратной связи для размещения графических объектов в окне КОМПАС в соответствии с заданным вариантом, приведенным в приложении Г.

3 Подготовить ответы на контрольные вопросы.

### 5.5 Содержание отчета

В отчете по лабораторной работе согласно СТО 02069024.101–2015 РАБОТЫ СТУДЕНЧЕСКИЕ [3] должны содержаться следующие пункты:

– название лабораторной работы;

– цель работы;

– задание на лабораторную работу;

– код разработанной программы с комментариями;

– экранные формы с отображением результата выполнения программы в системе КОМПАС;

– выводы;

– список использованных источников.

### 5.6 Контрольные вопросы

1 Назовите функции для работы со стандартными диалоговыми окнами.

2 На какие три группы можно разделить операции ввода данных?

3 Как может выполняться редактирование уже созданной библиотечной детали?

4 Что такое вариантное изображение?

5 Объясните параметры функции Cursor.

6 Объясните параметры функции Placement.

7 Опишите структуру параметров RequestInfo.

8 Опишите структуру параметров Phantom.

# Список использованных источников

1 Богуславский, А.А. Инструментальные средства разработки прикладных САПР КОМПАС-МАСТЕР 5 [Электронный ресурс] / А.А. Богуславский. - Коломна: Коломенский государственный педагогический институт, 2002. – URL: https://old.support.ascon.ru/library/documentation/items/?dl_id=97

2 Кидрук, М. Компас - 3D V10 на 100 % [Комплект] / М. Кидрук. – Санкт Петербург : Питер, 2009. - 560 с. : ил. + 1 электрон. опт. диск (CD-ROM).

3 СТО 02069024.101–2015 РАБОТЫ СТУДЕНЧЕСКИЕ. Общие требования и правила оформления

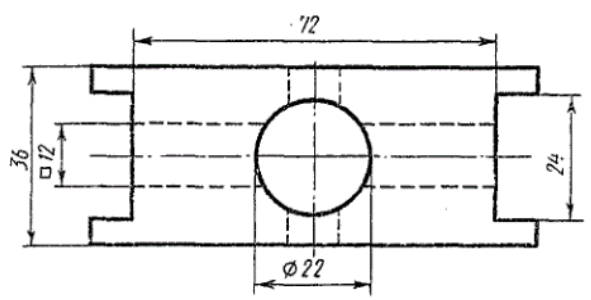
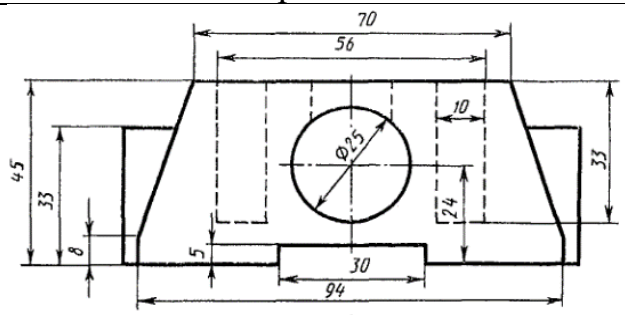
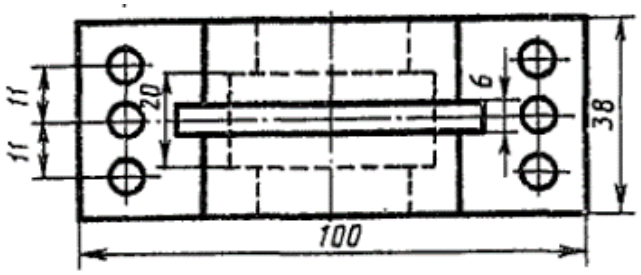
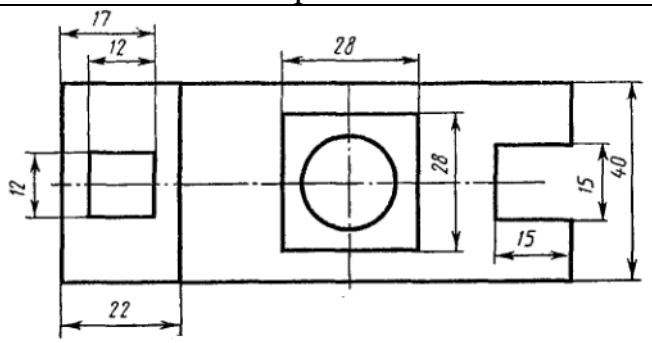
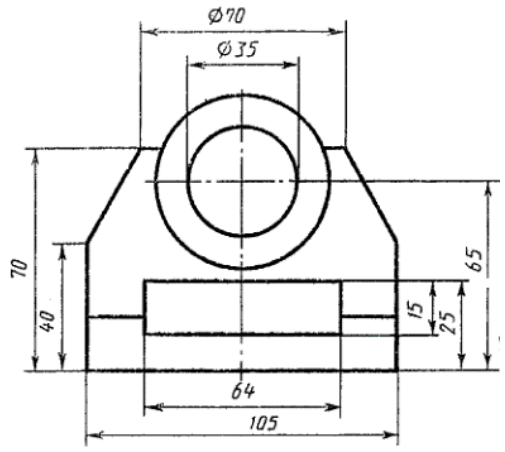
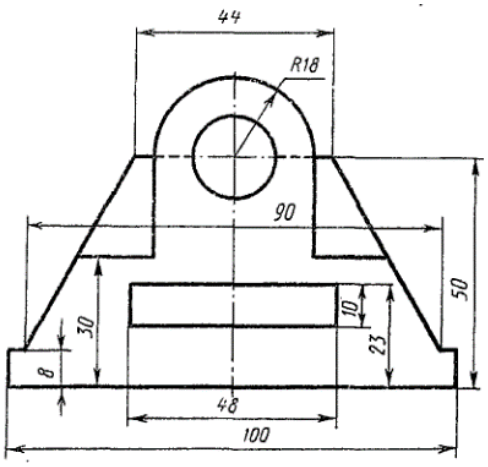
4 Боголюбов, С.К. Индивидуальные задания по курсу черчения: Учебное пособие для средних специальных учебных заведений. 3-е изд., стереотипное. Перепечатка со второго издания 1994 г. / С.К. Боголюбов. – М. : ООО ИД «Альянс», 2007. – 368 с.

## Приложение А
## (обязательное)

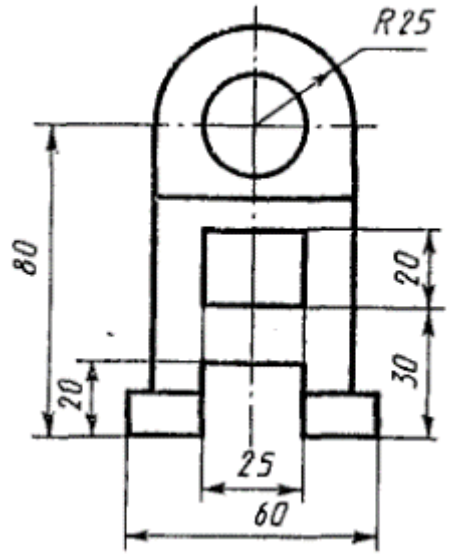
### Задания для построения отрезков и дуг окружностей функциями КОМПАС

Варианты заданий приведены из учебного пособия [4]. Штриховые линии строить не требуется.

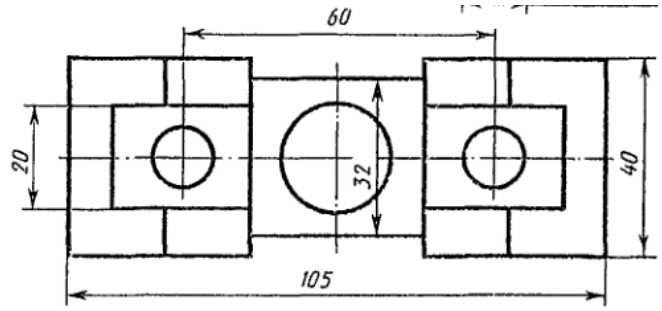
Таблица А.1 – Варианты заданий для построения отрезков и дуг окружностей функциями КОМПАС

| Вариант 1 | Вариант 2 |
|---|---|
|  |  |
| Вариант 3 | Вариант 4 |
|  |  |
| Вариант 5 | Вариант 6 |
|  |  |

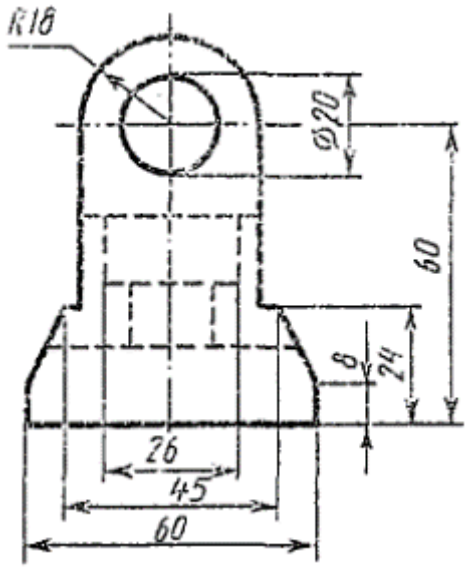
| Вариант 7 | Вариант 8 |
|---|---|



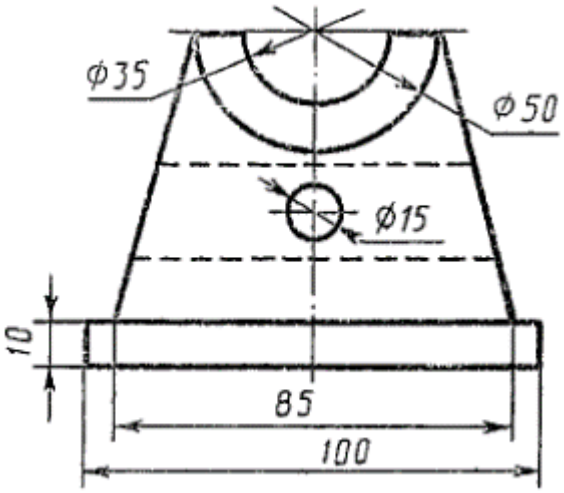

Вариант 9



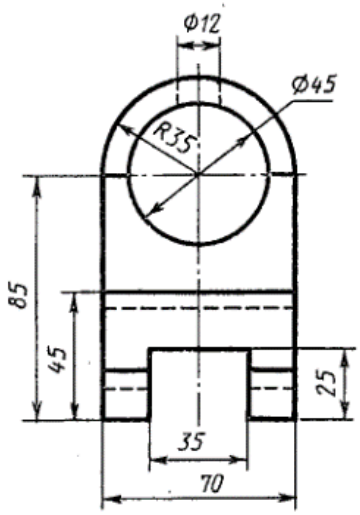

Вариант 10



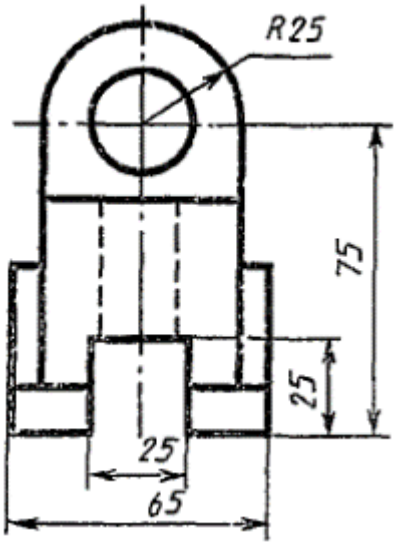

Вариант 11



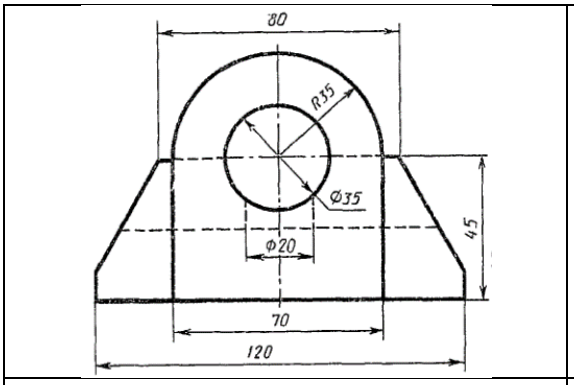

Вариант 12



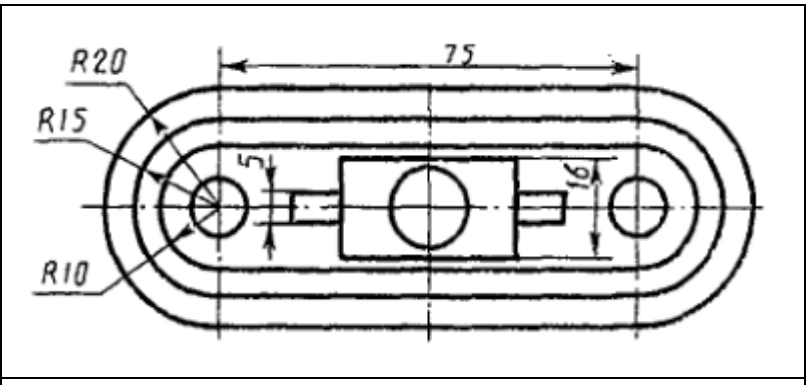

Вариант 13



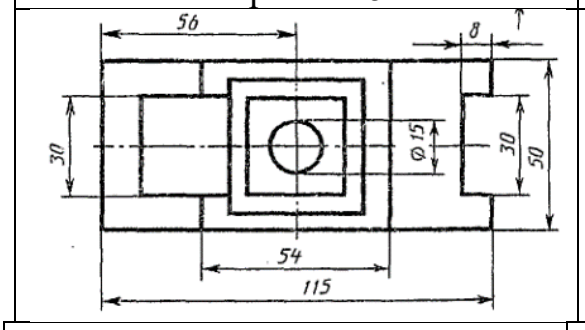

Вариант 14



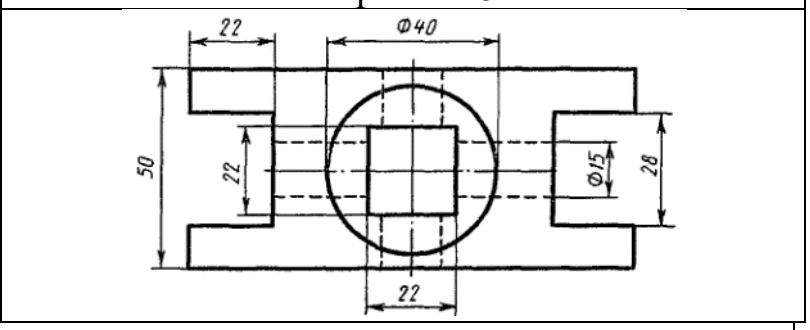

Вариант 15



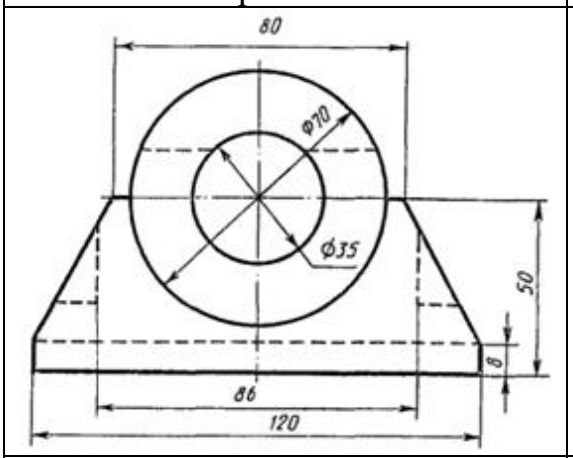

Вариант 16



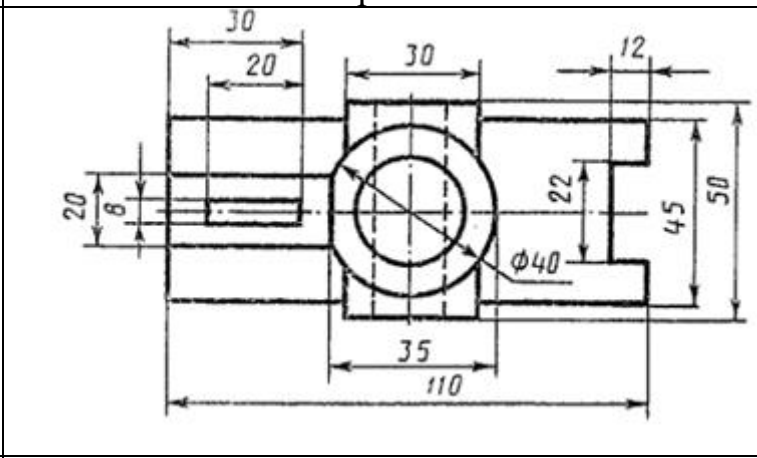

Вариант 17



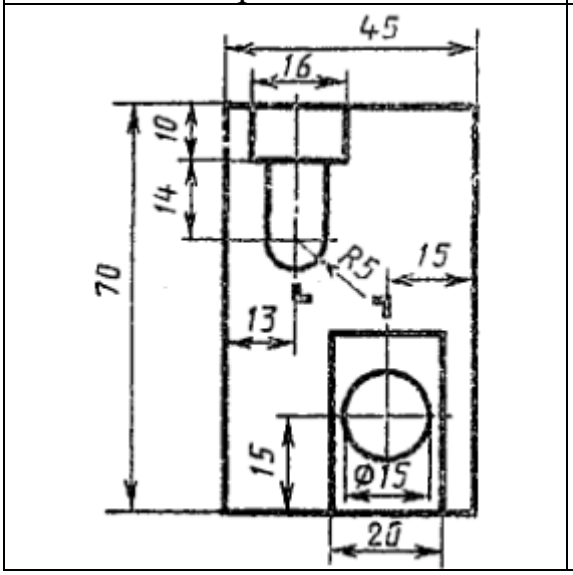

Вариант 18

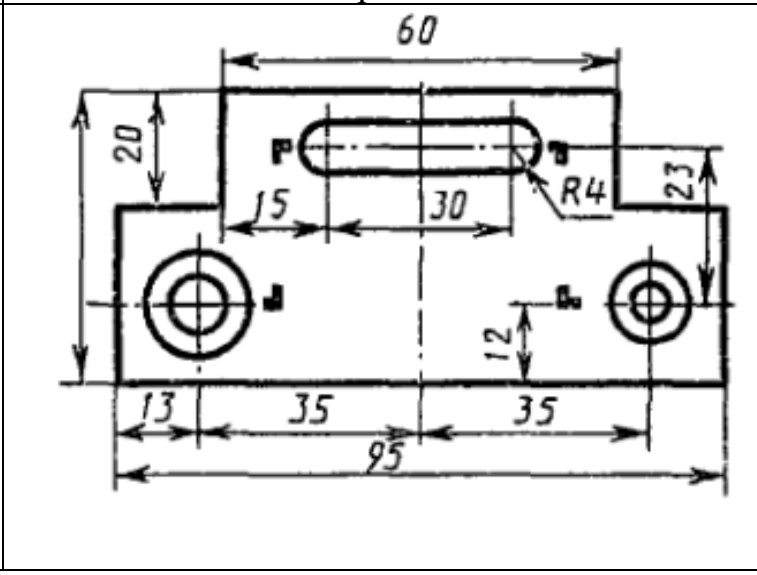



Вариант 19



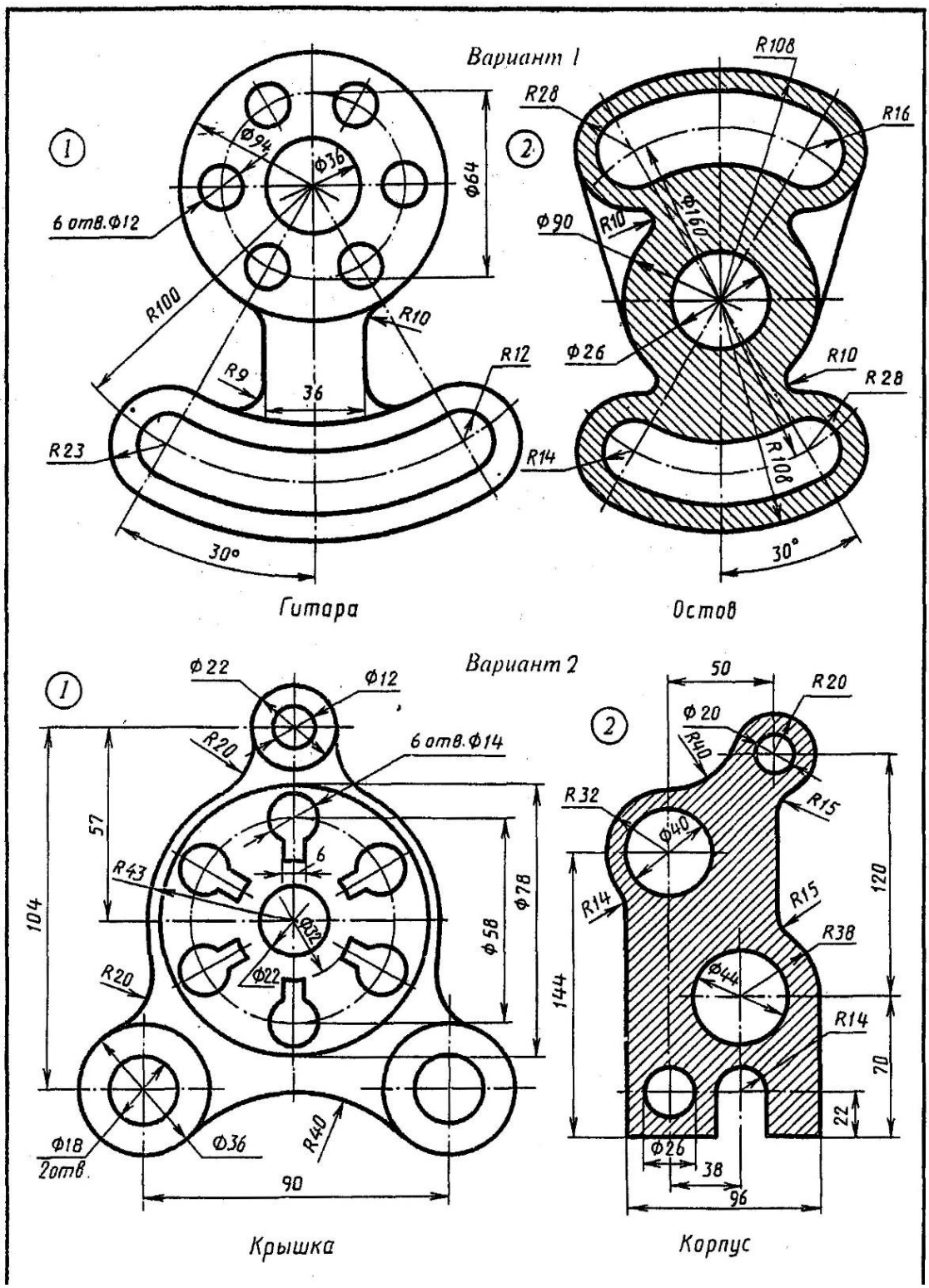

Вариант 20

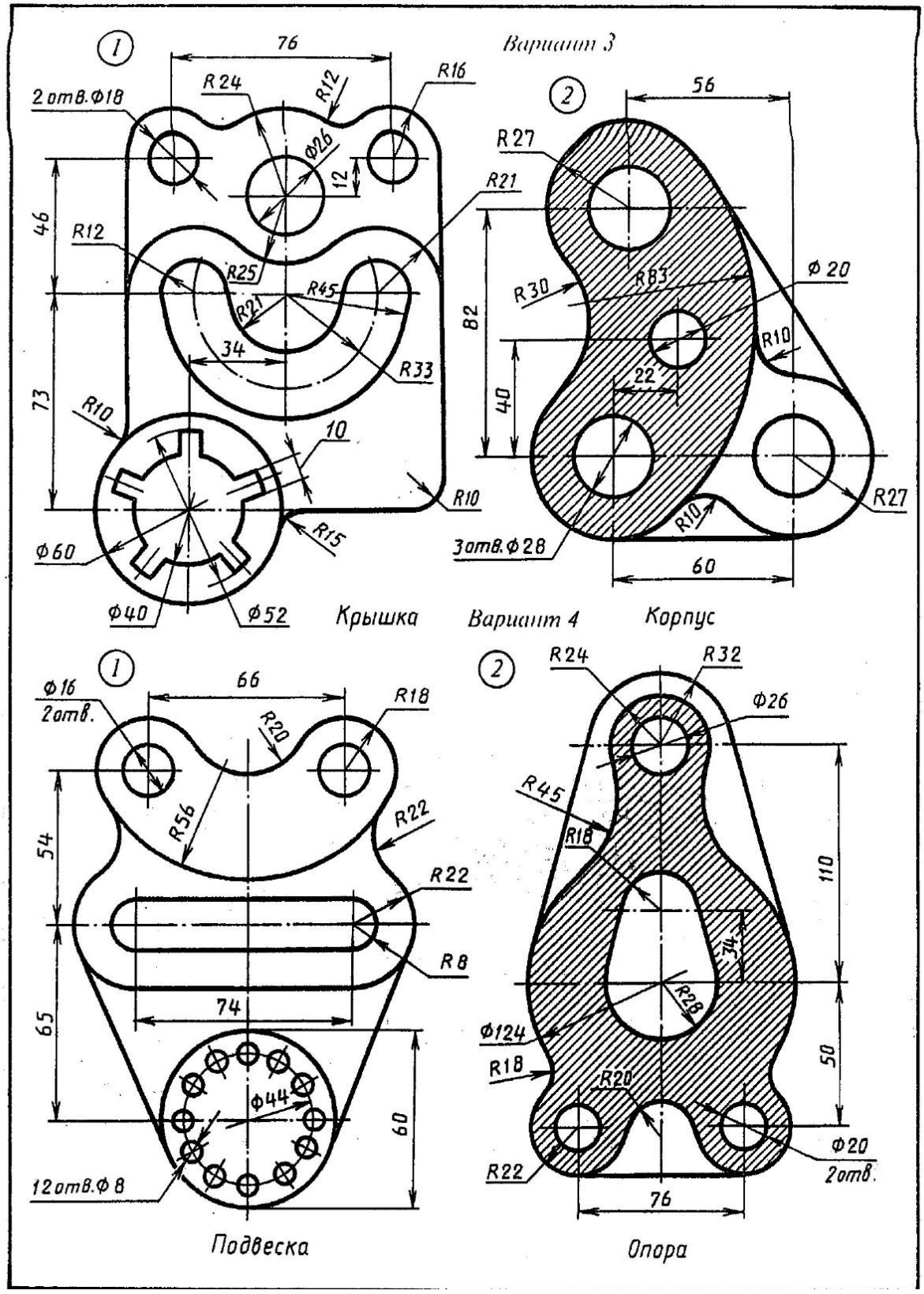


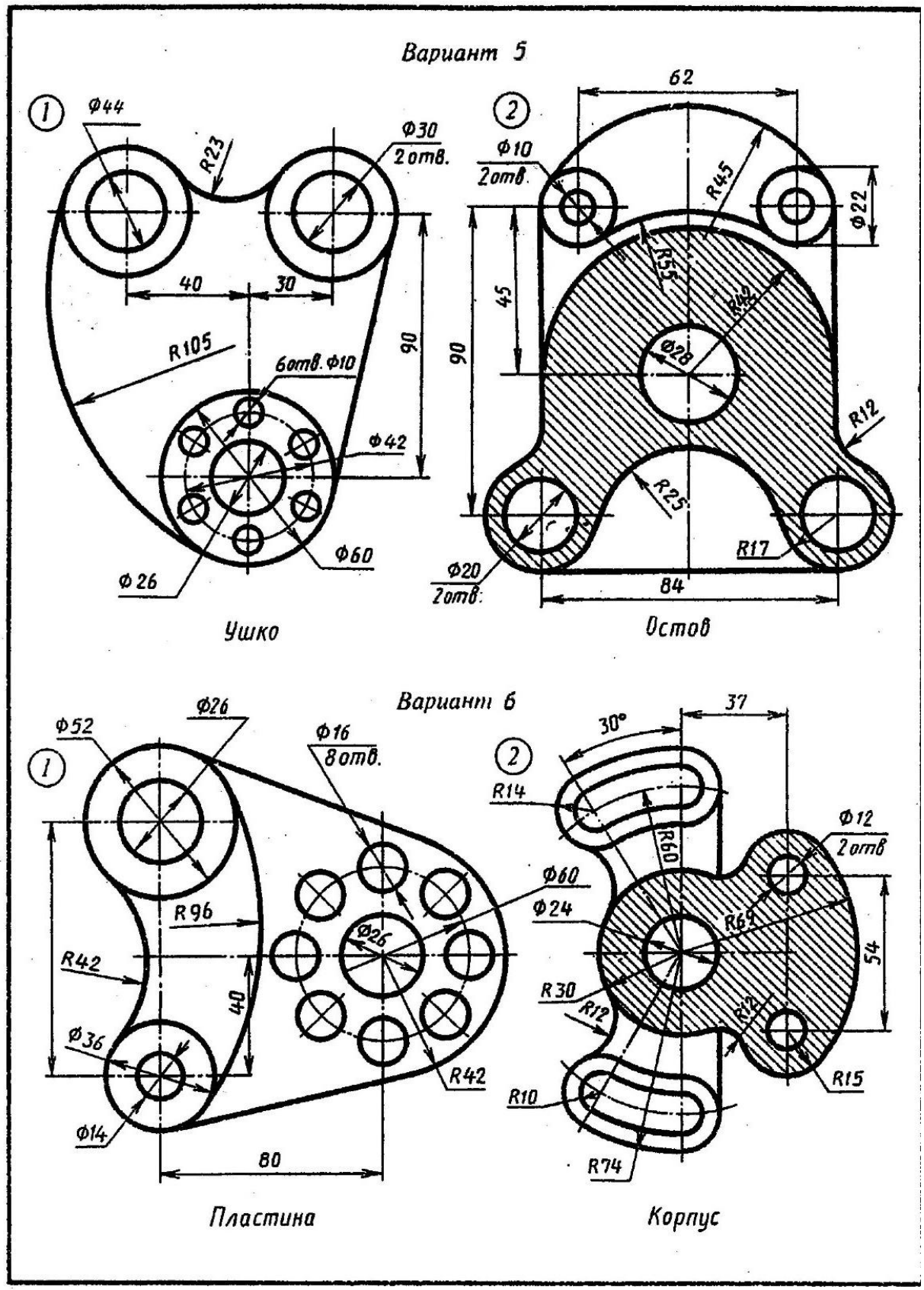



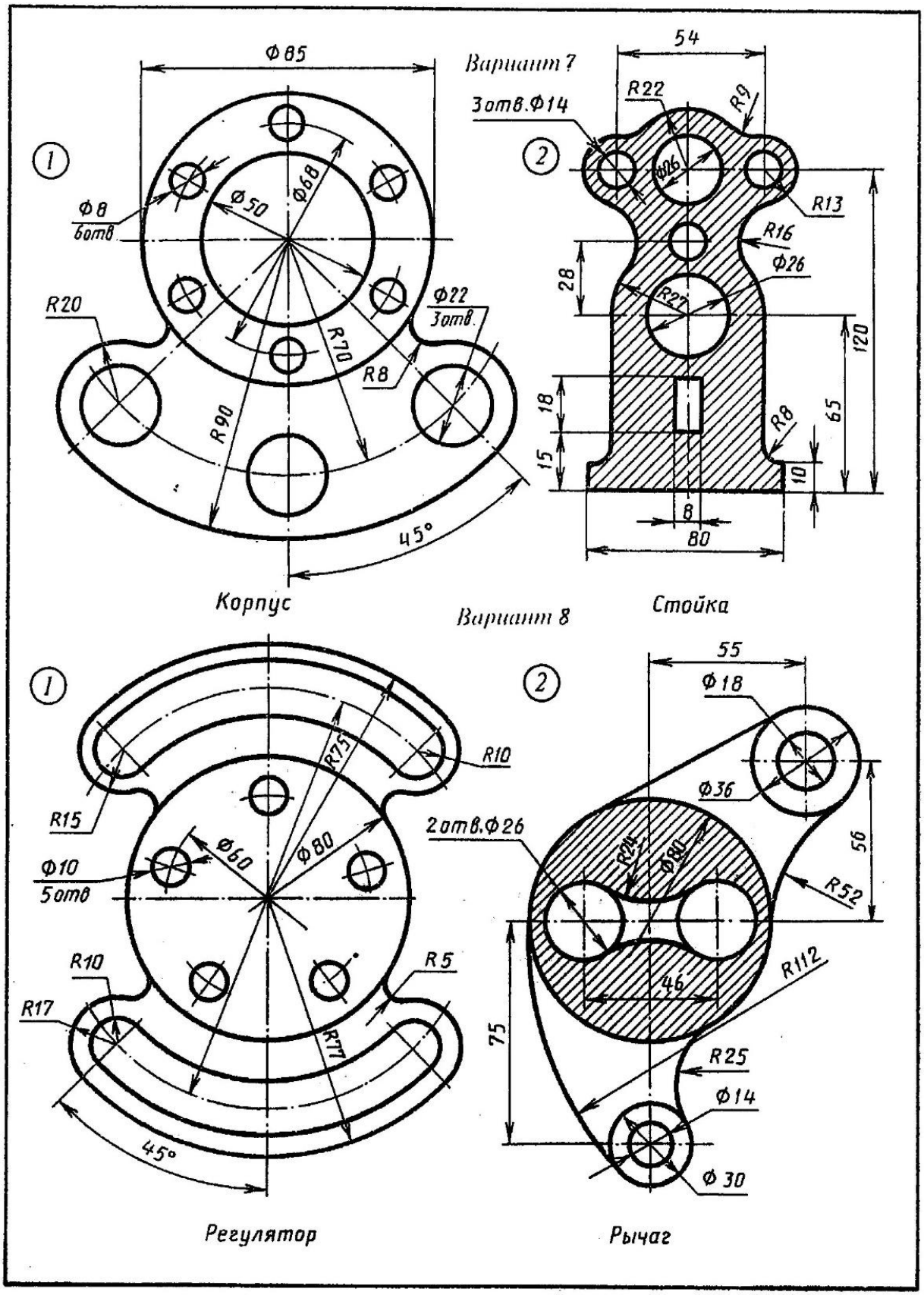
**Приложение Б**
**(обязательное)**

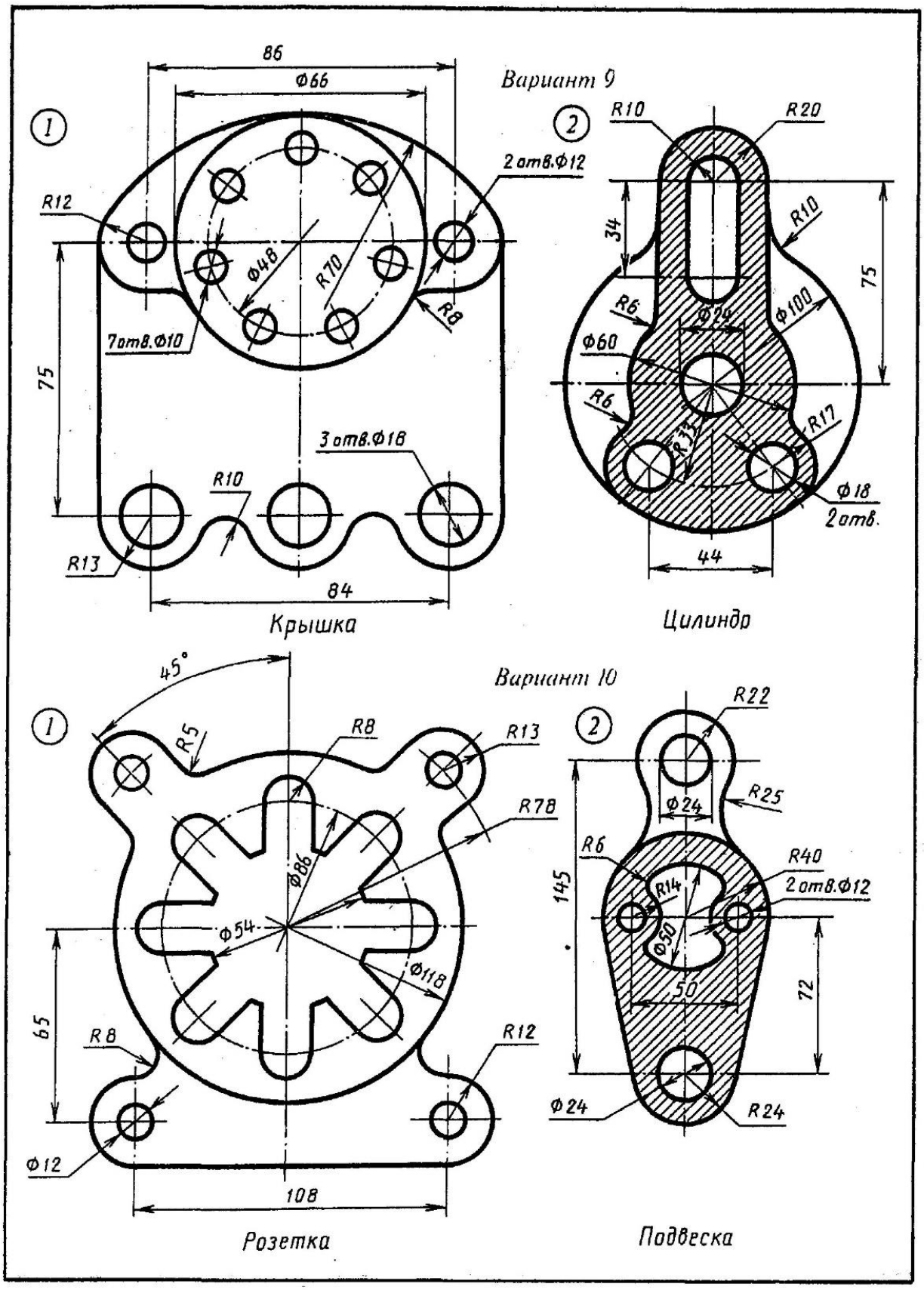
**Задания для работы со вспомогательными построениями и оформлением чертежа**

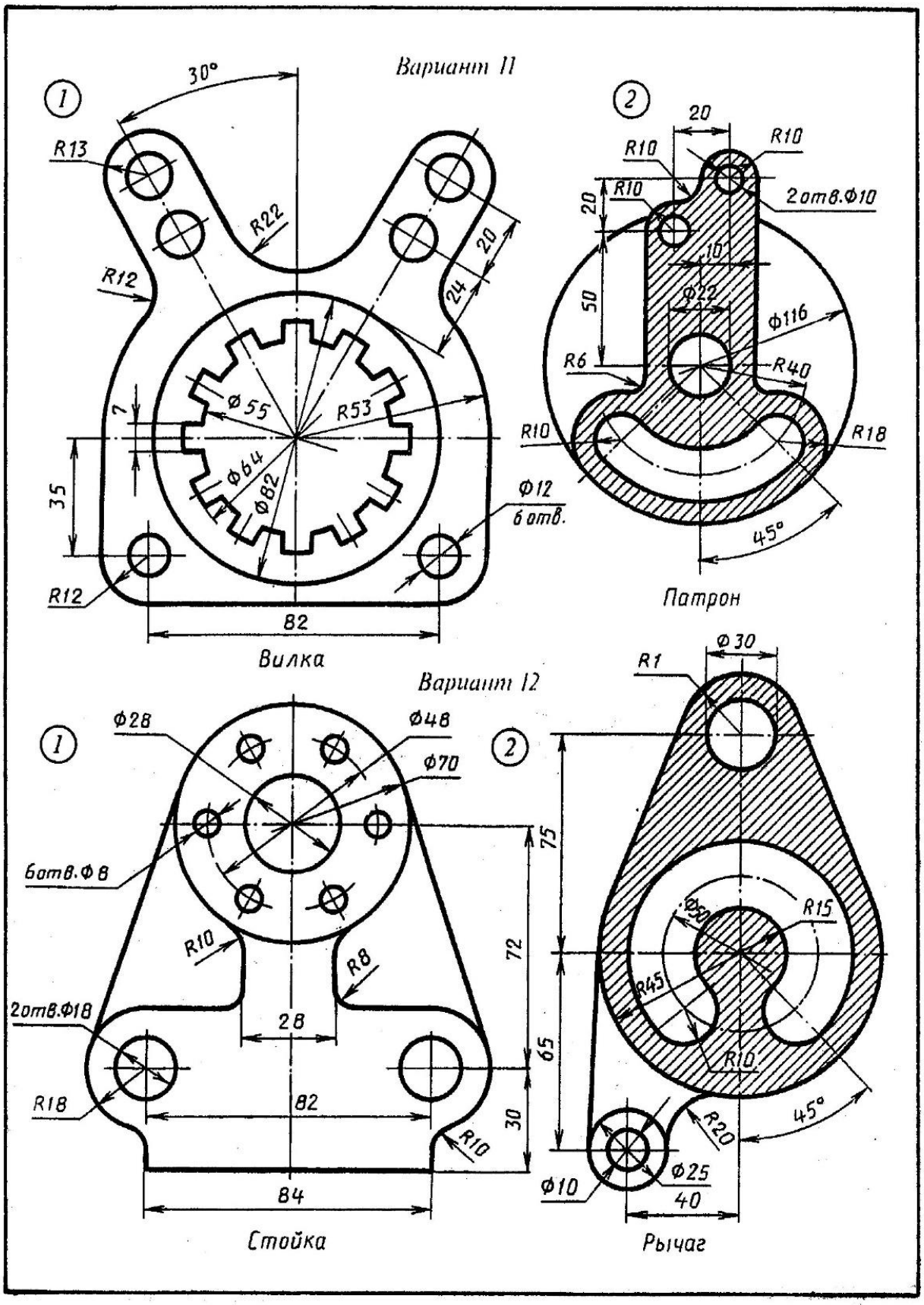


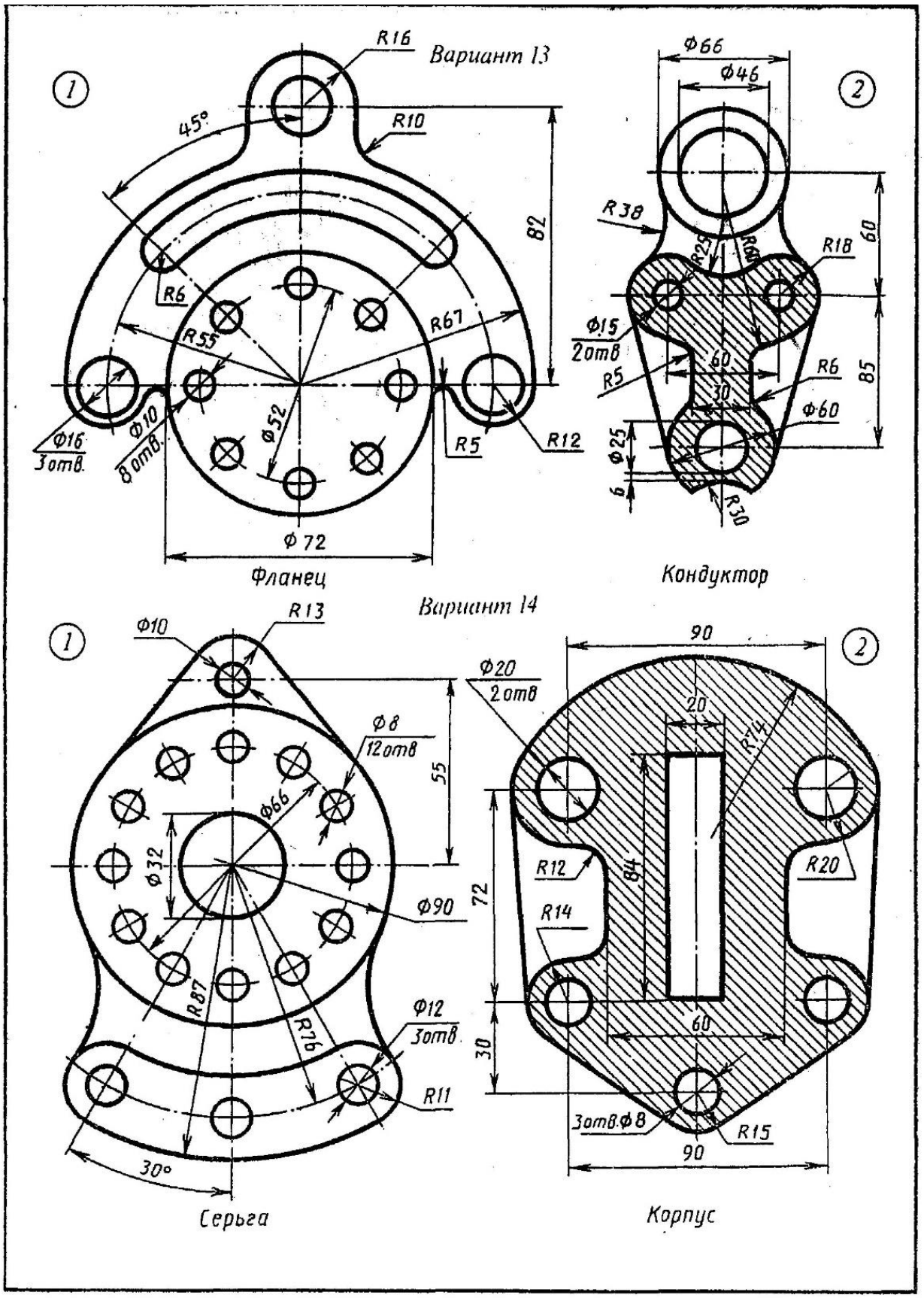

Вариант 3
1
76
R24
R12
R16
2 отв. Ф18
Ф26
12
46
R12
R25
R45
R21
34
R33
73
R10
10
R10
R15
Ф60
Ф40
Ф52
Крышка
2
56
R27
R21
Ф20
R30
R83
82
R10
40
22
R27
R10
3 отв. Ф28
60
Вариант 4
Корпус
1
Ф16
2 отв.
66
R18
R20
R56
R22
54
R22
R8
74
65
Ф44
60
12 отв. Ф8
Подвеска
2
R24
R32
Ф26
R45
R18
110
34
R28
Ф124
50
R18
R20
Ф20
2 отв.
R22
76
Опора

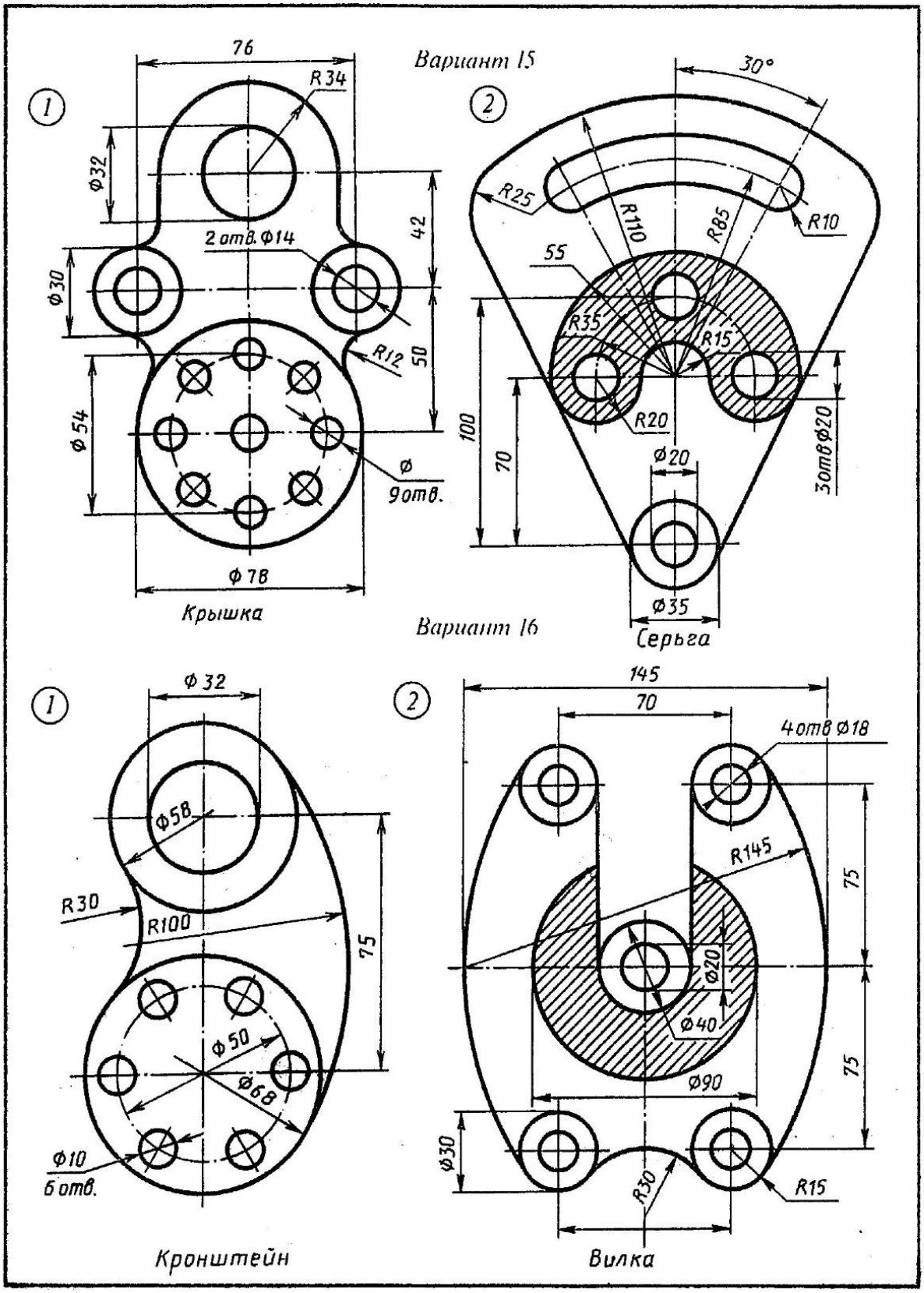

Вариант 5
1
Ф44
R23
Ф30
2 отв.
40
30
90
R105
6 отв. Ф10
Ф42
Ф60
Ф26
Ушко
2
62
Ф10
2 отв.
R45
Ф22
R55
45
90
R12
Ф28
R12
R25
R17
Ф20
2 отв.
84
Остов
Вариант 6
1
Ф52
Ф26
Ф16
8 отв.
R96
R42
40
Ф36
Ф14
Ф26
R42
80
Пластина
2
30°
37
R14
R60
Ф12
2 отв
Ф60
Ф24
R69
54
R30
R12
R12
R15
R10
R74
Корпус

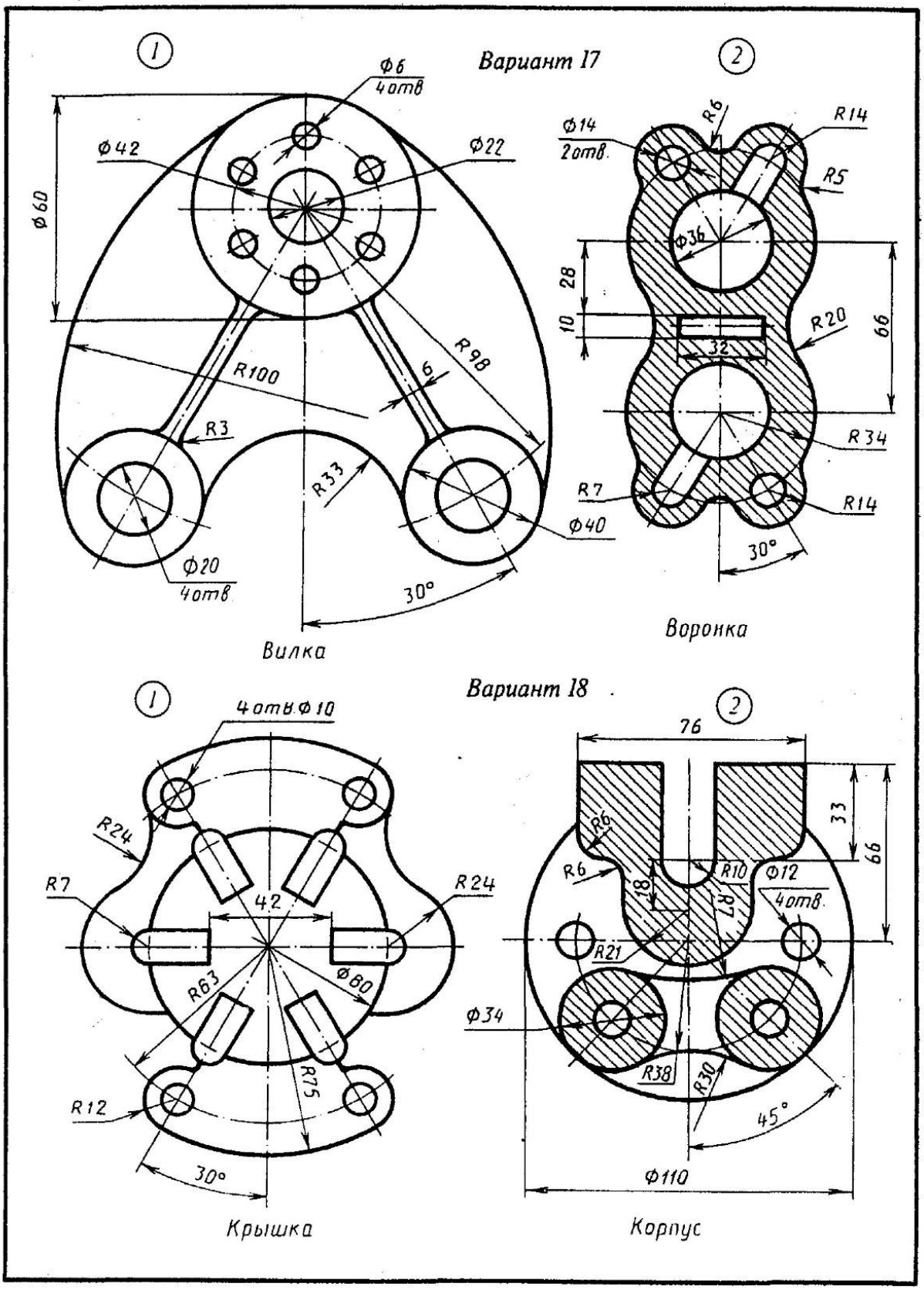

Вариант 7
Ф85
Ф8
6отв.
Ф50
Ф68
R20
Ф22
3отв.
R70
R8
R90
45°
Корпус
54
3отв.Ф14
R22
R9
Ф26
R13
R16
28
R27
120
18
15
65
R8
10
8
80
Стойка
Вариант 8
R75
R10
R15
Ф10
5отв
Ф60
Ф80
R10
R17
R5
R77
45°
Регулятор
55
Ф18
Ф36
2отв.Ф26
R24
Ф80
56
R52
R112
46
75
R25
Ф14
Ф30
Рычаг

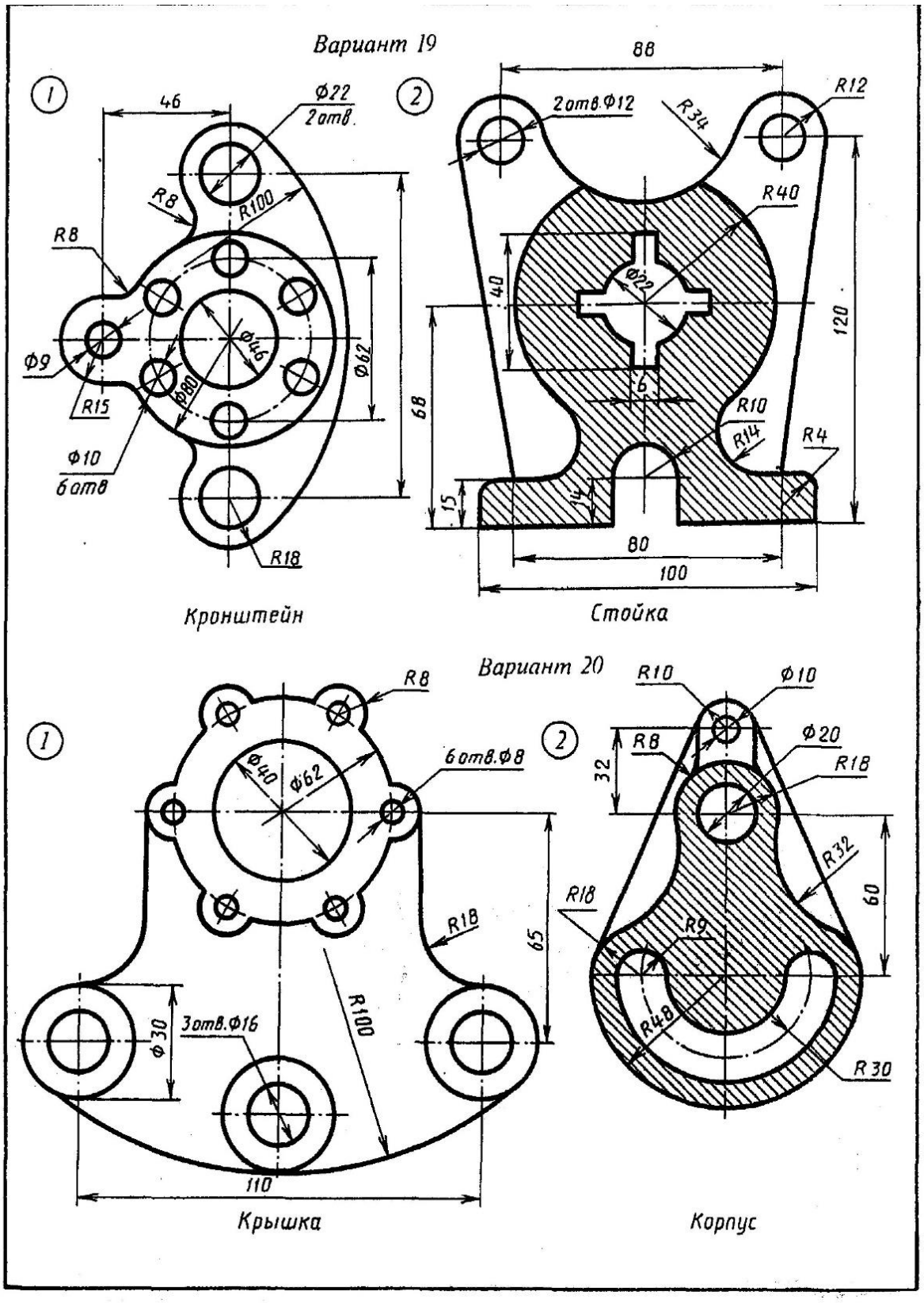

Вариант 9
1
86
Ф66
2 отв.Ф12
R12
Ф48
R70
R8
7 отв.Ф10
75
3 отв.Ф16
R10
R13
84
Крышка
2
R10
R20
34
R10
75
R6
Ф24
Ф100
Ф60
R6
R17
R37
Ф18
2 отв.
44
Цилиндр
Вариант 10
1
45°
R5
R8
R13
R78
Ф86
Ф54
Ф118
65
R8
R12
Ф12
108
Розетка
2
R22
Ф24
R25
R6
R40
145
R14
2 отв.Ф12
Ф50
50
72
Ф24
R24
Подвеска


Вариант 11
1
30°
R13
R22
R12
20
24
Ø55
R53
7
35
Ø64
Ø82
Ø12
6 отв.
R12
82
Вилка
2
20
R10
R10
R10
2 отв.Ø10
20
10
50
Ø22
Ø116
R6
R40
R10
R18
45°
Патрон
Вариант 12
1
Ø28
Ø48
Ø70
6 отв.Ø8
R10
R8
2 отв.Ø18
28
R18
82
R10
84
75
72
65
30
Стойка
2
Ø30
R1
Ø50
R15
R45
R10
45°
R20
Ø25
Ø10
40
Рычаг

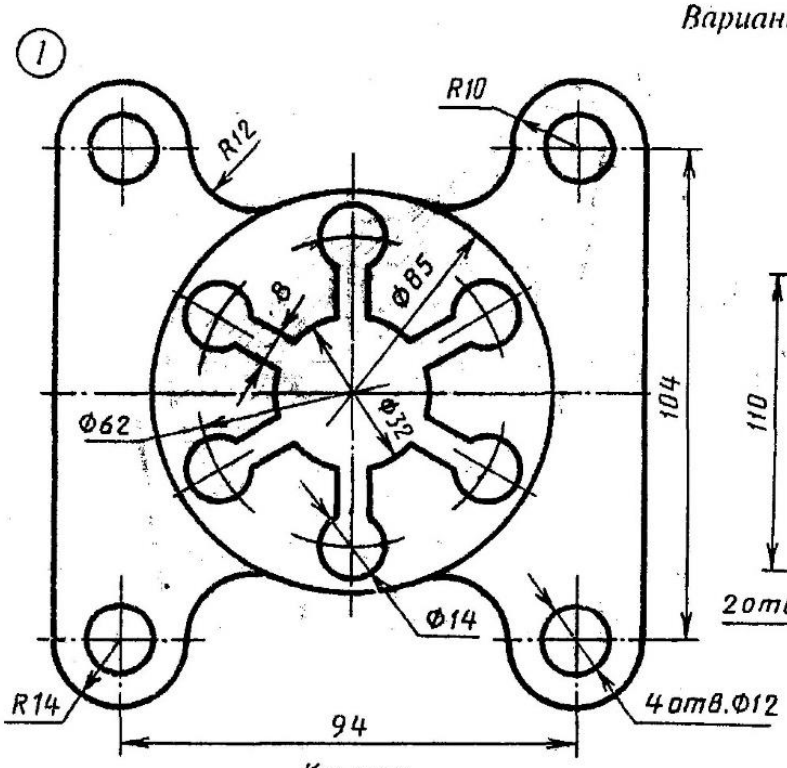

Вариант 13
1
R16
45°
R10
82
R6
R55
R67
Ф16
3 отв.
Ф10
8 отв.
Ф52
R5
R12
Ф72
Фланец
2
Ф66
Ф46
R38
R27
R60
R18
60
Ф15
2 отв
R5
60
30
R6
Ф60
85
Ф25
6
R30
Кондуктор
Вариант 14
1
Ф10
R13
Ф8
12 отв
55
Ф66
Ф32
Ф90
R87
R76
Ф12
3 отв
R11
30°
Серьга
2
90
Ф20
2 отв
20
R14
R12
84
R20
72
R14
30
60
3 отв Ф8
R15
90
Корпус


Вариант 15
1
76
R34
ф32
42
2 отв. ф14
ф30
R12
50
ф54
ø
9 отв.
ф78
Крышка
2
30°
R25
R110
R85
R10
55
R35
R15
R20
100
70
ø20
3 отв ø20
ø35
Серьга
Вариант 16
1
ф32
ф58
R30
R100
75
ф50
ø68
ф10
6 отв.
Кронштейн
2
145
70
4 отв ø18
R145
75
ø20
ø40
ø90
75
ø30
R30
R15
Вилка

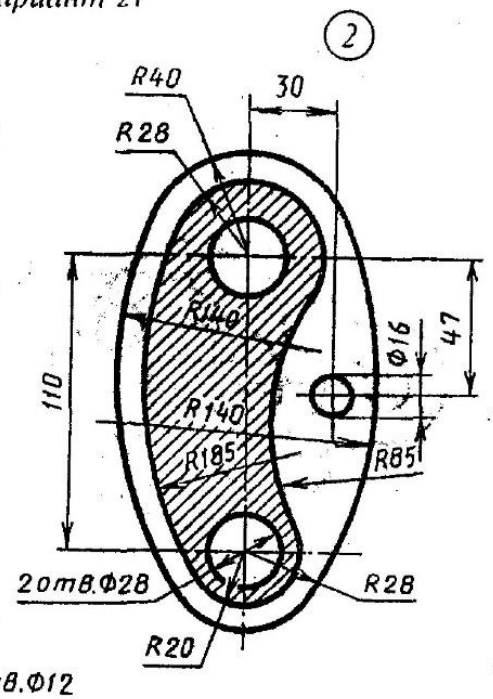

Вариант 17
1
2
Ф6
4отв
Ф42
Ф22
Ф60
R100
R98
6
R3
R33
Ф20
4отв
30°
Вилка
Ф14
2отв
R6
R14
R5
Ф36
28
10
R20
66
32
R34
R7
R14
Ф40
30°
Воронка
Вариант 18
1
4отв Ф10
R24
R7
R24
42
R63
Ф80
R75
R12
30°
Крышка
2
76
33
66
R6
R6
R10
Ф12
4отв
18
R7
R21
Ф34
R38
R30
45°
Ф110
Корпус


Вариант 19
1
46
Ф22
2 отв.
R8
R100
R8
Ф9
R15
Ф80
Ф46
Ф62
Ф10
6 отв
R18
68
Кронштейн
2
88
2 отв.Ф12
R34
R12
R40
40
Ф22
120
6
R10
R14
R4
15
80
100
Стойка
Вариант 20
1
R8
Ф40
Ф62
6 отв.Ф8
R18
65
Ф30
3 отв.Ф16
R100
110
Крышка
2
R10
Ф10
R8
Ф20
32
R18
R32
60
R18
R9
R48
R30
Корпус

Вариант 21

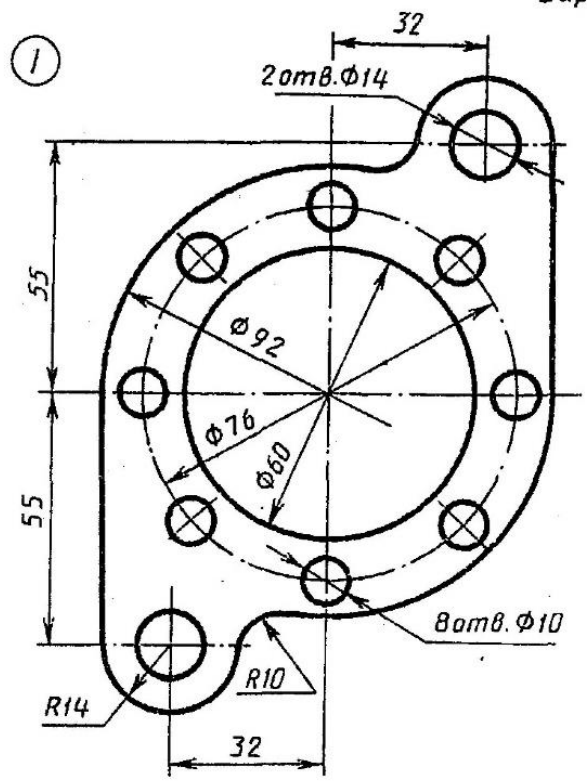



Крышка

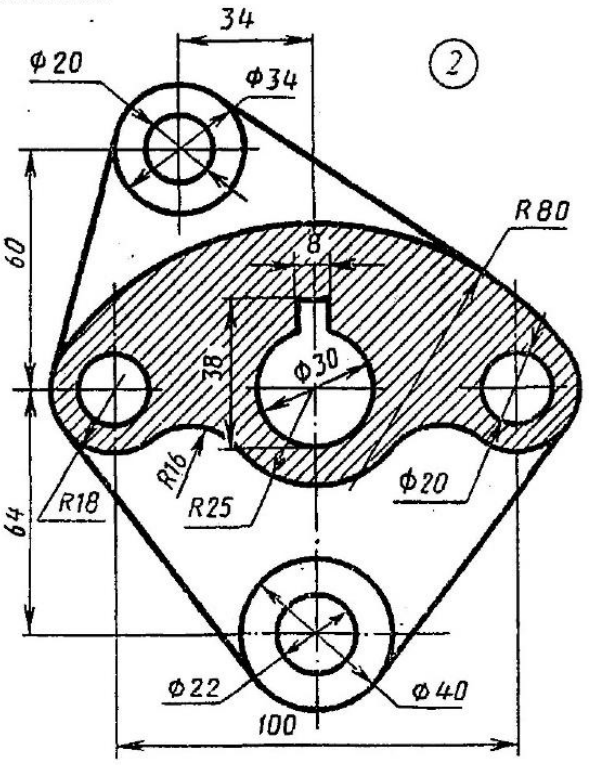



Коромысло

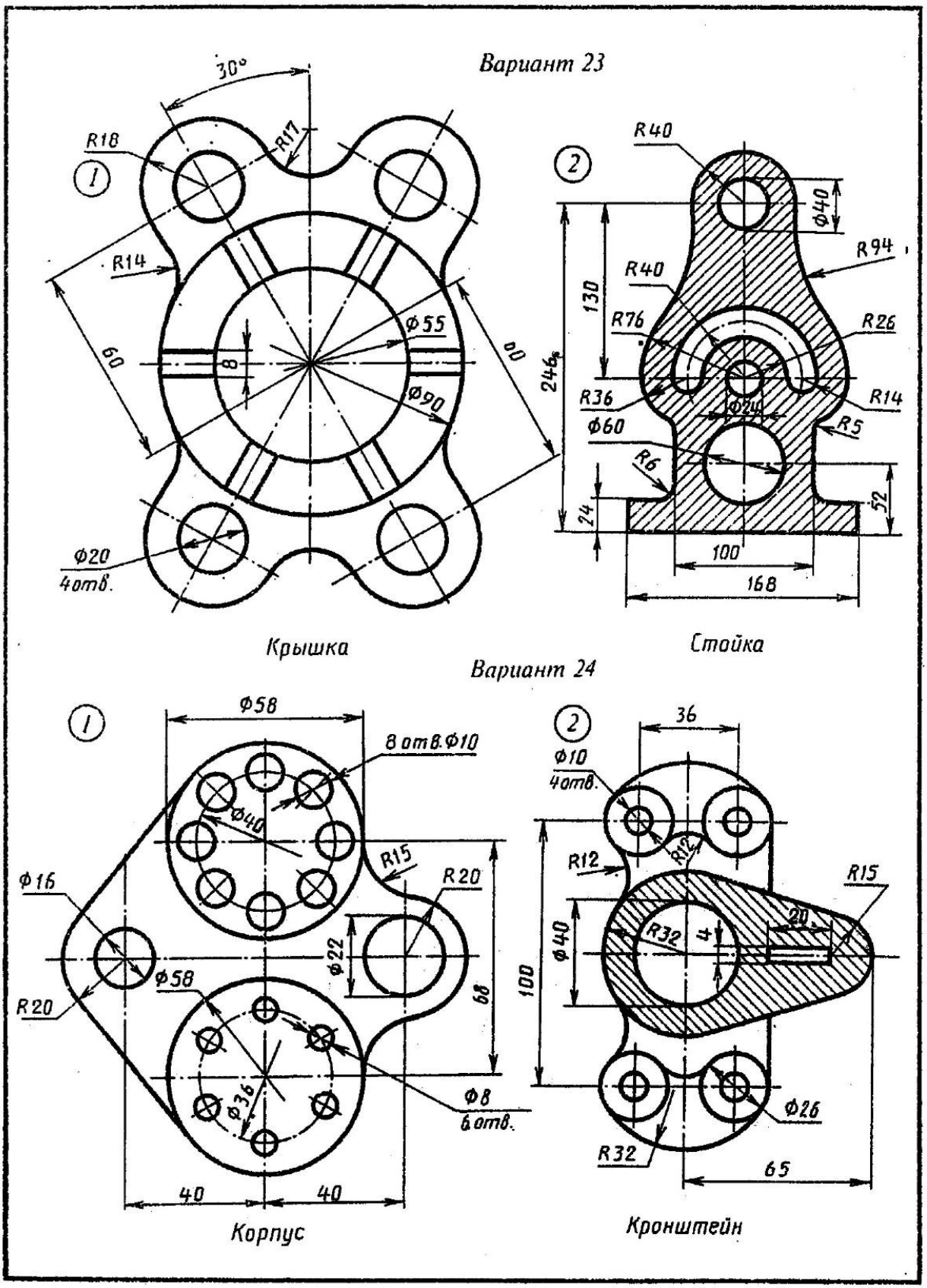
Вариант 22

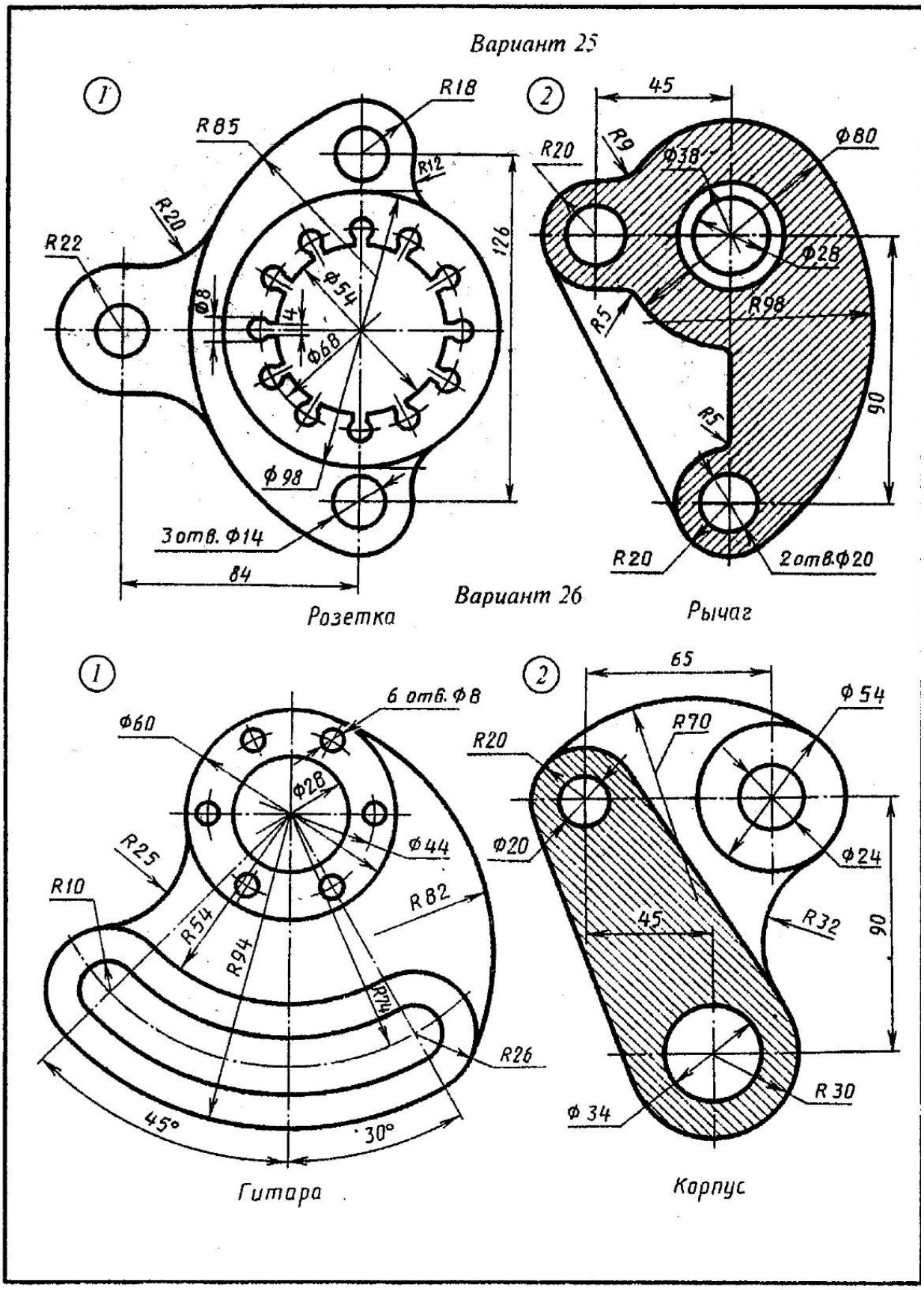


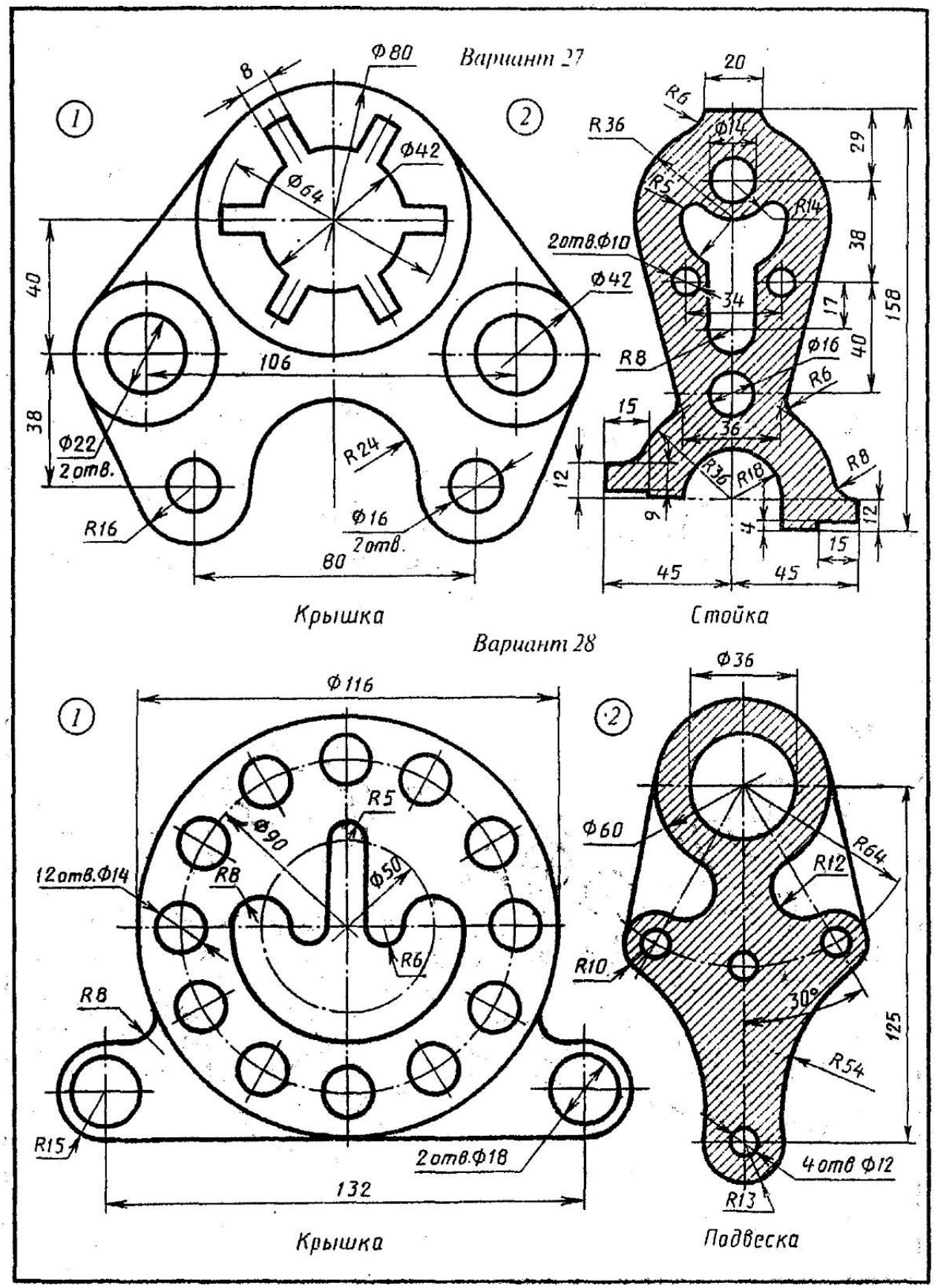

Прокладка




Рычаг


Вариант 23
30°
R18
R17
1
R14
60
8
Ф55
Ф90
00
Ф20
4отв.
2
R40
Ф40
R94
130
R40
R76
R26
246
R36
Ф24
R14
Ф60
R5
R6
52
24
100
168
Крышка
Стойка
Вариант 24
1
Ф58
8 отв.Ф10
Ф40
R15
R20
Ф16
Ф22
68
Ф58
R20
Ф36
Ф8
6 отв.
40
40
2
36
Ф10
4отв.
R12
R12
R15
R32
4
20
Ф40
100
Ф26
R32
65
Корпус
Кронштейн


Вариант 25
1
R18
R85
R12
R20
R22
126
Ф54
Ф8
4
Ф68
Ф98
3 отв. Ф14
84
Розетка
2
45
R9
Ф80
R20
Ф38
Ф28
R98
R5
90
R5
R20
2 отв. Ф20
Рычаг
Вариант 26
1
6 отв. Ф8
Ф60
Ф28
Ф44
R25
R10
R82
R54
R94
R74
R26
45°
30°
Гитара
2
65
Ф54
R70
R20
Ф20
Ф24
45
R32
90
R30
Ф34
Корпус


Вариант 27
1
2
Ф80
8
Ф42
Ф64
40
106
38
Ф22
2 отв.
R24
R16
Ф16
2 отв.
80
Крышка
20
R6
R36
Ф14
29
R5
R14
38
2 отв. Ф10
Ф42
34
17
158
40
Ф16
R8
R6
15
36
R36
R18
R8
12
9
4
12
15
45
45
Стойка
Вариант 28
1
Ф116
R5
Ф90
12 отв. Ф14
R8
Ф50
R6
R8
R15
2 отв. Ф18
132
Крышка
2
Ф36
Ф60
R64
R12
R10
30°
125
R54
4 отв Ф12
R13
Подвеска

## Приложение В
## (обязательное)

## Задания для работы с моделью графического документа

**Задание В.1.**

Написать библиотеку, которая будет создавать 4 новых вида функцией `CreateSheetView` и рисовать в нем несколько геометрических элементов. Выполнить разнесение видов на листе чертежа, так, чтобы их точки привязки были равномерно распределены по направлению, указанному в строке «Разнесение видов» таблицы В.1. В соответствии со своим вариантом, библиотека должна выполнять операции, номера которых приведены в строке «Номера операций» таблицы 1.

Таблица В.1 – Варианты заданий для операций, соответствующих таблице В.2

| **Вариант** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
|---|---|---|---|---|---|---|---|---|---|---|
| **Номера операций** | 123 | 456 | 712 | 345 | 671 | 234 | 567 | 124 | 457 | 713 |
| **Разнесение видов** |  |  |  |  |  |  |  |  |  |  |
| **Вариант** | **11** | **12** | **13** | **14** | **15** | **16** | **17** | **18** | **19** | **20** |
| **Номера операций** | 346 | 672 | 235 | 561 | 134 | 467 | 356 | 612 | 245 | 571 |
| **Разнесение видов** |  |  |  |  |  |  |  |  |  |  |

Таблица В.2 – Функции для работы с видами

| № | **Имя функции** | **Назначение** |
|---|---|---|
| 0 | CreateSheetView | Создать новый вид в чертеже |
| 1 | GetViewReference | Получить указатель на вид по номеру вида |
| 2 | GetViewNumber | Получить номер вида по указателю на вид или объект вида |
| 3 | OpenView | Сделать текущим существующий вид с указанным номером |
| 4 | NewViewNumber | Определить номер следующего вида (т.к. при создании нового вида требуется указывать его номер) |
| 5 | GetViewObjCount | Получить количество объектов вида |
| 6 | SheetToView | Пересчитать координаты точки из СК листа в СК текущего вида |
| 7 | ViewToSheet | Пересчитать координаты точки из СК текущего вида в СК листа |

Так, например, для 10 варианта должны быть реализованы функции: `ViewToSheet, GetViewReference, OpenView`.

В каждом виде должен содержаться графический элемент, входящий в именованную группу.

**Задание В.2.**

Создать итератор, перебирающий объекты в соответствии с вариантом, приведенным в таблице В.3. Для каждого объекта вывести на экран его параметры (не обязательно все) после чего изменить объект.

Таблица В.3 – Варианты типов объектов

| **Вариант** | **Объект** | **Вариант** | **Объект** |
|---|---|---|---|
| 1 | Отрезок | 11 | Прямоугольник |
| 2 | Окружность | 12 | Эквидистанта |
| 3 | Дуга | 13 | Многоугольник |
| 4 | Точка | 14 | Обозначение центра |
| 5 | Штриховка | 15 | Вид |
| 6 | Кривая Безье, сплайн | 16 | Слой |
| 7 | Шероховатость | 17 | Осевая линия |
| 8 | Полилиния | 18 | Технические требования (TechnicalDemandParam) |
| 9 | Эллипс | 19 | Допуск формы |
| 10 | Дуга эллипса | 20 | База |

## Приложение Г
## (обязательное)

## Задания для интерактивного взаимодействия с КОМПАС

**Задание В.1.**

Написать библиотеку в соответствии с вариантом, приведенным в таблице Г.1, которая будет формировать меню для размещения графических объектов в окне КОМПАС.

Таблица Г.1 – Варианты заданий для функций, соответствующих таблице Г.2

| **Вариант** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
|---|---|---|---|---|---|---|---|---|---|---|
| **Номера функций** | 11 8<br>14 1 | 2 4<br>9 12 | 10 3<br>13 8 | 4 7<br>9 10 | 5 14<br>2 7 | 6 13<br>1 11 | 7 4<br>12 9 | 8 3<br>13 5 | 9 6<br>12 7 | 10 8<br>2 14 |
| **Вариант** | **11** | **12** | **13** | **14** | **15** | **16** | **17** | **18** | **19** | **20** |
| **Номера функций** | 11 4<br>10 1 | 12 2<br>14 8 | 13 1<br>10 9 | 14 1<br>10 5 | 14 7<br>9 12 | 13 8<br>5 10 | 12 2<br>10 7 | 5 7<br>2 14 | 11 3<br>6 13 | 10 3<br>9 4 |

Таблица Г.2 – Используемые функции

| **Номер** | **Функция** | **Номер** | **Функция** |
|---|---|---|---|
| 1 | ksColouring | 8 | ksRegularPolygon |
| 2 | ksConicArc | 9 | Bezier |
| 3 | ArcByAngle | 10 | Nurbs |
| 4 | ksEllipse | 11 | NurbsForConicCurve |
| 5 | ksEllipseArc | 12 | Text |
| 6 | ksParEllipseArc | 13 | Hatch |
| 7 | ksRectangle | 14 | ksHatch |

Так, например, для 10 варианта должно быть создано меню для выбора построения объектов с применением функций: `Nurbs, ksRegularPolygon, ksConicArc, ksHatch`.