**ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ**
Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования
**«Московский технический университет связи и информатики»**
Кафедра «Информационная безопасность»

Лабораторный практикум
по дисциплине

# ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ СИСТЕМ ЗАЩИТЫ ИНФОРМАЦИИ

(для студентов направлений подготовки 09.03.01, 10.03.01, 11.03.02)

Москва 2017

План УМД на 2017/18 уч. г.

Лабораторный практикум
по дисциплине
ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ СИСТЕМ ЗАЩИТЫ ИНФОРМАЦИИ
(для студентов направлений подготовки 09.03.01, 10.03.01, 11.03.02)

Составители: Симонян А.Г., к.т.н., доцент (МТУСИ)

Барков В.В., ассистент (МТУСИ)

Рецензент: Шелухин О.И., д.т.н., профессор (МТУСИ)

Издание утверждено на заседании совета факультета ИТ
Протокол № 9 от 16.03.2017 г.

Рекомендовано к изданию кафедрой «ИБ»
Протокол № 9 от 14.02.2017 г.

# РАЗДЕЛ 1
# ПРОЦЕДУРНОЕ ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ С

## Лабораторная работа №1
## Компиляция файлов исходного кода и компоновка полученных объектных файлов в исполняемый модуль. Линейные алгоритмы

### Цель работы

Овладеть навыками создания однофайловых и многофайловых проектов в интегрированной среде разработки Microsoft Visual Studio 2015 Community Edition и научится создавать линейные программы на языке С с применением арифметических операций.

### Краткие теоретические сведения

***Структура программы***

Программа на языке С состоит из одного или нескольких файлов исходного кода с расширением *.c. Общие для группы файлов исходного кода объявления функций, глобальных переменных, структур, объединений и перечислений могут быть вынесены в отдельный файл, называемый заголовочным файлом. Такие файлы обычно имеют расширение *.h. Файл исходного кода со всеми включёнными в него заголовочными файлами называется единицей трансляции.

Каждый файл исходного кода независимо от других файлов преобразуется компилятором в файл с объектным кодом *.obj. Далее множество объектных файлов связываются компоновщиком в исполняемый файл *.exe или динамическую библиотеку компоновки *.dll.

Множество объектных файлов также может быть упаковано специальной программой-архиватором в статическую библиотеку. Статические библиотеки вместе с заголовочными файлами могут поставляться другим разработчикам. Такие библиотеки подаются на вход компоновщику вместе с другими объектными файлами.

Объектные модули, исполняемые файлы и динамические библиотеки обычно содержат сегмент глобальных инициализированных данных, размер сегмента глобальных неинициализированных данных и сегмент кода.

***Идентификаторы***

В файлах исходного кода объявляются глобальные переменные и функции.

Функциям и переменным назначаются идентификаторы, с помощью которых к ним можно обращаться.

***Область видимости идентификаторов***

Идентификаторы могут иметь 4 различных области видимости: область видимости файла, области видимости блока, область видимости функции и область видимости прототипа функции.

Область видимости файла имеют идентификаторы, объявленные вне функции и вне круглых скобок. Их область видимость начинается от точки объявления и заканчивается в конце единицы трансляции.

Область видимости блока имеют формальные параметры функции и переменные, объявленные внутри блока. Их обрласть видимости начинается от точки объявления и заканчивается в конце блока.

Область видимости прототипа функции имеют формальные параметры в прототипе функции. Область видимости начинается от точки объявления и заканчивается в конце прототипа функции.

Область видимости функции имеют идентификаторы меток, использующиеся в инструкции goto. Область видимости ограничена телом функции.

***Связывание идентификаторов***

Для того чтобы идентификатор функции или глобальной переменной был доступен в других единицах трансляции он должен иметь внешнее связывание. Внешнее связывание имеют по

умолчанию все глобальные переменные и функции. Для явного указания внешнего связывания можно указать спецификатор extern.

Идентификаторы, имеющие внутреннее связывание, недоступны в других единицах трансляции. Такие идентификаторы объявляются с ключевым словом static.

***Продолжительность хранения***

Время жизни объектов определяется продолжительностью хранения.

Время жизни объектов со статической продолжительностью хранения является время выполнения программы. Они инициализируются один раз до начала выполнения программы. Такие объекты располагаются в сегменте данных.

Время жизни объектов с продолжительностью хранения потока является время выполнения потока. Объекты инициализируются во время запуска потока.

Время жизни объектов с автоматической продолжительностью хранения является время выполнения блока. Объект инициализируется в момент входа в блок (иногда в точке определения) и хранится в сегменте стека.

Время жизни динамически выделенных объектов определяется вызовами функций выделения и освобождения памяти. Такие объекты хранятся в области динамической памяти.

***Типы данных***

Каждый идентификатор имеет тип данных, который задаётся при его объявлении.

Язык С поддерживает типы данных, указанные в таблице 1.

Таблица 1 – Базовые типы данных языка С

| | Тип данных | Краткая запись | Литерал | Спецификатор преобразования |
|---|---|---|---|---|
| Стандартные знаковые типы | signed char | signed char | | %hhi %hhd |
| | signed short int | short int, short, signed short | | %hi %hd |
| | signed int | int, signed | 10 | %i %d |
| | | | 012 | |
| | | | 0xA | |
| | signed long int | long, signed long | 10L | %li %ld |
| | | | 012L | |
| | | | 0xAL | |
| | signed long long int | long long, signed long long, long long int | 10LL | %lli %lld |
| | | | 012LL | |
| | | | 0xALL | |
| Стандартные беззнаковые типы | unsigned char | unsigned char | - | %hhu %hho %hhx %hhX |
| | unsigned short int | unsigned short | - | %hu %ho %hx %hX |
| | unsigned int | unsigned | 10U | %u |
| | | | 012U | %o |
| | | | 0xAU | %x %X |
| | unsigned long int | unsigned long | 10UL, 10LU | %lu |
| | | | 012UL, 012LU | %lo |
| | | | 0xAUL,0xALU | %lx %lX |
| | unsigned long long int | unsigned long long | 10ULL, 10LLU | %llu |
| | | | 012ULL, 012LLU | %llo |
| | | | 0xAULL, 0xALLU | %llx %llX |
| | _Bool | _Bool | 0, 1 | |
| | float | float | 00.5F, .5F, 1.F, | |

| | | | | |
|---|---|---|---|---|
| Вещественные типы с плавающей точкой | | | 0.125E2F, 0.125E+2F, 0.125E-2F 5E2F, 5E+2F, 5E-2F | |
| | double | double | 0.5, .5, 1. 0.125E2, 0.125E+2, 0.125E-2, 5E2, 5E+2, 5E-2 | %f %F %e %E |
| | long double | long double | 0.5L, .5L, 1.L, 0.125E2L, 0.125E+2L, 0.125E-2L 5E2L, 5E+2L, 5E-2L | %lf %lF %le %lE |
| Символьные типы | char | char | ‘A’ | %c |

Тип данных переменных описывает размер, формат данных и допустимые операции.

Тип данных функции описывает количество и типы аргументов и тип возвращаемого значения.

***Объявление и определение идентификаторов***

Перед использованием идентификатор должен быть объявлен.

Объявления лишь указывают компилятору, какой тип имеет переменная или функция. Ниже представлены примеры объявления переменной и функций.

| Объявление глобальной переменной | Объявление глобальной переменной: |
|---|---|
| `extern short y;` | `double sin(double x);`<br>`void execute(void);` |

Для того чтобы компилятор разместил код функции в сегменте кода или разместил глобальную переменную в сегменте данных необходимо их определить.

Определение идентификатора одновременно является его объявлением. Ниже представлены примеры определения глобальных переменных и функций:

| Определения глобальных переменных | Определение функции |
|---|---|
| `int x1;`<br>`int x2 = 5;`<br>`static int x3 = 10;`<br>`static int x4;`<br>`extern int x5 = 15;` | `void execute(void)`<br>`{`<br>`}` |

Для сборки исполняемого модуля в одном из файлов объектного кода должно быть определение функции с одним из следующих типов:

```
int main(void);
int main(int argc, char *argv);
```

**Задание**

По номеру Вашего варианта выбрать задачу, решаемую в этой лабораторной работе, и выполнить для нее следующие задания.

***Задание 1***

Составить программу, имеющую линейный алгоритм и состоящую из одной функции `int main()`. Программу записать в файл с именем task1.c. Скомпилировать, скомпоновать и выполнить.

В функции main организовать вычисление задачи вашего варианта дважды:

- для исходных данных, значения которых задать в виде констант в тексте функции main;

- для исходных произвольных данных, значения которых пользователь Вашей программы должен ввести с клавиатуры в процессе выполнения программы.

Проанализировать результаты работы и сделать выводы. Перенести полученные результаты в отчет.

Вывод действительных чисел осуществлять с точностью до 0.0001

Примеры вывода:

| Для функции одной переменной | Для функции двух переменных |
|---|---|
| x = 5<br>y = 8.2537<br>f = 0.4514<br>x = 3.0051<br>y = 21<br>f = 84.5201 | x = 5<br>f = 0.4514<br>x = 3.0051<br>f = 84.5201 |

***Задание 2***

Линейный алгоритм функции из задания 1 разделить на две процедуры, выделив в одну вычислительные операции этого алгоритма, а в другую все операции ввода-вывода. Каждую процедуру оформить как функцию. Вычислительную часть алгоритма оформить как функцию с параметрами, передаваемыми по значению, и возвращаемым значением. Прототип функции:

```
double f(double x) // Если функция имеет один параметр
double f(double x, double y) // Если функция имеет два параметра
```

Другую часть алгоритма оформить как функцию `int main()`, вызывающую первую функцию нужное количество раз. Записать тексты функций в файл с именем task2.c в следующем порядке: функция с параметрами, функция main. Скомпилировать, скомпоновать и выполнить.

***Задание 3***

Создать файл task3.c, в котором изменить порядок записи текстов функций, созданных в задании 2. Функции записать в следующем порядке: функция main, функция с параметрами, организующая вычисления (`double f(double x)` или `double f(double x, double y)`). Внести требуемые дополнения, добиться успешной компиляции, скомпоновать и выполнить.

***Задание 4***

По-прежнему линейный алгоритм функции из задания 1 разделить на две процедуры, выделив в одну вычислительные операции этого алгоритма, а в другую все операции ввода-вывода.

Вычислительную часть алгоритма оформить как функцию без параметров и без возвращаемого значения. Прототип функции `void f(void)`

Другую часть алгоритма оформить как функцию `int main()`, вызывающую вычислительную функцию. Обмен данными между функциями организовать через глобальные объекты (`double x, y` – аргументы, `double result` – результат вычисления). Записать тексты функций в файл с именем task4.c в следующем порядке: функция main, затем функция без возвращаемого значения и без параметров. Скомпилировать, скомпоновать и выполнить.

***Задание 5***

В этом задании необходимо разделить текст файла task3.c на два файла. В первый файл с именем task5_main.c поместить текст функции main. Скомпилировать только файл task5_main.c.

Во второй файл с именем task5_func.c поместить текст функции с параметрами (`double f(double x)` или `double f(double x, double y)`). Скомпилировать только файл task5_func.c. После раздельной компиляции осуществить совместную компоновку. Полученный исполняемый файл выполнить.

Проанализировать результаты работы и сделать выводы.

***Задание 6***

В этом задании необходимо разделить текст файла task4.c на два файла.

В первый файл с именем task6_main.c поместить текст функции main. Скомпилировать только файл task6_main.c.

Во второй файл с именем task6_func.c поместить определения глобальных объектов (`double x, y` – аргументы, `double result` – результат вычисления) и текст функции без параметров (`void f(void)`). Скомпилировать только файл task6_func.c.

После раздельной компиляции осуществить совместную компоновку. Разобраться в проблемах, возникающих при совместной компиляции и компоновки. Полученный исполняемый файл выполнить.

Проанализировать результаты работы и сделать выводы.

***Задание 7***

В этом задании необходимо модифицировать тексты файлов из задания 6.

Описание функции и внешних переменных выделить в отдельный заголовочный файл func.h, включить его содержимое в файлы task7_main.c и task7_func.c. Определение внешних переменных произвести в файле task7_func.c.

Скомпилировать по отдельности файлы task7_main.c и task7_func.c. Произвести компоновку, выполнить полученный исполняемый файл.

***Задание 8***

Скомпилировать по отдельности файлы из задания 7 task7_main.c и task7_func.c. Создать статическую библиотеку task8_lib.lib, включающую объектный файл task7_func.obj. Полученную статическую библиотеку скомпоновать с файлом task7_main.obj. Выполнить полученный исполняемый файл.

**Индивидуальные варианты заданий**

| | |
|---|---|
| 1. $f(x) = 2\sin^2(3\pi - 2x)\cos^2(5\pi + 2x)$ | 2. $f(x) = \cos(x) + \sin(x) + \cos(3x) + \sin(3x)$ |
| 3. $f(x) = \cos^2(x) - \sin^2(x)$ | 4. $f(x) = \frac{\sin(2x)+\sin(5x)-\sin(3x)}{\cos(x)-\cos(3x)+\cos(5x)}$ |
| 5. $f(x) = 1 - \frac{1}{4}\sin^2(2x) + \cos(2x)$ | 6. $f(x) = \cos(x) + \cos(2x) + \cos(6x) + \cos(7x)$ |
| 7. $f(x) = \cos^2\left(\frac{3}{8}\pi - \frac{x}{4}\right) - \cos^2(\frac{11}{8}\pi + \frac{x}{4})$ | 8. $f(x,y) = \cos^4(x) + \sin^2(y) + \frac{1}{4}\sin^2(2x) - 1$ |
| 9. $f(x,y) = (\cos(x) - \cos(y))^2 - (\sin(x) - \sin(y))^2$ | 10. $f(x) = \frac{\sin(\frac{\pi}{2}+3x)}{1-\sin(3x-\pi)}$ |
| 11. $f(x) = \frac{1-2\sin^2(x)}{1+\sin(2x)}$ | 12. $f(x) = \frac{\sin(4x)}{1+\cos(4x)} \cdot \frac{\cos(2x)}{1+\cos(2x)}$ |
| 13. $f(x,y) = \frac{\sin(x)+\cos(2y-x)}{\cos(x)-\sin(2y-x)}$ | 14. $f(x) = \frac{\cos(x)+\sin(x)}{\cos(x)-\sin(x)}$ |
| 15. $f(x) = \frac{\sqrt{2x+2\sqrt{x^2-4}}}{\sqrt{x^2-4}+x+2}$ | 16. $f(x) = \frac{x^2+2x-3+(x+1)\sqrt{x^2-9}}{x^2-2x-3+(x-1)\sqrt{x^2-9}}$ |
| 17. $f(x) = \frac{\sqrt{(3x+2)^2-24x}}{3\sqrt{x}-\frac{2}{\sqrt{x}}}$ | 18. $f(x) = \left(\frac{x+2}{\sqrt{2x}} - \frac{x}{\sqrt{2x}+2} + \frac{2}{x-\sqrt{2x}}\right) \cdot \frac{\sqrt{x}-\sqrt{2}}{x+2}$ |
| 19. $f(x) = \left(\frac{1+x+x^2}{2x+x^2} + 2 - \frac{1-x-x^2}{2x-x^2}\right)^{-1}(5 - 2x^2)$ | 20. $f(x,y) = \frac{(x-1)\sqrt{x}-(y-1)\sqrt{y}}{\sqrt{x^3y}+xy+x^2-x}$ |
| 21. $f(x,y) = \frac{\sqrt{x}-\sqrt{y}}{x}$ | 22. $f(x) = \frac{4-x^2}{2}$ |
| 23. $f(x) = \frac{1}{\sqrt{x}+\sqrt{2}}$ | 24. $f(x) = -\sqrt{x}$ |
| 25. $f(x) = \sqrt{\frac{x+3}{x-3}}$ | 26. $f(x) = \frac{1}{\sqrt{x+2}}$ |
| 27. $f(x) = tg(2x) + \sec(2x)$ | 28. $f(x) = \frac{1+\sin(2x)}{\cos(2x)}$ |
| 29. $f(x) = ctg\left(\frac{3}{2}\pi - x\right)$ | 30. $f(x) = \frac{1-tg(x)}{1+tg(x)}$ |
| 31. $f(x) = ctg\left(\frac{5}{4}\pi + \frac{3}{2}x\right)$ | 32. $f(x,y) = -4\sin^2\left(\frac{x-y}{2}\right)\cos(x+y)$ |

| | |
|---|---|
| 33. $f(x,y) = \sin(y+x)\sin(y-x)$ | 34. $f(x) = \frac{\sqrt{2}}{2}\sin(\frac{x}{2})$ |
| 35. $f(x) = 4\cos\left(\frac{x}{2}\right)\cos\left(\frac{5}{2}x\right)\cos(4x)$ | 36. $f(x) = \cos^2(x) + \cos^4(x)$ |
| 37. $f(x) = tg(3x)$ | 38. $f(x) = 2\cos^2(x) - 1$ |
| 39. $f(x) = 2\sqrt{2}\cos(x)\sin\left(\frac{\pi}{4} + 2x\right)$ | 40. $f(x) = \frac{1}{4} - \frac{1}{4}\sin\left(\frac{5}{2}\pi - 8x\right)$ |

**Контрольные вопросы**

1) Что такое машинный код?
2) Что такое объектный файл?
3) Что такое исходный файл?
4) Какую работу выполняет компилятор?
5) Какую работу выполняет компоновщик?
6) Что такое заголовочный файл?
7) Что такое статическая библиотека?
8) Что такое динамическая библиотека?
9) Что такое исполняемый файл?
10) Что такое область видимости переменной?
11) На какие четыре группы можно разделить все переменные по области видимости?
12) Как определить переменную с областью видимости в пределах файла?
13) Как определить переменную с областью видимости в пределах блока?
14) Что такое связывание?
15) Чем отличаются глобальные переменные с внутренним и внешним связыванием?
16) Как определить переменную с внутренним связыванием?
17) Как определить переменную с внешним связыванием?
18) Как определить переменную без связывания?
19) Что такое продолжительность хранения?
20) Как определить переменную с автоматической продолжительностью хранения?
21) Как определить переменную со статической продолжительностью хранения?
22) В какой части памяти хранятся переменные со статической продолжительностью хранения?
23) В какой части памяти хранятся переменные с автоматической продолжительностью хранения?
24) Перечислите классы памяти переменных и укажите для каждого класса область видимости, связывание и продолжительность хранения. Как определяется переменная каждого класса?
25) Какие классы памяти могут иметь функции?
26) Что нужно добавить в код, чтобы можно было вызвать функцию, определённую в другом файле?
27) Что нужно добавить в код, чтобы можно было изменить значение глобальной переменной, определённой в другом файле?
28) Как сделать глобальную переменную или функцию недоступной из других файлов?
29) Какие действия выполняются при вызове функции?
30) Какие действия выполняются при возврате значения?
31) Что такое прототип функции?

## Лабораторная работа №2
## Разветвляющиеся алгоритмы

### Цель работы

Овладеть навыками создания разветвляющихся алгоритмов на языке С с применением инструкций выбора, условных выражений, логических операций и операций отношения

### Краткие теоретические сведения

В языке С разветвляющиеся алгоритмы реализуются с помощью инструкций выбора if, switch и условного выражения.

***Инструкция выбора if***

Инструкция выбора if имеет следующий синтаксис:

```
if (выражение)
	инструкция1;
else
	инструкция2;
```

В качестве условия может использоваться любое выражение.

Обычно в качестве условия используют операции равенства, операции отношения и их комбинации (логические операции).

К операциям равенства относятся операции равенства (==) и неравенства (!=).

К операциям отношения относятся операции больше (>), больше или равно (>=), меньше (<) и меньше или равно (<=)

К логическим операциям относятся логическое И (&&), логическое ИЛИ (||), логическое НЕ (!)

Логические выражения применяют ленивые вычисления: если по первому выражению возможно определить значение, второе выражения не вычисляется. В частности, если выражение1 равно 0 в операции И, либо выражение1 не равно 0 в операции ИЛИ, второе выражение не вычисляется.

Примеры выражений:

- $x > 4$ (x больше 4)
- $x >= 5.5$ (x больше или равно 5,5)
- $y < x$ (y меньше x)
- $y <= 20$ (y меньше или равно 20)
- $x >= 5 \&\& x < 10$ (x больше или равно 5 и x меньше 10, т.е. $x \in [5; 10)$)

Если выражение в круглых скобках имеет ненулевое значение, выполняется инструкция1, в противном случае – инструкция2. В качестве инструкции может выступать составная инструкция – множество инструкций заключённых в фигурные скобки { }

Блок else является необязательным. В случае выражение в скобках имеет нулевое значение и отсутствует блок else инструкция1 не будет выполнена и управление передастся следующей за if инструкции.

Инструкции выбора могут быть вложенными.

Примеры использования инструкции if:

```
if (x > 0 && y > 0)
	printf("Первая четверть");
else if (x < 0 && y > 0)
	printf("Вторая четверть");
else if (x < 0 && y < 0)
	printf("Третья четверть");
else if (x > 0 && y < 0)
	printf("Четвёртая четверть");
else if (x == 0)
	printf("На оси Y");
else
```

```
        printf("На оси X");
```

***Инструкция выбора switch***

Инструкция выбора switch позволяет выбрать одну из нескольких возможных ветвей. Она имеет следующий синтаксис:

```
switch (выражение)
{
case константа1:
        инструкция1;
case константа1:
        инструкция2;
default:
        инструкция:
}
```

Выражение в скобках вычисляется и сопоставляется с каждой константой. Если выражение совпадает с одной из констант, выполняются инструкции, начиная с метки case константа:.

В случае если выражение не соответствует ни одной константе, управление передаётся инструкции с меткой default. Метка default может отсутствовать, в этом случае управление передаётся инструкции, следующей за инструкцией switch.

Для того чтобы предотвратить исполнение кода в ветках, находящихся ниже, необходимо использовать инструкцию break;

Пример:

```
switch (_getch())
{
case '1':
        printf("Задание 1");
        break;
case '2':
        printf("Задание 2");
        break;
default:
        printf("Неверный ввод");
        break;
}
```

***Условное выражение***

Условное выражение имеет вид

```
выражение ? выражение1 : выражение2
```

Если значение выражения ненулевое, значение всего выражение будет равно выражению1, в противном случае выражению2.

Выражение1 и выражение2 должны иметь совместимые типы данных

Пример:

```
x > 5 ? x * x : -2 * x
```

**Задание**

По номеру Вашего варианта выбрать задачи и выполнить следующие задания.

***Задание 1***

Написать функцию, которая получает в качестве параметров координаты точки $(x, y)$ и определяет, попадает ли она в заштрихованную область на рисунке, который соответствует Вашему варианту (см. таблицу 1). Попадание на границу области считать попаданием в область.

Функция возвращает 1, если точка попадает в область, 0 в противном случае.

Прототип функции `_Bool isInArea(double x, double y).`

Функция не должна использовать функции консольного ввода-вывода.

***Задание 2***

Используя условную операцию (условие ? оператор : оператор) написать функцию для вычисления выражения, указанного в таблице 2.

Функция получает $x$ в качестве входного параметра и возвращает значение выражения.
Прототип функции `double f(double x).`
Функция не должна использовать функции консольного ввода-вывода.

***Задание 3***

Написать функцию main(), которая будет выводить меню и ожидать ответа пользователя:

1. Задание 1
2. Задание 2

Обработку ответа пользователя осуществить с помощью оператора switch. При вводе пользователем цифры 1 перейти к демонстрации задания 1. При вводе пользователем цифры 2 перейти к демонстрации задания 2.

Для демонстрации задания 1 запросить у пользователя ввод двух чисел, вызвать разработанную в задании 1 функцию и вывести на экран результат – попадает ли точка в заданную область или нет.

Для демонстрации задания 2 запросить у пользователя ввод числа $x$, вызвать разработанную в задании 2 функцию и вывести результат расчёта на экран.

Скомпилировать все файлы, скомпоновать и выполнить полученный исполняемый файл.

**Индивидуальные варианты заданий**

Таблица 1 – Индивидуальные варианты для задания 1

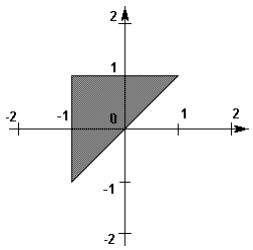
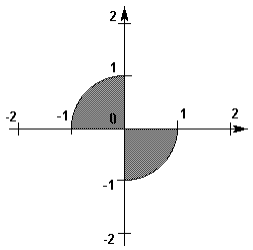
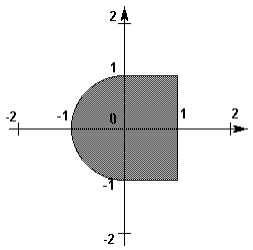
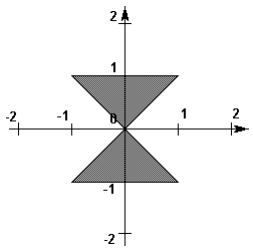
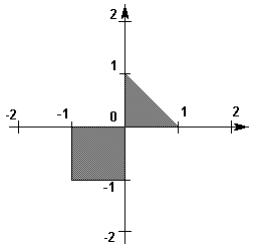
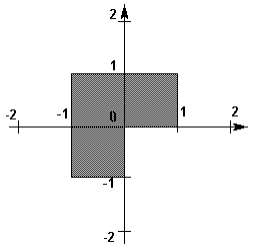
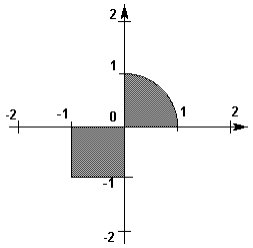
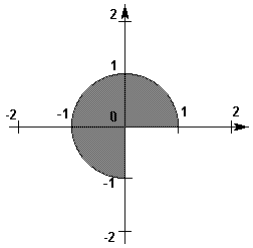
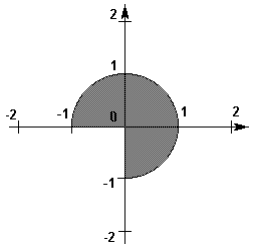
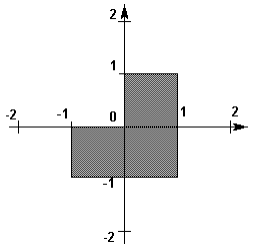
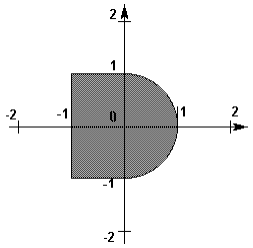
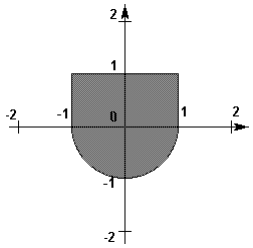
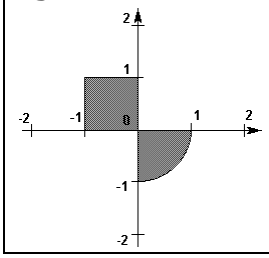
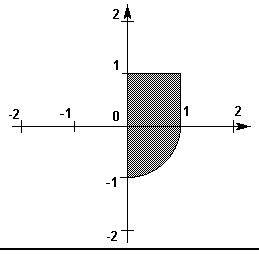
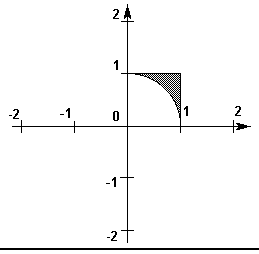
| | | | |
|---|---|---|---|
| 1  | 2  | 3  | 4  |
| 5  | 6  | 7  | 8  |
| 9  | 10  | 11  | 12  |
| 13  | 14  | 15  | |

Таблица 2 – Индивидуальные варианты для задания 2

| | |
|---|---|
| 1<br>$f(x)=\begin{cases} x^2-3x+9, x\le 3 \\ \dfrac{1}{x^3+6}, x>3 \end{cases}$ | 2<br>$f(x)=\begin{cases} 9-x, x\ge 1{,}1 \\ \frac{\sin(3x)}{x^4+1}, x<1{,}1 \end{cases}$ |
| 3<br>$f(x)=\begin{cases} -x^2-1{,}1x+9, x\le -3 \\ \dfrac{\ln(x+3)}{x^2+9}, x>-3 \end{cases}$ | 4<br>$f(x)=\begin{cases} \ln(x)+9,\ x>0 \\ 0,\ x=-\sqrt{7} \\ -\frac{x}{x^2-7},\ x\le 0, \\ x\ne -\sqrt{7} \end{cases}$ |
| 5<br>$f(x)=\begin{cases} \cos(2x)+9, x\le 9 \\ -\dfrac{\cos(x)}{x-9}, x>9 \end{cases}$ | 6<br>$f(x)=\begin{cases} x^2+3x+9, x\le 3 \\ \dfrac{\sin(x)}{x^2-9}, x>3 \end{cases}$ |
| 7<br>$f(x)=\begin{cases} 1{,}2x^2-3x-9, x>3 \\ \dfrac{12}{2x^2+1}, x\le 3 \end{cases}$ | 8<br>$f(x)=\begin{cases} x^4+9, x<3{,}2 \\ \dfrac{54x^4}{-5x^2+7}, x\ge 3{,}2 \end{cases}$ |
| 9<br>$f(x)=\begin{cases} 45x^2+5, x>3{,}6 \\ \dfrac{5x}{10x^2+1}, x\le 3{,}6 \end{cases}$ | 10<br>$f(x)=\begin{cases} -3x+9, x>3 \\ \dfrac{x^3}{x^2+8}, x\le 3 \end{cases}$ |
| 11<br>$f(x)=\begin{cases} 4x^2+2x-19, x\ge -3.5 \\ -\dfrac{2x}{-4x+1}, x<-3.5 \end{cases}$ | 12<br>$f(x)=\begin{cases} \lvert -x^2+3x+8\rvert, x\le 0 \\ \sqrt{x^2+\dfrac{1}{x}}, x>0 \end{cases}$ |
| 13<br>$f(x)=\begin{cases} \cos(2x), x\ge 0 \\ \dfrac{1}{x^2+1}, x<0 \end{cases}$ | 14<br>$f(x)=\begin{cases} \dfrac{1+x^2}{\sqrt{2x}}, x\ge 2 \\ \lvert 1-\sin(2x)\rvert, x<2 \end{cases}$ |
| 15<br>$f(x)=\begin{cases} \sqrt{x^2+4}, x\le 4 \\ \cos(\lvert 2+x\rvert), x<4 \end{cases}$ | |

**Контрольные вопросы**

1) Как организуются множественные действия в операторе условия if?
2) Какой формат записи имеет тернарный оператор условия?
3) Какой оператор условия рекомендуется использовать для программирования меню?
4) Как можно организовать переходы в различные точки программы на C++?
5) Какие логические операторы отношения используются в языке C++?
6) Что произойдет, если в операторе switch после метки case не использовать оператор break?
7) Что произойдет, если в операторе switch не поставить метку default и условие переключения не совпадет ни с одной меткой case?

## Лабораторная работа №3
## Регулярные и итерационные циклы

### Цель работы

Овладеть навыками создания циклических алгоритмов на языке С с применением инструкций цикла на примере алгоритмов работы с последовательностями чисел

### Краткие теоретические сведения

В языке С циклические алгоритмы реализуются с помощью инструкций цикла.

Регулярные циклы – это циклы, число итераций которых заранее известно.

Итерационные циклы – это циклы, число итераций которых заранее неизвестно.

Существует два типа циклов: с предусловием и с постусловием. Циклы с предусловием проверяют выражения до начала итерации, а циклы с постусловием – после завершения итерации. Если выражение изначально равно нулю, то цикл с предусловием не выполнится ни разу, а цикл с пост условием выполнится один раз.

К циклам с предусловием относятся циклы while и for. Циклом с постусловием является цикл do … while.

***Инструкция цикла while***

Цикл while имеет следующий синтаксис:

```
while(выражение)
	инструкция;
```

Пока выражение не равно нулю будет выполнятся инструкция.

***Инструкция цикла for***

Цикл for имеет следующий синтаксис:

```
for(выражение1; выражение2; выражение3)
	инструкция;
```

Перед началом цикла вычисляется выражение1. Перед началом каждой итерации вычисляется выражение2. Если оно отлично от нуля, выполняется инструкция. Перед завершением итерации вычисляется выражение3.

Обычно выражение1 инициализирует значение счётчика, выражение3 изменяет его значение

Цикл for может быть преобразован в цикл while:

```
выражение1;
while (выражение2)
{
	инструкция;
	выражение3;
}
```

Однако следует учесть, если инструкция является составной и содержит инструкции break и continue, то перед ними также должно быть добавлено выражение3.

***Инструкция цикла do … while***

Цикл do … while имеет следующий синтаксис:

```
do
	инструкция;
while(выражение);
```

Выполняется инструкция, а затем вычисляется выражение. Если оно не равно нулю инструкция повторяется.

***Примеры циклов***

Для вывода на экран чисел от 0 до 4 можно воспользоваться циклом:

| Цикл for | Цикл while | Цикл do … while |
|---|---|---|
| `for (int i = 0; i < 5; ++i)`<br>`  printf("%d\n");` | `int i = 0;`<br>`while (i < 5)`<br>`{` | `int i = 0;`<br>`do`<br>`{` |

| | | |
|---|---|---|
| | printf("%d\n");<br>++i;<br>} | printf("%d\n");<br>++i;<br>} while (i < 5); |

***Выход из цикла***

Для того чтобы избежать бесконечного цикла необходимо в теле цикла модифицировать хотя бы один элемент выражения.

Для того чтобы прервать выполнение текущей итерации и перейти к следующей необходимо воспользоваться инструкцией continue. Для досрочного завершения цикла необходимо воспользоваться инструкцией break;

Примеры использования циклов.

Печать чисел от 0 до 4, кроме 2:

| Цикл for | Цикл while | Цикл do … while |
|---|---|---|
| for (int i = 0; i < 5; ++i)<br>{<br>if (i == 2)<br>continue;<br>printf("%d\n");<br>} | int i = 0;<br>while (i < 5)<br>{<br>if (i == 2)<br>{<br>++i;<br>continue;<br>}<br>printf("%d\n");<br>++i;<br>} | int i = 0;<br>do<br>{<br>if (i == 2)<br>{<br>++i;<br>continue;<br>}<br>printf("%d\n");<br>++i;<br>} while (i < 4); |

**Задание**

По номеру Вашего варианта выбрать задачи и выполнить следующие задания.

Все указанные ниже функции разработать в трёх вариантах: с использованием цикла for, while, do … while. Функциям из задания дать одинаковые имена. Каждую функцию поместить в отдельный файл. Прототип функции вынести в заголовочный файл. В решении создать три проекта.

***Задание 1***

Написать функцию для вычисления выражения $\sum_{i=0}^{n-1} a_i$.

Функция в качестве параметра принимает значение $n$ и вычисляет сумму $n$ слагаемых.

Прототип функции `double summ(int n).`

Функция не должна использовать функции консольного ввода-вывода.

***Задание 2***

Написать функцию для вычисления выражения $\sum_{i=0}^{\infty} a_i$ с точностью $\varepsilon$.

Функция в качестве параметра принимает значение $\varepsilon$ и вычисляет сумму до тех пор, пока не выполнится условие $|a_i| \leq \varepsilon$.

Прототип функции `double summ2(double eps).`

Функция не должна использовать функции консольного ввода-вывода.

***Задание 3***

Написать функцию, которая печатает $n$ членов последовательности $\{a_i\}$, исключая из неё каждый $k$-ый член.

Числа $n$ и $k$ передаются в функцию в качестве параметра. Для исключения члена последовательности используйте оператор continue.

Прототип функции `void print(int n, int k).`

Пример вывода: 1.2 1.3 0.75

***Задание 4***

Написать функцию, которая возвращает номер первого члена последовательности $\{a_i\}$, для которого выполняется условие $|a_i| \leq \varepsilon$.

Функция в качестве параметра принимает значение $\varepsilon$.

Выход из цикла организовать с помощью оператора break.

Прототип функции `int findFirstElement(double eps).`

Функция не должна использовать функции консольного ввода-вывода.

***Задание 5***

Написать функцию, которая возвращает номер первого отрицательного члена последовательности $\{a_i\}$, для которого выполняется условие $|a_i| \leq \varepsilon$.

Функция в качестве параметра принимает значение $\varepsilon$.

Выход из цикла организовать с помощью оператора return.

Прототип функции `int findFirstNegativeElement(double eps).`

Функция не должна использовать функции консольного ввода-вывода.

***Задание 6***

Разработать функцию main(), которая в цикле выводит на экран меню из 6 пунктов и ожидает ответа пользователя:

1. Задание 1
2. Задание 2
3. Задание 3
4. Задание 4
5. Задание 5
6. Выход

Обработку пользовательского ввода выполнить с помощью оператора switch. При нажатии на соответствующую цифру меню стирается с экрана и приложение переходит в режим демонстрации работы выбранного задания. По завершении демонстрации программа ожидает пользовательского ввода для перехода в меню. Выход из программы должно осуществляться при нажатии цифры 6 в режиме меню. Скомпилировать 3 версии программы:

1) Все функции реализованы с помощью цикла for
2) Все функции реализованы с помощью цикла while
3) Все функции реализованы с помощью цикла do … while

Убедиться, что все три версии программы работают одинаково

**Индивидуальные варианты заданий**

| | | |
|---|---|---|
| 1<br>$a_i = (-1)^i \frac{1}{(i+1)(i+2)(i+3)}$ | 2<br>$a_i = (-1)^i \frac{i+1}{i^3+2}$ | 3<br>$a_i = (-1)^i (1 - \frac{2i-1}{2(i+1)})$ |
| 4<br>$a_i = (-1)^i \frac{i^2+1}{i^3+3}$ | 5<br>$a_i = (-1)^i \frac{i+1}{3^i+2^i}$ | 6<br>$a_i = (-1)^i (1 - \frac{(i+1)^2}{(i+2)^2})$ |
| 7<br>$a_i = (-1)^i \frac{2^i}{i^{i+1}+1}$ | 8<br>$a_i = (-1)^i (1 - \frac{2^i}{2^i+1})$ | 9<br>$a_i = (-1)^i \frac{i+1}{2^{i-1}}$ |
| 10<br>$a_i = (-1)^i \frac{i+1}{i^3-i^2+1}$ | 11<br>$a_i = (-1)^i \frac{2^{i+1}}{2^{2i}+1}$ | 12<br>$a_i = (-1)^i \frac{1}{i^2+2^i}$ |
| 13<br>$a_i = (-1)^i \frac{1+3i}{3^i}$ | 14<br>$a_i = (-1)^i \frac{i+1}{i^3+1}$ | 15<br>$a_i = (-1)^i \frac{i+1}{i^2+2i+1}$ |
| 16<br>$a_i = (-1)^i \frac{2(i+1)}{2+(i+1)(i-1)}$ | 17<br>$a_i = (-1)^i (1 - \frac{i^2+1}{i^2+3})$ | 18<br>$a_i = (-1)^i \frac{i+1}{i^2+1}$ |
| 19 | 20 | |

| | | |
|---|---|---|
| $a_i = (-1)^i \frac{1}{2(i+1)}$ | $a_i = (-1)^i \frac{i-1}{2i^2+1}$ | |

**Контрольные вопросы**

1) Какие циклы с предусловием существуют в языке С?
2) Какие циклы с постусловием существуют в языке С?
3) Чем отличаются циклы с предусловием от циклов с постусловием?
4) Какое минимальное количество раз может выполниться цикл с предусловием и цикл с постусловием?
5) Что такое регулярный цикл?
6) Что такое итерационный цикл?
7) Чем отличаются регулярные и итерационные циклы?
8) Каким образом можно пропустить итерацию или её часть?
9) Какими способами можно завершить цикл?

## Лабораторная работа №4
## Производные типы: указатели, массивы, строки, структуры, объединения. Статическое и динамическое выделение памяти

**Цель работы**

Овладеть навыками работы с производными типами данных в языке С, научиться динамически выделять память

**Краткие теоретические сведения**

***Указатели***

Переменные в языке С могут хранить не только значения базовых типов данных. Переменные могут хранить адреса памяти, назначенные другим переменным. Такие переменные имеют производный тип данных, называемый указателем.

Для объявления переменной-указателя необходимо добавить символ * к идентификатору. Для получения адреса существующей переменной необходимо воспользоваться операцией взятия адреса (&). Для того чтобы получить значение переменной, хранящейся по адресу, записанному в переменной необходимо воспользоваться операцией взятия объекта (*) (разыменовывание указателя)

| Определение указателя | Операция взятия адреса | Разыменовывание указателя |
|---|---|---|
| `int *x, *y;`<br>`double *z;` | `int x = 5;`<br>`int *y = &x;` | `int x = 1, *y = &x;`<br>`*y = 5;` |

Над указателями определены операции сложения, вычитания, инкремента и декремента.

*Операция инкремента* увеличивает адрес, хранимый в указателе таким образом, чтобы он указывал на элемент, следующий за текущим. *Операция декремента* уменьшает адрес, хранимый в указателе таким образом, чтобы он указывал на предыдущий элемент. *Операция сложения* изменяет адрес, хранимый в указателе таким образом, чтобы он указывал на n-ый элемент после текущего. *Операция вычитания* изменяет адрес, хранимый в указателе таким образом, чтобы он указывал на n-ый элемент до текущего. Вычитание указателей показывает, как далеко в памяти находятся два элемента. Если разность равна 1, то элементы являются соседними. Если разность равна 0, то указатели указывают на один и тот же элемент памяти.

***Массивы***

Массивы – это последовательность объектов одного типа. Элементы массива имеют порядковый номер и занимают в памяти соседние ячейки. Определяются массивы путём добавления к идентификатору квадратных скобок с указанием размерности массива.

Имя массива arr является указателем на его первый элемент. Для доступа к элементу массива можно использовать операцию индексации массива или операцию разыменования указателя. При этом указатель должен указывать на элемент, к которому необходимо обратиться.

| Операция индексации массива | Использование указателей |
|---|---|
| `int arr[5];`<br>`arr[0] = 5;`<br>`arr[1] = arr[0] * 2;` | `int arr[5];`<br>`*arr = 5;`<br>`*(arr + 1) = *arr * 2;` |

Обычно элементы массива обходятся в цикле.

| | |
|---|---|
| `for (int i = 0; i < 5; ++i)`<br>`  printf("arr[%d] = %d", i, arr[i]);` | `for (int *cur = arr; cur - arr < 5; ++cur)`<br>`printf("arr[%d] = %d\n", cur - arr, *cur);` |

***Строки***

Частным случаем массивов являются строки – массив символов типа char.

По соглашению, в конце строки добавляется нулевой символ. Он является маркером конца строки. Поэтому алгоритмам работы со строками необязательно передавать длину массива.

***Структуры***

Структура – это последовательность объектов, имеющих в общем случае разные типы. Элементы структуры имеют название и занимают в памяти соседние ячейки. Ниже показано определение структуры и пример её использования.

| Определение структуры | Использование структуры |
|---|---|
| `struct Person`<br>`{`<br>`  char firstName[20];`<br>`  char lastName[20];`<br>`  int age;`<br>`  double height;`<br>`  double weight;`<br>`};` | `void copy(char *dest, int maxSize, char *source) { /* поэлементное копирование массива опущено в целях экономии места */ }`<br><br>`void struct_example(void)`<br>`{`<br>`  struct Person p;`<br>`  const char firstName[] = "John";`<br>`  const char lastName[] = "Smith";`<br>`  copy(p.firstName, 20, firstName);`<br>`  copy(p.lastName, 20, lastName);`<br>`  p.age = 22;`<br>`  p.height = 180;`<br>`  p.weight = 80;`<br>`}` |

***Объединения***

Объединения объявляются и используются так же как и структуры с одним единственным отличием: вместо ключевого слова struct используется ключевое слово union. Элементы объединения начинаются с одного и того же адреса. Размер объединения определяется наибольшим размером элемента

***Перечисления***

Перечисления по сути являются набором именованных констант. Значения элементам перечисления могут быть заданы явно или неявно. Во втором случае значения элементам присваиваются от 0 и далее по порядку. Если одному из элементов присвоено значение, а следующему за ним нет, то следующий элемент будет иметь значение на единицу большее предыдущего.

| Определение перечисления | Пример использования |
|---|---|
| `enum Color { Red, Green, Blue };`<br>`enum Material { Metal = 1, Glass };` | `void enum_example(void)`<br>`{`<br>`  enum Color c = Red; //0`<br>`  enum Material m = Glass; //2`<br>`}` |

***Динамическое выделение памяти***

Указатели часто используются при работе с динамической памятью. Для того чтобы выделить память необходимо вызвать одну из функций malloc, calloc, realloc

Функция malloc принимает количество байт, которые необходимо выделить, и возвращает значение типа void * - универсального указателя на участок выделенной памяти.

Функция realloc принимает указатель на ранее выделенный блок памяти и новый размер. Если новый размер равен 0, то функция освобождает выделенную память и возвращает нулевой указатель. Если увеличить существующий блок не удаётся, функция выделяет новый блок памяти, копирует содержимое переданного блока, освобождает переданный блок и возвращает указатель на новый блок памяти.

Функция calloc принимает количество элементов count и размер size элемента и выделяет count*size байтов памяти, присваивая каждому выделенному байту нулевое значение.

Для того чтобы точно знать размер типа данных используется операция sizeof.

Для того чтобы присвоить значение переменной типа int * необходимо явное преобразование типа.

После того, как переменная больше не нужна, следует освободить память с помощью функции free. Функции free нельзя передавать указатель, который получен не от функций malloc, calloc, realloc.

Пример использования функций malloc, free:

| Выделение памяти с помощью malloc | Выделение массива с помощью malloc |
|---|---|
| `#include <stdlib.h>`<br>`void malloc_example(void)`<br>`{`<br>`  int *p = (int *)malloc(sizeof(int));`<br>`  *p = 5;`<br>`  free(p);`<br>`}` | `#include <stdlib.h>`<br>`void malloc_array_example(void)`<br>`{`<br>`  int *arr = (int *)malloc(5 * sizeof(int));`<br>`  for (int *cur = arr; cur - arr < 5; ++cur)`<br>`  *cur = cur - arr;`<br>`  free(arr);`<br>`}` |

Пример использования функций realloc, calloc, free:

| Выделение массива с помощью realloc | Выделение массива с помощью calloc |
|---|---|
| `#include <stdlib.h>`<br>`void realloc_example(void)`<br>`{`<br>`  int *arr = (int *)malloc(5 * sizeof(int));`<br>`  for (int *cur = arr; cur - arr < 5; ++cur)`<br>`  *cur = cur - arr;`<br>`  arr = realloc(arr, 10);`<br>`  for (int *cur = arr + 5; cur - arr < 10; ++cur)`<br>`  *cur = cur - arr;`<br>`  free(arr);`<br>`}` | `#include <stdlib.h>`<br>`void calloc_example(void)`<br>`{`<br>`  int *arr = (int *)calloc(5, sizeof(int));`<br>`  for (int *cur = arr; cur - arr < 5; ++cur)`<br>`  *cur = cur - arr;`<br>`  free(arr);`<br>`}` |

***Функции и производные типы данных***

Функции могут как принимать значения производных типов данных, так и возвращать их.

Массивы передаются в функцию только по указателю, при этом размер массива необходимо передавать дополнительным параметром:

| Способ 1 | Способ 2 |
|---|---|
| `void f1(int arr[], int n)`<br>`{ }` | `void f2(int *arr, int n)`<br>`{ }` |

Передача других производных типов ничем не отличается от передачи базовых типов.

Функции могут возвращать указатели. При этом на возвращаемое значение накладываются ограничения: нельзя возвращать адрес локальной переменной или параметра функции, так как после выхода из функции эта память будет освобождена; нельзя возвращать адрес, после того, как он был освобождён функцией free.

При возврате адреса из функции необходимо следить за тем, где будет освобождена выделенная память.

**Задание**

***Задание 1***

Написать функцию, которая в качестве параметра получает указатель на целое число и увеличивает его значение на единицу. Функция должна осуществлять проверку переданного указателя на нуль. Перед началом выполнения и перед выходом функция выводит адрес и значение параметра.

Прототип функции `void increment(int *n).`

Формат вывода: адрес значение адрес значение.

***Задание 2***

Написать функцию, которая создаёт на стеке целое число, осуществляет ввод, выводит на экран адрес переменной и её значение. Далее вызывает функцию из задания 1, а затем повторно выводит адрес и значение переменной.

Прототип функции `void incrementStackVariable().`

Формат вывода: адрес значение адрес значение адрес значение адрес значение.

***Задание 3***

Написать функцию, которая создаёт в динамической памяти целое число, осуществляет ввод, выводит на экран адрес переменной и её значение. Далее вызывает функцию из задания 1, а затем повторно выводит адрес и значение переменной. Перед выходом из функции необходимо освободить ранее выделенную память.

Прототип функции `void incrementHeapVariable().`

Формат вывода: адрес значение адрес значение адрес значение адрес значение.

***Задание 4***

Написать функцию вывода массива целых чисел на экран. Функция получает указатель на первый элемент массива и его длину.

Прототип функции `void writeArray(int *arr, int n).`

Формат вывода: 1 2 3 4 5 6

***Задание 5***

Написать функцию ввода массива целых чисел с клавиатуры. Функция получает указатель на первый элемент массива и его длину.

Прототип функции `void readArray(int *arr, int n).`

***Задание 6***

Написать функцию сортировки массива с использованием алгоритма сортировки, указанном в индивидуальном задании. Функция должна получать указатель на первый элемент массива и количество элементов в массиве.

Прототип функции `void sort(int *arr, int n).`

Функция не должна использовать функции консольного ввода-вывода.

***Задание 7***

Написать функцию, которая создаёт на стеке массив из N элементов (число N определяется константой в коде), выводит на экран N и с помощью разработанных ранее функций (задания 4-6) осуществляет ввод данных, вывод массива на экран, сортировку и повторный вывод отсортированного массива на экран

Прототип функции `void sortStackArray(int *arr, int n).`

Формат вывода (первая строка – количество элементов массива):

5

5 1 9 7 8

1 5 7 8 9

***Задание 8***

Написать функцию, которая создаёт в динамической памяти из N элементов (число N вводится пользователем с клавиатуры), выводит n на экран и с помощью разработанных ранее функций (задания 4-6) осуществляет ввод данных, вывод массива на экран, сортировку и повторный

вывод отсортированного массива на экран. Перед выходом из функции необходимо освободить выделенную ранее память.

Прототип функции `void sortHeapArray(int *arr, int n).`

Формат вывода (первая строка – количество элементов массива):

5

5 1 9 7 8

1 5 7 8 9

***Задание 9***

Разработать функцию main, демонстрирующую работу функций из заданий 2,3,7,8. Организовать меню и возможность многократной демонстрации заданий.

**Индивидуальные варианты заданий**

1. Сортировка выбором
2. Сортировка вставками
3. Пузырьковая сортировка
4. Сортировка Шелла
5. Быстрая сортировка
6. Сортировка слиянием

**Контрольные вопросы**

1. Что такое указатель?
2. Влияет ли изменение значения параметра функции на фактически переданную в функцию переменную, если передача происходит по значению?
3. Влияет ли изменение значения параметра функции на фактически переданную в функцию переменную, если передача происходит через указатель?
4. Можно ли возвращать указатель на локальный объект? Если нет, почему?
5. Как представляется массив в памяти?

# РАЗДЕЛ 2
# ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ С++

## Лабораторная работа №5
## Понятие класса. Члены класса. Друзья класса. Перегрузка операторов

**Цель работы**

Научиться создавать простые классы, создавать конструкторы, перегружать операции, создавать объекты классов и передавать их в функции

**Краткие теоретические сведения**

***Динамическое выделение памяти***

В языке С++ динамическое выделение памяти осуществляется с помощью оператора new, а освобождение с помощью оператора delete. Переменной можно задать начальное значение. Оператор new можно использовать также для выделения массива. При этом освобождение памяти осуществляется с помощью оператора delete[].

| Выделение памяти | Выделение памяти с инициализацией | Выделение памяти под массив |
|---|---|---|
| `int *a = new int();`<br>`// некие действия`<br>`delete a;` | `int *a = new int(0);`<br>`// некие действия`<br>`delete a;` | `int *a = new int[5];`<br>`// некие действия`<br>`delete[] a;` |

***Ссылки***

В языке C++ также появились типы-ссылки: lvalue и rvalue. Ссылки подобны указателям, однако они не могут работать с динамической памятью и не могут быть изменены после инициализации. Можно считать, что ссылки – это псевдонимы. Ссылки lvalue ссылаются на объекты, а rvalue обычно на временные объекты. Ссылки можно использовать при передаче параметров функции. В таком случае функция сможет изменить значения передаваемых переменных. Ниже представлен пример определения и использования ссылок.

| Определение ссылок | Использование ссылок |
|---|---|
| `int x = 5;`<br>`int &a = x; //lvalue ссылка`<br>`int &&b = x + 5; //rvalue ссылка` | `void swap(int &x, int &y)`<br>`{`<br>`  int temp = x;`<br>`  x = y;`<br>`  y = temp;`<br>`}` |

Однако в таком случае, передать временные объекты в функцию не получится. Для того чтобы это исправить, необходимо сообщить компилятору, что мы не будем изменять передаваемые параметры с использованием ключевого слова const либо передавать параметры с использованием rvalue-ссылок

| Передача параметров по ссылке на константный объект | Передача параметров по rvalue-ссылке |
|---|---|
| `void f(const int &a, const int &b) { }` | `void f(int &&a, int &&b) { }` |

***Перегрузка функций***

В языке C++ в отличие от C разрешено давать функциям одинаковые имена, если они имеют разные типы аргументов. Такое определение функций называется перегрузкой функции:

```
void f(void);
void f(int);
void f(double);
void f(double, int);
```

***Аргументы функции по умолчанию***

Язык C++ поддерживает аргументы функции по умолчанию. Суть в том, что при объявлении функции параметрам указываются значения аргументов по умолчанию. В случае, если при вызове функции аргумент не был передан, ему присваивается значение по умолчанию. Аргументы по умолчанию должны быть последними в списке аргументов.

| Аргументы по умолчанию в определении | Аргументы по умолчанию в прототипе | Примеры вызова |
|---|---|---|
| `void f(double x,`<br>`  double y = 0,`<br>`  double z = 1)`<br>`{`<br><br>`}` | `void f(double x,`<br>`  double y = 0,`<br>`  double z = 1);`<br><br>`void f(double x, double y,`<br>`  double z)`<br>`{`<br><br>`}` | `void example()`<br>`{`<br>`  f(1, 2, 3);`<br>`//x = 1, y = 2, z = 3`<br>`  f(1, 2); \`<br>`//x = 1, y = 2, z = 1`<br>`  f(5);`<br>`//x = 5, y = 0, z = 1`<br>`}` |

***Определение классов: данные-члены, функции-члены, уровень доступа к членам***

Язык C++ является объектно-ориентированным языком программирования. В C++ используется класс-ориентированный подход к созданию программ. Определение простого класса напоминает определение структуры в языке C.

В отличие от структур в языке C структуры и классы в языке C++ могут иметь не только данные, но и функции. Данные, определённые внутри структур и классов, называются данными-членами. Функции, определённые внутри структур и классов, называются функциями-членами. Классы и структуры в языке C++ отличаются друг от друга уровнем доступа к членам по умолчанию. Члены класса по умолчанию доступны только функциям-членам этого класса и не доступны другим функциям. Члены структур по умолчанию доступны любым функциям. При

определении структур и классов можно явно указать, какие члены будут видны извне, а какие только внутри. Для этого используются секции public, protected, private.

Ниже приведён пример определения структур и классов в языке C++

| Определение структуры | Определение класса |
|---|---|
| `struct A`<br>`{`<br>`    int x;`<br>`    int y;`<br>`    double z;`<br>`};` | `class A`<br>`{`<br>`    int x;`<br>`    int y;`<br>`    double z;`<br>`};` |

Обычно данные-члены делают закрытыми и добавляют открытые функции для взаимодействия с данными:

| Определение структуры с функциями-членами и секциями private и public | Определение структуры с функциями-членами и секциями private и public |
|---|---|
| `struct A`<br>`{`<br>`public:`<br>`    int GetX() { return x; }`<br>`    int GetY() { return y; }`<br>`    int GetZ() { return y; }`<br>`private:`<br>`    int x;`<br>`    int y;`<br>`    double z;`<br>`};` | `class A`<br>`{`<br>`public:`<br>`    int GetX() { return x; }`<br>`    int GetY() { return y; }`<br>`    int GetZ() { return y; }`<br>`private:`<br>`    int x;`<br>`    int y;`<br>`    double z;`<br>`};` |

Учитывая уровень доступа по умолчанию для структур и классов описанные выше структуру и класс можно определить следующим образом:

| Определение структуры с функциями-членами и секциями private и public | Определение структуры с функциями-членами и секциями private и public |
|---|---|
| `struct A`<br>`{`<br>`    int GetX() { return x; }`<br>`    int GetY() { return y; }`<br>`    int GetZ() { return y; }`<br>`private:`<br>`    int x;`<br>`    int y;`<br>`    double z;`<br>`};` | `class A`<br>`{`<br>`    int x;`<br>`    int y;`<br>`    double z;`<br>`public:`<br>`    int GetX() { return x; }`<br>`    int GetY() { return y; }`<br>`    int GetZ() { return y; }`<br>`};` |

Каждой функции-члену передаётся неявный аргумент this типа A * const, который может использоваться для доступа к членам класса. Однако члены класса доступны и без указания this, если только их не перекрывают локальные переменные или параметры. Если объект объявлен с ключевым словом const, то неявный аргумент this будет иметь тип const A * const и ему будут доступны только методы, помеченные словом const. Поэтому методы, не изменяющие состояние объекта должны определяться со словом const.

Функции-члены можно определять вне класса. В этом случае в определении класса указывается прототип, а вне определения класса определяется функция. К имени функции добавляется префикс класса. Префикс отделяется двойным двоеточием (::)

| Объявление константных функций | Определение функций членов вне класса |
|---|---|
| `class A`<br>`{`<br>`public:`<br>`    int GetX() const { return x; }`<br>`    int GetY() const { return y; }`<br>`    int GetZ() const { return y; }`<br>`    void Set(int x, int y)`<br>`    {`<br>`        this->x = x;`<br>`        this->y = y;` | `class A`<br>`{`<br>`public:`<br>`    void Set(int x, int y);`<br>`    // Ранее определённые функции опущены в`<br>`//целях экономии места`<br>`private:`<br>`    int x;`<br>`    int y;`<br>`    double z;` |

```
            z = x + y;
        }
private:
        int x;
        int y;
        double z;
};
```

```
};

void A::Set(int x, int y)
{
        this->x = x;
        this->y = y;
        z = x + y;
}
```

Обычно класс определяется в заголовочном файле, а реализация его методов в cpp файле.

***Статические члены класса***

Каждый объект класса имеет собственную копию данных-членов класса. Иногда требуется, чтобы все экземпляры класса имели одну копию данных. C++ позволяет определять такие данные с помощью ключевого слова static. Обращение к таким данным может осуществляться как через экземпляр объекта (как к обычным данным-членам), так и через имя класса.

Статические функции-члены отличаются от нестатических тем, что они не имеют неявного аргумента this, поэтому не могут обращаться к нестатическим функциям-членам.

Нестатические функции-члены могут обращаться к статическим членам класса.

Определение статических данных-членов и функций членов

```
class B
{
    static int x;
public:
    static int GetX() { return x; }
    static void SetX(int x) { B::x = x; }
};
```

Использование статических функций-членов

```
void example()
{
    B::SetX(1); // B::x равно 1
    B b;
    b.SetX(5);  // B::x равно 5
    int a = B::GetX();  // a равно 5
}
```

***Конструкторы***

При создании объекта вызывается специальная функция-конструктор. Если мы не предоставляем её реализацию, она генерируется автоматически.

Существуют 5 видов конструкторов:

- Конструктор по умолчанию – конструктор, не требующий аргументов
- Конструктор преобразования – конструктор, получающий единственный аргумент
- Конструктор инициализации – конструктор, получающий несколько аргументов
- Конструктор копирования – конструктор, получающий lvalue ссылку на другой экземпляр класса того же типа
- Конструктор перемещения – конструктор, получающий rvalue ссылку на другой экземпляр класса того же типа

При определении конструктора тип возвращаемого значения не указывается, имя совпадает с именем класса. Пример определения конструктора по умолчанию, конструктора преобразования, конструктора инициализации:

Определение конструктора по умолчанию

```
class A
{
public:
    A()
    {
        x = y = 0;
        z = 0.0;
    }
    //Ранее определённые функции
//опущены
private:
    int x;
    int y;
    double z;
};
```

Определение конструктора преобразования

```
class A
{
public:
    A(int a)
    {
        x = a;
        y = a;
        z = a;
    }
    //Ранее определённые
//функции опущены
private:
    int x;
    int y;
    double z;
```

Определение конструктора инициализации

```
class A
{
public:
    A(int a, int b, double c)
    {
        x = a;
        y = b;
        z = c;
    }
    //Ранее определённые
//функции опущены
private:
    int x;
    int y;
    double z;
```

| | | |
|---|---|---|
| | }; | }; |

При определении конструктора можно указать список инициализации. В этом случае значения членам класса будет присвоено в том порядке, в котором определены члены в классе, а не в том порядке, в котором они перечислены в списке инициализации. Использование списков инициализации эффективнее с точки зрения производительности.

В C++11 появились две новые возможности: делегирующие конструкторы и инициализация членов класса при объявлении. При определении нескольких видов конструкторов можно вызывать один конструктор из другого. Инициализация членов класса при их объявлении аналогична использованию списков инициализации. Если один и тот же член инициализирован при объявлении и в списке инициализации одного из конструкторов, будет выполнена только инициализация, указанная в списке инициализации.

Использование списков инициализации

```
class A
{
public:
    A(int a, int b, double c)
        : x(a), y(b), z(c)
    { }
    //Ранее определённые
//функции опущены
private:
    int x;
    int y;
    double z;
};
```

Делегирующие конструкторы

```
class A
{
public:
    A()
        : A(0, 0, 0)
    { }
    A(int a, int b, double c)
        : x(a), y(b), z(c)
    { }
    //Ранее определённые
//функции опущены
private:
    int x;
    int y;
    double z;
};
```

Инициализация при объявлении

```
class B
{
public:
    B()
    { /* x = 0, y = 0, z = 0*/ }
    B(int a)
        : x(a)
    { /* x = a, y = 0, z = 0*/ }
    B(int a, int b, double c)
        : x(a), y(b), z(c)
    { /* x = a, y = b, z = c*/ }
private:
    int x = 0;
    int y = 0;
    double z = 0;
};
```

При использовании аргументов по умолчанию можно заменить конструкторы инициализации, преобразования и конструктор по умолчанию одним конструктором.

Ниже приведено определение такого конструктора и примеры вызова конструкторов:

Пример определения конструктора с аргументами по умолчанию

```
class A
{
public:
    A(int a = 0, int b = 0, double c = 0)
        : x(a), y(b), z(c)
    { }
    //Ранее определённые
    //функции опущены
private:
       int x;
       int y;
       double z;
};
```

Примеры вызова конструкторов

```
A a1; // конструктор по умолчанию
A a2 = 1; // конструктор преобразования
A a3(1, 2, 3.0); // конструктор инициализации
A a4 = a1; // конструктор копирования
```

***Деструктор***

При уничтожении объекта вызывается специальная функция-деструктор. Пример объявления деструктора:

```
class A
{
public:
       ~A()
       {}
       //Ранее определённые функции опущены
private:
       int x;
       int y;
```

```
        double z;
};
```

***Передача объектов в функции***

Объекты в функции могут передаваться по значению, по ссылке или с помощью указателя. В случае передачи объекта по значению вызывается конструктор копирования. Если его определение отсутствует, оно дополняется компилятором.

В случае передачи объекта по ссылке копируется лишь адрес памяти, по которому расположен объект. Передача объектов в функцию по ссылке предотвращает копирование данных объектов и вызов конструктора копирования. Для объектов, занимающих много памяти и имеющих сложную логику копирования передача по ссылке предпочтительней. Для того чтобы предотвратить случайное изменение объекта, его следует передавать в функцию по ссылке на константу. Имея ссылку на константный объект, невозможно изменить его данные. Возможно лишь вызывать функции, помеченные ключевым словом const. Это предотвратит возможность случайного изменения объекта, а также позволит передавать в функцию временные объекты. Временные объекты можно передавать по rvalue-ссылке.

При создании объекта в динамической памяти, для управления объектом доступен лишь указатель. При этом доступ к членам осуществляется с помощью операции ->.

Ниже приведены примеры варианты передачи параметров в функцию.

| По значению | По ссылке | По ссылке на константу | По ссылке (rvalue) | По указателю |
|---|---|---|---|---|
| `void f(A a)`<br>`{`<br>`  a.Set(1, 2);`<br>`}` | `void f(A &a)`<br>`{`<br>`  a.Set(1, 2);`<br>`}` | `void g(const A &a)`<br>`{`<br>`  a.GetX();`<br>`}` | `void f(A &&a)`<br>`{`<br>`  a.Set(1, 2);`<br>`}` | `void f(A *a)`<br>`{`<br>`  a->Set(1, 2);`<br>`}` |

***Дружественные функции и классы***

Функции имеют доступ только к публичным членам класса. Однако в языке C++ существует механизм, позволяющий функциям и классам обращаться к приватным и защищённым членам. Такие функции и классы должны быть объявлены друзьями:

```
class B
{
private:
        int x;
        friend class A;
        friend void f(B b);
};

void f(B b)
{
        b.x = 10;
}
```

***Перегрузка операций***

Язык C++ также поддерживает перегрузку операций. Для перегрузки необходимо объявить функцию специального вида. Функция может быть как членом класса, так и свободной функцией. Если свободной функции требуется доступ к приватным членам класса, необходимо объявить её дружественной. Порядок аргументов при перегрузке важен. Пример:

```
class C
{
public:
        C operator+(const C &rhs)
        {
                C result;
                result.x = x + rhs.x;
                return result;
        }
        friend C operator-(const C &lhs, const C &rhs);
private:
        int x;
```

```
};

C operator-(const C &lhs, const C &rhs)
{
	C result;
	result.x = lhs.x + rhs.x;
	return result;
}
```

При перегрузке операции невозможно изменить приоритет операций, создать новые операции и изменить количество операндов.

Операция вывода в поток имеет следующий вид:

```
ostream &operator<<(ostream &stream, const C &instance)
{
	stream << x;
	return stream;
}
```

***Автоматически генерируемые функции-члены***

Для каждого класса компилятор автоматически генерирует конструктор по умолчанию, конструктор копирования, деструктор и оператор присваивания. При определении пользователем конструктора по умолчанию, конструктора инициализации или конструктора по умолчанию, компилятор не генерирует конструктор по умолчанию.

Для того чтобы явно запретить генерировать конструктор необходимо воспользоваться ключевым словом delete (начиная с C++11):

Если при определении собственного конструктора необходимо, чтобы компилятор сгенерировал конструктор по умолчанию, необходимо воспользоваться ключевым словом default (начиная с C++11):

| Запрет генерации конструктора по умолчанию и конструктора копирования | Указание сгенерировать конструктор по умолчанию |
|---|---|
| `class A`<br>`{`<br>`public:`<br>`  A() = delete;`<br>`  A(const A&) = delete;`<br><br>`private:`<br>`  int x;`<br>`  int y;`<br>`  double z;`<br>`};` | `class A`<br>`{`<br>`public:`<br>`  A() = default;`<br>`  A(int a, int b, double c)`<br>`  : x(a), y(b), z(c)`<br>`  { }`<br><br>`private:`<br>`  int x;`<br>`  int y;`<br>`  double z;`<br>`};` |

**Задание**

Выберите для выполнения лабораторной работы свой вариант.

Для выбранного варианта определите класс, включив в него:

- конструктор по умолчанию;
- конструктор инициализации;
- конструктор преобразования базового типа к типу, определяемому разрабатываемым классом;

В разрабатываемом классе перегрузите потоковые операции для объектов класса. Для выполнения задания предложенного варианта перегрузите необходимые математические операции.

При разработке класса вашего варианта учтите индивидуальные уточнения для функций – членов класса.

Разработайте для объектов вашего класса предложенные в каждом варианте пользовательские функции.

Разработайте функцию main, организующую ввод данных и демонстрацию работы разработанных функций

**Индивидуальные варианты заданий**

В соответствии с Вашим вариантом разработать класс. Название класса, приватные данные-члены и необходимые публичные функции-члены указаны в таблице ниже.

Для всех вариантов необходимо перегрузить операции + - * / и операцию приведения типа к типу double.

Составить пользовательскую функцию y, указанную в индивидуальном задании и необходимые вспомогательные функции

| Варианты | Название класса | Приватные данные-члены | Публичные функции-члены | Дополнительные функции |
|---|---|---|---|---|
| 1-5 | Complex | `double re;` (действительная часть)<br><br>`double im;` (мнимая часть) | `double Re() const;`<br>`double Im() const;`<br>`double R() const;`<br>`double Phi() const;` | `Complex y(const Complex &z);`<br><br>`Complex sin(const Complex &z);`<br>`Complex cos(const Complex &z);`<br>`Complex ch(const Complex &z);`<br>`Complex sh(const Complex &z);`<br>`Complex exp(const Complex &z);`<br>`Complex pow(const Complex &z1, const Complex &z2);` |
| 6-10 | Complex | `double r;` (модуль)<br><br>`double phi;` (аргумент) | `double Re() const;`<br>`double Im() const;`<br>`double R() const;`<br>`double Phi() const;` | `Complex y(const Complex &z);`<br><br>`Complex sin(const Complex &z);`<br>`Complex tg(const Complex &z);`<br>`Complex th(const Complex &z);`<br>`Complex sh(const Complex &z);`<br>`Complex pow(const Complex &z1, const Complex &z2);` |
| 11-15 | Rational | `int nominator;` (числитель)<br><br>`int denominator;` (знаменатель)<br><br>Функция сокращения дроби | `int GetNominator() const;`<br>`int GetDenominator() const;` | `Rational y1(const Rational &x);`<br>`double y2(double x);` |

Индивидуальные варианты заданий:

| `Complex y(const Complex &z);` | `Complex y(const Complex &z);` | `Rational y1(const Rational &x);`<br>`double y2(double x);` |
|---|---|---|
| 1.<br>$y(z) = 2z + sin(z - i)$ | 6.<br>$y(z) = 1 - z^5 - th(\frac{z}{2})$ | 11.<br>$y(\mathrm{x}) = 2x + \frac{1.3}{x}$ |
| 2.<br>$y(z) = z^2 - cos(2z)$ | 7.<br>$y(z) = 2z + e^5(1 + z)$ | 12.<br>$y(\mathrm{x}) = \frac{x}{3} - \frac{1}{0.2 + x}$ |
| 3.<br>$y(z) = \frac{z}{2} + ch(1 + z)$ | 8.<br>$y(z) = i - z \cdot sin(2z)$ | 13.<br>$y(\mathrm{x}) = 2.2 \cdot x^2 + x - 1$ |
| 4.<br>$y(z) = i + z * sh(1 + z)$ | 9.<br>$y(z) = 2 + z \cdot tg(z)$ | 14.<br>$y(\mathrm{x}) = \frac{x - 1.3}{x + 1.6}$ |
| 5. | 10. | 15. |

| | | |
|---|---|---|
| $y(z) = 2 + 3i * e^{-z}$ | $y(z) = z^3 + (1+2i)z^2 + (1-2i)z^{-5i}$ | $y(\mathrm{x}) = \frac{3}{7} + \left(\frac{5}{11}\right)x - \frac{2.5}{x}$ |

Для вариантов 11-15 в качестве процедуры сокращения используйте в этой функции алгоритм Эвклида для нахождения наибольшего общего делителя (НОД) двух целых чисел a и b:

Пусть b <= a и r остаток от деления a на b.

Тогда:

1. Если b = 0, тогда НОД = a.
2. Иначе a = b, b = r. Перейти к 1.

Используйте функцию сокращения каждый раз, когда создается новое значение рационального числа.

### Контрольные вопросы

1. Приведите пример определения класса.
2. Приведите пример описания класса.
3. Когда вызывается конструктор класса?
4. Какие типы конструкторов вы знаете?
5. Когда вызывается деструктор класса?
6. Какие виды доступа к членам класса бывают?
7. В чем особенность друзей классов?
8. Что такое перегрузка функций в целом и перегрузка операторов в частности?

## Лабораторная работа №6
## Классы с динамическими структурами данных. Шаблонные классы

### Цель работы

Изучить динамические структуры данных, овладеть навыками создания конструкторов копирования, перемещения, деструкторов, перегрузки операций копирования и перемещения.

### Краткие теоретические сведения

При разработке классов, содержащих ссылки на какие-либо ресурсы, будь то динамически выделенная память или другие ресурсы операционной системы, возникают проблемы при копировании объектов. Обычно ресурсы захватываются в конструкторе класса и освобождаются в деструкторе. Автоматически генерируемый конструктор копирования выполняет почленное копирование. В этом случае после копирования два объекта будут иметь ссылку на один и тот же ресурс и в момент удаления каждый из них попытается его освободить. Уже после того как первый объект освободит ресурс, второй объект будет работать некорректно. Но ещё хуже, когда второй объект повторно попытается освободить ресурс. Для того чтобы избежать таких ситуаций, необходимо предоставить собственную реализацию конструктора копирования.

***Конструктор копирования и оператор присваивания***

Пусть имеется определение класса Stack, представляющего стек на основе массива:

```
class Stack
{
public:
    Stack(int count);
    Stack(const Stack &other);
    Stack(Stack &&other);
    Stack &operator=(const Stack &other);
    ~Stack();
    int GetSize() const;
    void Push(int element);
    int Pop();
private:
    int *arr;
```

```
Stack::Stack(const Stack &other)
{
    arr = new int[other.count];
    count = other.count;
    for (int i = 0; i < count; ++i)
    {
        arr[i] = other.arr[i];
    }
}

Stack &Stack::operator=(const Stack &other)
{
    if (this == &other)
```

```
    int count;
};

Stack::Stack(int count)
    : count(count)
{
    arr = new int[count];
}
Stack::~Stack()
{
    delete[] arr;
}
```

```
    {
        return *this;
    }
    delete[] arr;
    arr = new int[other.count];
    count = other.count;
    for (int i = 0; i < count; ++i)
    {
        arr[i] = other.arr[i];
    }
}
```

Конструктор копирования для этого класса выполняет следующее: сначала выделяет объем памяти, достаточный для копирования элементов массива, а потом собственно копирует элементы.

При таком подходе описанная выше проблема исчезает для случаев инициализации объекта, однако это не решает проблему при присваивании одного объекта другому. Причиной этого является то, что оператор присваивания, автоматически генерируемый для класса, также выполняет почленное копирование. Для того чтобы исправить ситуацию необходимо явно определить оператор присваивания.

Логика оператора присваивания немного сложнее, поскольку прежде чем копировать данные, необходимо освободить выделенную память (по сути повторить логику деструктора), а потом повторить логику конструктора копирования. Для того чтобы безвозвратно не потерять данные при присваивании объекта самому себе, необходимо отдельно обработать этот случай.

Можно избежать дублирование кода. Деструктор можно вызвать явно, а логику конструктора копирования выделить в отдельную приватную функцию.

***Конструктор перемещения и оператор присваивания с перемещением***

В стандарте C++11 появились конструктор перемещения и оператор присваивания с перемещением. Они призваны улучшить производительность копирования, когда объект-источник является временным объектом. Вместо того, чтобы выделять новую память и копировать данные можно стать владельцем уже выделенной, однако при этом временный объект должен перестать владеть ресурсом. В примере с динамической памятью мы должны обнулить указатель.

Конструктор перемещения

```
Stack::Stack(Stack &&other)
{
    arr = other.arr;
    count = other.count;
    other.arr = nullptr;
}
```

Оператор присваивания с перемещением

```
Stack &Stack::operator=(Stack &&other)
{
    if (this == &other)
    {
        return *this;
    }
    delete[] arr;
    arr = other.arr;
    count = other.count;
    other.arr = nullptr;
}
```

***Обобщённое программирование***

Язык C++ поддерживает обобщённое программирование с помощью механизма шаблонов. Шаблонами могут быть функции и классы. В шаблонную функцию или класс вводится один или несколько параметров, которые разрешаются во время компиляции. Такими параметрами могут быть как целые числа, так и имена типов. Создание класса или функции из шаблона называется инстанцированием шаблона. Компилятор генерирует определение функции или класса, подставляя все необходимые параметры. Если такое определение ошибочно, компилятор выдаст ошибку. Применительно к контейнерам, параметром шаблона является тип данных элемента.

```
#pragma once
template<typename T>
class Stack
```

```
{
public:
	Stack(int count);
	Stack(const Stack &other);
	Stack(Stack &&other);
	Stack &operator=(const Stack &other);
	Stack &operator=(Stack &&other);
	~Stack();
	int GetSize() const;
	void Push(const T &element);
	T Pop();

private:
	T *arr;
	int count;
};

template<typename T>
Stack<T>::Stack(int count)
	: count(count)
{
	arr = new T[count];
}
```

**Задание**

Для типа динамической структуры данных, указанного в индивидуальном задании, разработать соответствующий класс, предусмотрев в нем конструкторы инициализации, копирования, перемещения, деструктор, функции вставки и удаления элемента, просмотра доступного элемента и функцию, проверяющую наличие элементов.

Перегрузить операции присваивания, присваивание с перемещением и потокового вывода для вывода содержимого динамической структуры на экран. Обязательные функции-члены:

| Stack | Queue | Deque |
|---|---|---|
| `int GetSize() const;`<br>`void Push(const T &element);`<br>`T Pop();`<br>`T Peek();` | `int GetSize();`<br>`void Push(const T &element);`<br>`T Pop();`<br>`T Peek();` | `int GetSize() const;`<br>`void PushFront(const T &element);`<br>`T PopFront();`<br>`void PushBack(const T &element);`<br>`T PopBack();`<br>`T PeekFront() const;`<br>`T PeekBack() const;` |

Класс разработать в двух вариантах

1. Элементом динамической структуры данных является целое число.
2. Класс является шаблонным, в котором тип элемента ДСД задаётся параметром шаблона

**Индивидуальное задание**

1. Стек на основе массива
2. Стек на основе однонаправленного списка
3. Стек на основе двунаправленного списка
4. Стек на основе однонаправленного циклического списка
5. Стек на основе двунаправленного циклического списка
6. Очередь на основе массива
7. Очередь на основе однонаправленного списка
8. Очередь на основе двунаправленного списка
9. Очередь на основе однонаправленного циклического списка
10. Очередь на основе двунаправленного циклического списка
11. Дек на основе массива

12. Дек на основе однонаправленного списка
13. Дек на основе двунаправленного списка
14. Дек на основе однонаправленного циклического списка
15. Дек на основе двунаправленного циклического списка

**Контрольные вопросы**

1. Какие члены должны быть переопределены для корректной работы классов, содержащие указатели?
2. Что такое шаблонные методы и классы?
3. Что такое инстанцирование шаблона?

## Лабораторная работа №7
## Одиночное и множественное наследование. Виртуальные и чисто виртуальные функции. Абстрактные и конкретные классы

**Цель работы**

Овладеть навыками создания базовых классов с виртуальными и чисто виртуальными функциями, а также производных классов с переопределением указанных функций

**Краткие теоретические сведения**

***Наследование***

Язык C++ поддерживает несколько механизмов повторного использования кода. Одним из таких механизмов является наследование. В C++ поддерживается как одиночное наследование (от одного класса), так и множественное наследование (от двух и более классов). Класс, от которого наследуют, называется базовым или родительским классом. Класс, которые наследует, называется дочерним или производным классом. Производный класс получает все члены, определённые в базовом классе, может обращаться ко всем членам, кроме приватных. Производный класс может добавлять свои члены (данные и функции).

C++ определяет три вида наследования: публичное, защищённое, приватное. При публичном наследовании уровень доступа к членам базового класса не изменяется. При защищённом наследовании публичные члены базового класса становятся защищёнными, остальные остаются без изменений. При приватном наследовании все члены базового класса становятся приватными.

| Определение базового класса | Определения производных классов |
|---|---|
| `class BaseClass`<br>`{`<br>`public:`<br>`  void f() {}`<br>`protected:`<br>`  void g() {}`<br>`private:`<br>`  void h() {}`<br>`};` | `class DerrivedA : public BaseClass`<br>`{ };`<br><br>`class DerrivedB : protected BaseClass`<br>`{ };`<br><br>`class DerrivedC : private BaseClass`<br>`{ };` |

Классы DerrivedA, DerrivedB и DerrivedC являются производными классами. Ключевое слово private в определении класса DerrivedC является опциональным.

Ссылка либо указатель на базовый класс может хранить адрес объекта производного класса. Эта возможность доступна всегда при использовании публичного наследования и ограниченно при использовании защищённого наследования.

***Виртуальные функции***

C++ поддерживает механизм переопределения функций. Для этого в базовом классе такие функции должны быть явно помечены ключевым словом virtual. Адреса таких функций помещаются в таблицу виртуальных функций класса, а в сам класс добавляется указатель на эту таблицу. Все вызовы виртуальных функций производятся через указатель. Конструктор

производного класса берет на себя ответственность правильно указать адреса всех переопределённых функций. Определение того, какая реализация функции будет вызвана определяется в момент выполнения (динамическое связывание). Адреса функций, не являющихся виртуальными, известны во время компиляции (статическое связывание).

***Чисто виртуальные функции и абстрактные классы***

Язык C++ предоставляет возможность объявлять в базовом классе функции без реализации. Такие функции называются чисто виртуальными. В таком случае класс называется абстрактным и создание его экземпляров невозможно. Производные классы должны определить все чисто виртуальные функции, либо они также будут абстрактными. Класс, не содержащий чисто виртуальных функций называется конкретным.

***Пример***

| Определение базового класса | Определение производного класса |
|---|---|
| `class BaseClass`<br>`{`<br>`public:`<br>`    virtual void f() { }`<br>`    virtual void g() = 0;`<br>`};` | `class DerrivedA : public BaseClass`<br>`{`<br>`public:`<br>`    void f() override { }`<br>`    void g() override { }`<br>`};` |

В данном примере функция f является виртуальной, а функция g чисто виртуальной. Обе функции переопределены в классе DerrivedA. Ключевое слово override появилось в стандарте C++11 и является опциональным. Ранее это ключевое слово не указывалось.

Вызов виртуальных функций в конструкторах и деструкторах может привести к ошибкам в работе приложения.

**Задание**

Для динамической структуры данных, разработанной в предыдущей лабораторной работе (стек, очередь или дек) создать абстрактный класс, выделив в него необходимые операции.

Унаследовать разработанный в предыдущей лабораторной работе класс от созданного в этой работе абстрактного класса.

Разработать ещё одну реализацию динамической структуры данных, указанной в индивидуальном задании. Разработать соответствующий класс, унаследовав его от абстрактного класса и определив все требуемые операции. Предусмотреть конструкторы инициализации, копирования, перемещения, деструктор, функции вставки и удаления элемента, просмотра доступного элемента и функцию, проверяющую наличие элементов.

Перегрузить операции присваивания, перемещения и потокового вывода для вывода содержимого динамической структуры на экран.

Класс разработать в варианте шаблона.

Создать функцию, получающую указатель на базовый класс и демонстрирующую работу

Создать функцию, получающую ссылку на базовый класс и демонстрирующую работу.

**Индивидуальное задание**

1. Стек на основе однонаправленного циклического списка
2. Стек на основе двунаправленного циклического списка
3. Стек на основе массива
4. Стек на основе двунаправленного списка
5. Стек на основе однонаправленного списка
6. Очередь на основе двунаправленного циклического списка
7. Очередь на основе однонаправленного циклического списка
8. Очередь на основе массива
9. Очередь на основе двунаправленного списка
10. Очередь на основе однонаправленного списка
11. Дек на основе массива

12. Дек на основе однонаправленного списка
13. Дек на основе двунаправленного списка
14. Дек на основе однонаправленного циклического списка
15. Дек на основе двунаправленного циклического списка

**Контрольные вопросы**

1. Что такое абстрактный класс?
2. Что такое конкретный класс?
3. Что означает наследование интерфейса?
4. Что означает наследование реализации?
5. В чем отличие публичного, защищённого и закрытого наследования?
6. В чем особенность виртуальных функций?
7. В чем особенность чисто виртуальных функций?
8. Для чего при множественном наследовании в списке производных классов используется ключевое слово virtual?

## РАЗДЕЛ 3
## ПАТТЕРНЫ ПРОЕКТИРОВАНИЯ

### Лабораторная работа №8
### Изучение паттернов проектирования

**Цель работы**

Научиться применять распространённые паттерны проектирования (Стратегия и фабричный метод) при создании своих проектов

**Краткие теоретические сведения**

Проектирование объектно-ориентированных программ – сложный процесс. Ещё более сложным является повторное использование кода. При решении конкретной задачи необходимо подобрать подходящие объекты, отнести их к различным классам, выбрать степень детализации, определить интерфейсы классов и иерархию наследования, установить существенные взаимосвязи между классами. С одной стороны, дизайн должен соответствовать решаемой задаче, с другой – быть общим.

Паттерн проектирования описывает часто встречающуюся задачу и принцип её решения, принимая во внимание то, что это решение будет использоваться много раз.

Э.Гамма выделяет три типа паттернов проектирования: порождающие паттерны, структурные паттерны и паттерны поведения.

Порождающие паттерны проектирования абстрагируют процесс инстанцирования. Они делают систему независимой от способа создания, композиции и представления объектов. К порождающим паттернам относятся такие паттерны, как «Абстрактная фабрика», «Строитель», «Фабричный метод», «Прототип», «Одиночка».

В структурных паттернах рассматривается вопрос о том, как из классов и объектов образуются более крупные структуры. Структурные паттерны уровня класса используют наследование для составления композиций из интерфейсов и реализаций. Структурные паттерны уровня объекта компонуют объекты для получения новой функциональности. К структурным паттернам относятся «Адаптер», «Мост», «Компоновщик», «Декоратор», «Фасад», «Приспособленец», «Заместитель».

Паттерны поведения связаны с алгоритмами и распределением обязанностей между объектами. В паттернах поведения уровня класса используется наследование, чтобы распределить поведение между разными классами. В паттернах поведения уровня объектов используется композиция. К паттернам поведения относятся «Цепочка обязанностей», «Команда»,

«Интерпретатор», «Итератор», «Посредник», «Хранитель», «Наблюдатель», «Состояние», «Стратегия», «Шаблонный метод», «Посетитель».

**Задание**

Используя паттерн проектирования «Стратегия» отделить логику хранения в отдельную иерархию классов, с базовым классом Storage.

2 стратегии взять из предыдущих двух лабораторных работ, третью стратегию разработать.

Используя паттерн проектирования «Фабричный метод» создать иерархию классов для создания объектов класса с базовым классом Storage.

Создать класс ДСД, указанный в индивидуальном задании.

Создать функцию main, демонстрирующую работу программы.

**Индивидуальное задание**

Стратегии хранения:

1) Массив, двунаправленный циклический список, однонаправленный список
2) Однонаправленный список, двунаправленный циклический список, массив
3) Двунаправленный список, массив, однонаправленный циклический список
4) Однонаправленный циклический список, двунаправленный список, массив
5) Двунаправленный циклический список, однонаправленный список, массив

Структура данных

1-5: очередь

5-10: дек

11-15: стек

**Контрольные вопросы**

1. Какие порождающие паттерны проектирования вы знаете?
2. Расскажите об одном из порождающих паттернов проектирования
3. Какие структурные паттерны вы знаете?
4. Расскажите об одном из структурных паттернов
5. Какие поведенческие паттерны проектирования вы знаете?
6. Расскажите об одном из поведенческих паттернов проектирования?

## СПИСОК ЛИТЕРАТУРЫ


1. Керниган Брайан У., Ритчи Деннис М. Язык программирования С, 2-е издание. : Пер. с англ. – М. : Издательский дом «Вильямс», 2011 – 304 с. : ил. – Парал. тит. англ.
2. Бьерн Страуструп. Язык программирования С++. Специальное издание. Перевод с английского под редакцией Н.Н.Мартынова. Москва. Издательство БИНОМ.2011. 1136 стр.
3. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приёмы объектно-ориентированного проектирования. Паттерны проектирования. – СПб.: Питер, 2012. – 368 с.: ил.
4. Прата Стивен. Язык программирования С. Лекции и упражнения, 6-е издание. : Пер. с англ. – М. : ООО «И.Д. Вильямс», 2016. – 928 с : ил. – Парал. тит. англ.
5. Прата Стивен. Язык программирования С++. Лекции и упражнения, 6-е издание. : Пер. с англ. – М. : ООО «И.Д. Вильямс», 2013. – 1248 с : ил. – Парал. тит. англ.

Айрапет Генрикович Симонян

Вячеслав Валерьевич Барков

# ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ СИСТЕМ ЗАЩИТЫ ИНФОРМАЦИИ

Лабораторный практикум

Подписано в печать _____ Формат 60×90 1/16

Объём 2 усл. п.л. Тираж ___ экз. Изд. №__ Заказ №__

ООО «Информпресс-94». Москва, ул. Складочная , д. 15а.