Объектно-ориентированное программирование

Лабораторная работа №2

Структуры и перечисления

2 Задания к лабораторным работам

2.2 Лабораторная работа #2 Структуры и перечисления

2.2.1 Обработка исключений

2.2.1.1 Далее показана функция `Sort()`, выполняющая сортировку вещественных чисел, а также функция `DemoSort()`, показывающая пример вызова функции сортировки.

Перепишите данный код в среду разработку. Далее добавьте в функцию сортировки проверку – если значение `count` меньше нуля, то функция должна выбросить исключение типа `exception`. Для работы механизма исключений не забудьте подключить стандартную библиотеку `<exception>` и пространство имен `std`.

В функции `DemoSort()` дважды вызовите функцию `Sort()`, сначала для `count` равного 5, а затем равным -1. Убедитесь, что при положительных значениях `count` сортировка выполняется корректно, а при отрицательных генерируется исключение.

```
// Сортирует массив вещественных чисел
void Sort(double* values, int count)
{
   double swap;
   for (int i = 0; i < count; i++)
   {
      for (int j = 0; j < count; j++)
      {
         if (values[i] < values[j])
         {
            swap = values[i];
            values[i] = values[j];
            values[j] = swap;
         }
      }
   }
}

// Демонстрирует работу сортировки
void DemoSort()
{
```

```
    int count = 5;
    double* values = new double[count] {100.0, 249.0, 12.0, 45.0, 23.5};

    Sort(values, count);
}
```

2.2.1.2 Добавьте в функцию `DemoSort()` обработку созданного исключения. Для этого поместите вызов функции `Sort()` внутри блока `try{}`, а в блоке `catch{}` добавьте вывод на экран сообщения `«Exception catched!»`.

Убедитесь, что сообщение `«Exception catched»` будет выводиться только при вызове функции `Sort()` с отрицательным значением `count`, и не будет выводиться при передаче корректных данных.

> **Запомните: механизм обработки исключений нужен для того, чтобы одна функция могла сообщить другой функции об ошибке в ходе своей работы. Поэтому генерация исключения (`throw`) выполняется в используемой функции, а обработка (`try` и `catch`) во внешней функции.**

2.2.2 Создание структур

2.2.2.1 Ниже приведены структуры, описанные на формальном языке. Переведите ниже описанные структуры с формального языка на язык Си++. Убедитесь, что программа компилируется.

```
// Структура Прямоугольник
// Начало описания структуры
// Вещественное поле Длина
// Вещественное поле Ширина
// Строковое поле Цвет
// Конец описания структуры
```

```
// Структура Рейс
// Начало описания структуры
// Строковое поле Пункт Вылета
// Строковое поле Пункт назначения
// Целочисленное поле Время полета в минутах
// Конец описания структуры
```

```
// Структура Фильм
// Начало описания структуры
// Строковое поле Название
// Целочисленное поле Продолжительность в минутах
// Целочисленное поле Год выпуска
// Строковое поле Жанр
// Вещественное поле Рейтинг
// Конец описания структуры
```

```
// Структура Время
// Начало описания структуры
// Целочисленное поле Часы (от 0 до 23)
// Целочисленное поле Минуты (от 0 до 60)
// Целочисленное поле Секунды (от 0 до 60)
// Конец описания структуры
```

2.2.2.2 Для ниже перечисленных структур придумайте поля (3-4 поля), определите наиболее подходящие типы данных для полей, а затем напишите код этих структур на языке Си++. Убедитесь, что программа компилируется.

- Контакт (в телефонной книжке на смартфоне)
- Песня (в плеере)
- Дисциплина (как запись в зачетной книжке)

2.2.3 Создание объектов структур

2.2.3.1 Для каждой структуры из задания 2.2.2.1 создайте следующие функции:

```
void DemoRectangle() {…}
void DemoFlight() {…}
void DemoMovie() {…}
void DemoTime() {…}
```

В теле каждой функции создайте переменную соответствующей структуры и присвойте в их поля значения. Созданные функции вызовите из функции main(). Убедитесь, что программа компилируется.

2.2.3.2 В каждой функции, написанной в предыдущей задаче, создайте вторую переменную соответствующего перечисления. Далее напишите код, который будет запрашивать значения полей структуры у пользователя и помещать их в поля объекта. После ввода пользователем всех полей, функция должна вывести на экран их значения, как показано ниже (на примере структуры Прямоугольник):

```
Введите длину прямоугольника (положительное число):
> 15
Введите ширину прямоугольника (положительное число):
> 20
Прямоугольник имеет размеры: 15x20
```

2.2.3.3 В каждой функции, написанной в предыдущей задаче, создайте массив из 3-5 объектов соответствующей структуры (например, массив из трёх треугольников, массив из пяти рейсов и т.д.). Присвойте полям всех элементов массива какие-нибудь значения. Затем напишите цикл, который выводит на экран значения из полей. Пример вывода на экран для структуры Рейс:

```
Рейс 1 Москва - Санкт-Петербург находится в полете 45 минут
Рейс 2 Берлин - Лондон находится в полете 105 минут
Рейс 3 Москва - Барселона находится в полете 180 минут
```

2.2.4 Работа с объектами структур через указатели

2.2.4.1 В функции DemoRectangle(), написанной в предыдущих задачах, создайте указатель на структуру Прямоугольник. В указатель присвойте адрес первой переменной Прямоугольника, созданной в задаче 2.2.3.1.

Обращаясь к полям объекта через указатель (оператор `->`), выведите их значения на экран. Затем, также обращаясь к полям через указатель, присвойте полям новые значения.

2.2.4.2 Добавьте в функцию DemoRectangle() второй указатель, и поместите в него адрес того же объекта (адрес первой переменной). Выведите на экран адреса, которые хранятся в обоих указателях. Они должны совпадать.

2.2.4.3 Сделайте аналогичную работу с указателями для других структур (Рейс, Время, Фильм) в соответствующих им функциях.

2.2.4.4 Напишите функцию void WrongPointers().

Внутри функции создайте две переменных типа Рейс и Фильм. Затем создайте два указателя на типы Рейс и Фильм. Присвойте в указатель на Рейс адрес переменной Рейса, а в указатель на Фильм адрес переменной Фильма. Убедитесь, что программа компилируется.

Затем попробуйте присвоить в указатель на Рейс адрес переменной Фильм, а указатель на Фильм адрес переменной Рейс. Убедитесь, что при попытке компиляции компилятор сообщает об ошибке.

> **Также, как в указатель на double нельзя поместить адрес переменной типа int, также в указатель одной структуры нельзя поместить адрес объекта другой структуры. В противном случае, операция разыменования такого указателя выполнялась бы неправильно.**

2.2.5 Структуры и функции

Структуры можно передавать и возвращать из функций также, как и обычные типы данных, такие как int или double. Однако передавать их можно только по ссылке или по указателю. С чем связано данное ограничение и как его разрешить, будет рассмотрено позже.

2.2.5.1 Напишите функцию `void WriteRectangle(Rectangle& rectangle)`, которая принимает по ссылке объект типа rectangle, и выводит значения его полей на экран. Пример вывода на экран:

```
Прямоугольник имеет размер 15x20
```

2.2.5.2 Напишите функцию `void ReadRectangle(Rectangle& rectangle)`, которая принимает по ссылке объект типа Rectangle, и считывает значения его полей с клавиатуры. Пример считывания с клавиатуры:

```
Введите длину прямоугольника: 15
Введите ширину прямоугольника: 20
```

2.2.5.3 Напишите функцию `void DemoReadAndWriteRectangles()`, в которой создайте массив из пяти элементов типа Rectangle. Напишите цикл, который считывает значения всех пяти элементов массива с клавиатуры с помощью ранее написанной функции ReadRectangle(). Затем напишите цикл, который выводит значения всех элементов массива на экран с помощью ранее написанной функции WriteRectangle(). Вызовите функцию DemoReadAndWriteRectangles() в функции main() и убедитесь, что программа работает корректно

2.2.5.4 Напишите функцию `void Exchange(Rectangle& rectangle1, Rectangle& rectangle2)`, которая принимает по ссылке два объекта типа Rectangle, и переставляет значения их полей между собой (значения полей из rectangle1 присваиваются в поля rectangle2, и наоборот). Вызовите функцию Exchange() в функции DemoRectangle(), и убедитесь, что функция переставляет местами поля из объектов. Для этого выведите на экран значения переменных до вызова функции Exchange и после.

2.2.5.5 Напишите функцию `void FindRectangle(Rectangle* rectangles, int count)`, которая принимает указатель на массив прямоугольников и их количество в массиве. Функция должна перебрать все прямоугольники в массиве и найти прямоугольник с самой большой длиной. Пример работы функции:

```
Прямоугольник с максимальной длиной имеет размер 120x15
```

2.2.5.6 Напишите функцию `void FindMaxRectangle(Rectangle* rectangles, int count)`, которая принимает указатель на массив прямоугольников и их количество в массиве. В отличие от функции из предыдущего задания, функция FindMaxRectangle() должна найти и вывести на экран прямоугольник с наибольшей площадью. Для определения площади внутри цикла перемножайте значения полей длины и ширины каждого прямоугольника. Пример работы функции:

```
Прямоугольник с максимальной площадью имеет размер 60x60
```

2.2.6 Структуры и динамическая память

2.2.6.1 Напишите функцию void DemoDynamicFlight(). Внутри функции объявите указатель на тип структуры Рейс. Создайте динамически (через оператор new) объект структуры Рейс и поместите его в указатель.

Через указатель присвойте в поля динамического объекта значения (для обращения к полям объекта через указатель необходимо использовать оператор `->`). Значения можно задать из программного кода.

Выведите значения полей объекта на экран. После вывода на экран освободите память объекта, используя оператор delete. Пример вывода на экран:

```
Рейс Москва - Санкт-Петербург находится в полете 45 минут
```

2.2.6.2 Напишите функцию void DemoDynamicFlights() по аналогии с функцией DemoDynamicFlight() из предыдущего задания. В данной функции вместо создания одного объекта структуры Рейс, создайте массив из 4 объектов Рейса. Проинициализируйте объекты массива – присвойте полям каждого объекта Рейса значения.

Выведите данные из массива на экран, а затем освободите память из-под массива. Пример вывода на экран:

```
Рейс 1 Москва - Санкт-Петербург находится в полете 45 минут
Рейс 2 Томск - Москва находится в полете 190 минут
Рейс 3 Берлин - Лондон находится в полете 105 минут
Рейс 4 Москва - Барселона находится в полете 180 минут
```

2.2.6.3 Напишите функцию void FindShortestFlight(Flight* flights, int count), которая принимает на вход указатель на массив flights и его размер. Функция должна найти рейс с самым меньшим временем перелета, и вывести информацию об этом рейсе на экран.

Добавьте вызов функции FindShortestFlight() в функцию DemoDynamicFlights() из предыдущего задания, передав в функцию проинициализированный массив из 4 объектов. Вызовите функцию DemoDynamicFlights() из функции main(), и убедитесь, что все функции работают правильно. Пример работы программы:

```
Рейс 1 Томск - Москва находится в полете 190 минут
Рейс 2 Москва - Санкт-Петербург находится в полете 45 минут
Рейс 3 Берлин - Лондон находится в полете 105 минут
Рейс 4 Москва - Барселона находится в полете 180 минут

Самый короткий рейс Москва - Санкт-Петербург находится в полете 45 минут
```

2.2.7 Функции-конструкторы

Посмотрите на пример кода ниже, в котором создаются три объекта структуры.

```
struct Movie
{
   string Title;   // название фильма
   string Genre;   // жанр фильма
   int Year;       // год выпуска
   double Rate;    // рейтинг фильма от 0 до 10
};

void DemoMovie()
{
   Movie* movie1 = new Movie();
   movie1->Title = "Крепкий орешек";
   movie1->Genre = "Боевик";
   movie1->Year = 1988;
   movie1->Rate = 8.0;

   Movie* movie2 = new Movie();
   movie2->Title = "Побег из Шоушенка";
   movie2->Genre = "Драма";
   movie2->Year = 1994;
   movie2->Rate = 9.1;

   Movie* movie3 = new Movie();
   movie3->Title = "1+1";
```

```
    movie3->Genre = "Комедия, драма";
    movie3->Year = 2012;
    movie3->Rate = 8.8;
    ...
}
```

Инициализация каждого поля в отдельной строке выглядит громоздко, поэтому при объявлении нескольких объектов приходится писать много кода. Можно избавиться от дублирования, написав специальные ***функции-конструкторы***. Функциями-конструкторами будем называть такие функции, которые: а) на вход принимают значения для всех полей структуры; б) в теле функции создают объект структуры и присваивают значения всем его полям; в) возвращают указатель на созданный объект. Объект обязательно создается динамически. Далее представлен пример функции-конструктора:

```
Movie* MakeMovie(string& title, string& genre, int year, double rate)
{
    Movie* movie = new Movie();
    movie->Title = title;
    movie->Genre = genre;
    movie->Year = year;
    movie->Rate = rate;
    return movie;
}
```

Используя функцию-конструктор, код функция DemoMovie() станет гораздо короче:

```
void DemoMovie()
{
    Movie* movie1 = MakeMovie("Крепкий орешек", "Боевик", 1988, 8.0);
    Movie* movie2 = MakeMovie("Побег из Шоушенка", "Драма", 1994, 9.1);
    Movie* movie3 = MakeMovie("1+1", "Комедия, драма", 2012, 8.8);
    ...
}
```

Обратите внимание, как упростилось создание объектов – теперь объекты можно создавать одной строкой вместо инициализации отдельных полей. Чтобы отличать функции-конструкторы от других функций и быстро находить их через автодополнение в среде разработке, используйте слово Make и имя структуры в имени функции. Например, MakeMovie(), MakeRectangle(), MakeSong() и т.д.

Другой пример связан с копированием объектов. Часто, при разработке приложений необходимо сделать копию некоторого объекта. Например, нам необходимо сделать новый объект структуры Movie, где значения всех полей будут скопированы из другого объекта:

```
void DemoMovie()
{
   Movie* movie1 = MakeMovie("Крепкий орешек", "Боевик", 1988, 8.0);
   Movie* movie2 = MakeMovie("Побег из Шоушенка", "Драма", 1994, 9.1);
   Movie* movie3 = MakeMovie("1+1", "Комедия, драма", 2012, 8.8);

   // Создаём первую копию
   Movie* copiedMovie1 = new Movie();
   copiedMovie1->Title = movie1->Title;
   copiedMovie1->Genre = movie1->Genre;
   copiedMovie1->Year = movie1->Year;
   copiedMovie1->Rate = movie1->Rate;

   // Создаём вторую копию
   Movie* copiedMovie2 = new Movie();
   copiedMovie2->Title = movie2->Title;
   copiedMovie2->Genre = movie2->Genre;
   copiedMovie2->Year = movie2->Year;
   copiedMovie2->Rate = movie2->Rate;

   // Создаём вторую копию
   Movie* copiedMovie3 = new Movie();
   copiedMovie3->Title = movie3->Title;
   copiedMovie3->Genre = movie3->Genre;
   copiedMovie3->Year = movie3->Year;
   copiedMovie3->Rate = movie3->Rate;
}
```

Очевидное дублирование кода можно исключить, написав ***функцию копирования***:

```
Movie* CopyMovie(Movie& movie)
{
   Movie* copiedMovie = new Movie();
   copiedMovie->Title = movie.Title;
   copiedMovie->Genre = movie.Genre;
   copiedMovie->Year = movie.Year;
   copiedMovie->Rate = movie.Rate;
   return copiedMovie;
}
```

Функция копирования принимает по ссылке копируемый объект, создает новый динамический объект, затем последовательно копирует все поля исходного объекта и возвращает новый объект из функции. При использовании функции копирования код функции DemoMovie() становится кратким, читаемым и без дублирования:

```
void DemoMovie()
{
   Movie* movie1 = MakeMovie("Крепкий орешек", "Боевик", 1988, 8.0);
   Movie* movie2 = MakeMovie("Побег из Шоушенка", "Драма", 1994, 9.1);
   Movie* movie3 = MakeMovie("1+1", "Комедия, драма", 2012, 8.8);

   // Создаём первую копию
   Movie* copiedMovie1 = CopyMovie(*movie1);
   Movie* copiedMovie2 = CopyMovie(*movie2);
   Movie* copiedMovie3 = CopyMovie(*movie3);
}
```

Чтобы отличать функции копирования от других функций и быстро находить их через автодополнение в среде разработке, используйте слово Copy и имя структуры в имени функции. Например, CopyMovie(), CopyRectangle(), CopySong() и т.д.

> **Функцию копирования можно написать более лаконично, если вместо прямого копирования всех полей вызвать функцию-конструктор с передачей значений полей**

Функции-конструкторы и функции копирования позволяют избавиться от дублирования и писать меньше кода при разработке программ. В следующих лабораторных мы рассмотрим более правильный подход к созданию объектов структуры, основанный на функциях-конструкторах.

2.2.7.1 Для следующей структуры Circle создайте функцию-конструктор MakeCircle() и функцию копирования CopyCircle(). С помощью созданных функций избавьтесь от дублирования кода в функции DemoCircle():

```
struct Circle           // Структура круг
{
   double X;            // X-координата центра круга
   double Y;            // Y-координата центра круга
   double Radius;       // Радиус
   string Color;        // Цвет
};

void DemoCircle()
{
   Circle* circle1 = new Circle();
   circle1->X = 5.0;
   circle1->Y = 7.0;
   circle1->Radius = 7.5;
   circle1->Color = "Red";

   Circle* circle2 = new Circle();
   circle2->X = 2.0;
```

```
    circle2->Y = -12.0;
    circle2->Radius = 12.75;
    circle2->Color = "Green";

    Circle* circle3 = new Circle();
    circle3->X = -10.0;
    circle3->Y = 10.0;
    circle3->Radius = 1.45;
    circle3->Color = "Blue";

    Circle* copiedCircle1 = new Circle();
    copiedCircle1->X = circle1->X;
    copiedCircle1->Y = circle1->Y;
    copiedCircle1->Radius = circle1->Radius;
    copiedCircle1->Color = circle1->Color;

    Circle* copiedCircle2 = new Circle();
    copiedCircle2->X = circle2->X;
    copiedCircle2->Y = circle2->Y;
    copiedCircle2->Radius = circle2->Radius;
    copiedCircle2->Color = circle2->Color;

    Circle* copiedCircle3 = new Circle();
    copiedCircle3->X = circle3->X;
    copiedCircle3->Y = circle3->Y;
    copiedCircle3->Radius = circle3->Radius;
    copiedCircle3->Color = circle3->Color;
}
```

2.2.7.2 Напишите функции-конструкторы для структур Прямоугольник, Рейс, Фильм, Время, написанных ранее в задании 2.1.2.1. Вызовите конструкторы в функции main() и убедитесь, что они работают корректно.

2.2.7.3 Создайте функции копирования для структур Прямоугольник, Рейс, Фильм, Время, написанных ранее в задании 2.1.2.1. Вызовите функции копирования в функции main() и убедитесь, что они работают корректно.

2.2.8 Перечисления

2.2.8.1 Переведите с формального языка на язык Си++ следующие перечисления:

```
// Перечисление Цвет
// Начало перечисления
// Красный,
// Оранжевый,
// Желтый,
// Зеленый,
// Голубой,
// Синий,
// Фиолетовый
// Конец перечисления
```

```
// Перечисление День недели
// Начало перечисления
// Понедельник,
// Вторник,
// Среда,
// Четверг,
// Пятница,
// Суббота,
// Воскресенье
// Конец перечисления
```

```
// Перечисление Жанр
// Начало перечисления
// Комедия,
// Драма,
// Триллер,
// Боевик,
// Ужасы,
// Блокбастер
// Конец перечисления
```

2.2.8.2 Создайте следующие перечисления на Си++:

- Форма обучения студента (очное, заочное, вечернее, дистанционное)
- Производители смартфонов (использовать названия компаний)
- Время года

2.2.8.3 Создайте функцию DemoEnums(). Внутри функции создайте переменную типа перечисления Цвет и присвойте ей значение. Затем создайте переменные других типов перечислений, созданных вами в задачах 2.1.8.1, 2.1.8.2.

2.2.8.4 Внутри функции DemoEnums() создайте массив из 6 значений перечисления Цвет и проинициализируйте его из программного кода. Затем создайте массивы других типов перечислений, созданных вами в задачах 2.1.8.1, 2.1.8.2.

2.2.8.5 Напишите функцию `void WriteColor(Color color)`, которая принимает на вход значение типа перечисления Цвет и выводит его на экран. Пример вывода на экран:

```
Красный цвет
Синий цвет
Желтый цвет
```

Примечание: при попытке вывода значения перечисления через cout (`cout << color`) в консоль выведется целое число, соответствующее значению перечисления. Чтобы вывести на экран текстовое название цвета, в Си++ необходимо написать конструкцию switch-case, где для каждого варианта цвета будет выводится соответствующая ему строка:

```
switch (color)
{
   case Red:
      cout << "Красный цвет" << endl;
      break;
   case Yellow:
      cout << "Красный цвет" << endl;
      break;
   ... // аналогично для других цветов
}
```

В других языках программирования, таких как C#, работа с перечислениями организована гораздо удобнее.

2.2.8.6 Напишите функцию `Color ReadColor()`, которая считывает с клавиатуры целое число и возвращает ему соответствующее значение цвета. Пример работы функции:

```
Введите число от 0 до 6 (0 – красный, 1 – оранжевый, 2 – желтый, 3 –
зеленый, 4 – голубой, 5 – синий, 6 – фиолетовый):
> 1
```

Примечание: при считывании значений перечислений также приходится делать искусственный приём – считывать с клавиатуры целое число, а затем с помощью конструкции switch-case (или явного приведения типов) преобразовывать число в значение перечисления. Для некоторых перечислений используют считывание с клавиатуры букв вместо цифр. Например, y/n для считывания перечисления Yes/No, m/f для считывания пола человека (Male / Female) и т.д. Использование букв вместо цифр в некоторых случаях комфортнее воспринимается конечным пользователем программы.

2.2.8.7 Напишите функцию int CountRed(Color* colors, int count), которая принимает на вход массив переменных типа Цвет и их количество. Внутри функция перебирает весь массив и подсчитывает сколько раз в массиве встречается красный цвет. Полученное число возвращается из функции с помощью оператора return.

Например:

для входного массива `{Красный, Синий, Желтый, Красный, Желтый, Зеленый}` функция должна вернуть значение 2.

2.2.8.8 На основе функции CountRed() из предыдущей задачи напишите функцию int CountColor(Color* colors, int count, Color findedColor), которая подсчитывает сколько раз определенный цвет встречается в массиве. Для этого в функцию передается дополнительный аргумент findedColor, который указывает, какой цвет нужно искать. Вызовите функцию CountColor() в функции main() и убедитесь, что она правильно считает цвета в массиве.

Например:

для входного массива цветов `{Красный, Синий, Желтый, Красный, Желтый, Зеленый}` и findedColor равного Красному, функция должна вернуть значение 2. Если findedColor равен Синему, то функция должна вернуть 1, а если findedColor равен Фиолетовому, то 0.

2.2.9 Использование перечислений в структурах

Перечисления редко используются сами по себе. Как правило, перечисления создаются, чтобы использовать их в качестве полей структур. Например, ранее была написана структура Circle со строковым полем Color, хранящим цвет. Проблема хранения некоторых видов данных в виде строк заключается в том, что для конечного пользователя значения строк "Red" и "red" не отличаются, так как имеют один и тот же смысл. Однако для компьютера это две разные строки, в которой символы отличаются регистром. Как результат, пользователь может начать вводить цвет круга в программе в разных написаниях. Однако, если в дальнейшем, ему понадобится, например, найти все круги красного цвета, то программа

может выдать неправильный результат, так как будет искать значение цвета "Red", исключая из поиска значение "red" – что будет ошибочным поведением программы для пользователя.

С этой точки зрения, перечисления позволяют жестко ограничить область допустимых значений для поля, исключая подобные варианты неправильного ввода данных.

Следующий ряд задач направлен на то, чтобы научить вас создавать поля типа перечислений, а также работать с ними в структурах.

2.2.9.1 Замените в структуре Фильм строковое поле Жанр на поле типа перечисления Жанр.

2.2.9.2 Напишите функцию void DemoMovieWithGenre(). Внутри функции создайте переменную типа Фильм и присвойте значения во все поля, в том числе в поле Жанр.

2.2.9.3 Исправьте функцию-конструктор MakeMovie() для структуры Фильм: замените ранее передаваемую строку "Жанр" на значение типа перечисления. Вызовите исправленную версию функции MakeMovie() в функции DemoMovieWithGenre() для создания еще одной переменной типа Фильм. Не забудьте освободить память в конце функции DemoMovieWithGenre() для всех динамически созданных объектов.

Примечание: изменение типа поля в уже используемой структуре Фильм приведет к необходимости исправления и других функций, работающих с Фильмами. Исправьте эти функции. Очевидно, что изменение типов данных полей может привести к большому количеству правок исходного кода. Таким образом, при создании структур надо тщательно обдумывать типы данных для полей, чтобы сократить количество исправлений в последствии.

2.2.9.4 Напишите функцию int CountMoviesByGenre(Movie* movies, int count, Genre findGenre), которая принимает на вход массив фильмов и подсчитывает сколько среди них фильмов определенного жанра. Значение аргумента findedGenre задает искомый жанр.

В функции DemoMovieWithGenre() создайте динамический массив из десяти фильмов разных жанров. Вызовите функцию CountMoviesByGenre() для созданного массива и выведите количество найденных фильмов на экран. Запустите программу и убедитесь, что функция работает правильно.

2.2.9.5 Напишите функцию Movie* FindBestGenreMovie(Movie* movies, int count, Genre findGenre), которая принимает на вход массив фильмов, и среди всех фильмов находит

фильм заданного жанра с наиболее высоким рейтингом. То есть, если в качестве значения аргумента findGenre будет значение Комедии, тогда функция должна перебрать все фильмы массива, искать среди них комедии, и запоминать комедию с наибольшим рейтингом. По завершению поиска функция должна вернуть указатель на найденный фильм. Если среди элементов массива нет фильма заданного жанра, программа должна вернуть nullptr.

Теоретические вопросы

1. Что такое исключение?
2. Что такое структура?
3. Что такое перечисление?
4. Что такое абстрагирование?
5. Что такое состояние структуры?
6. Что такое функция-конструктор?
7. Что такое функция копирования?
8. Какие преимущества может дать использование перечислений в качестве полей по сравнению со строками?

Литература

1. **Лафоре Р. "Объектно-ориентированное программирование в C++".**
   Глава 4 "Структуры", Глава 14 "Шаблоны и исключения"