**#5 - Работа с указателями, а также ссылками**

**Указатели в C++**

Каждая переменная (int x = 1;) хранит данные в определенной ячейке в памяти компьютера. Каждый раз, создавая новую переменную, мы создаем новую ячейку в памяти, с новым значением в ней. Чем больше ячеек, тем больше компьютерной памяти будет занято.  
  
Адрес в памяти компьютера это число, к которому мы можем получить доступ. Указатель, в свою очередь - это тот же адрес в памяти, по которому мы получаем ту же самую переменную и, в итоге, то же самое значение.  
  
Чтобы работать с указателями необходимо воспользоваться двумя специальными символами: & и \*. Символ \* используется для получения адреса переменной, например:

int t = 237; // Простая переменная

int \*p; // Создание указателя,

// который принимает лишь адрес другой переменной

p = &t; // Устанавливаем адрес нашей первой переменной

Из примера выше, переменные t и p будут равны 237, при этом обе переменные используют лишь одну ячейку памяти, что сокращает общий объем используемой памяти.

**Ссылки в C++**

Ссылки и указатели схожи между собой, так как оба в качестве значения имеют лишь адрес некого объекта.  
  
Указатель хранит адрес ячейки и если мы захотим изменить значение этой ячейки, то нам придется выполнить операцию "*разыменования*":

float some = 391; // Простая переменная

float \*u = &some; // Указатель на переменную

\*u = 98; // Изменение значения переменной

В ссылках такого понятия нет, так как меняя ссылку вы автоматически меняете саму переменную. Ссылки напрямую ссылаются к переменной, поэтому их синтаксис проще.

char symbol = 'A'; // Простая переменная

char &ref = symbol; // Создание ссылки на переменную

// Поскольку мы ссылаемся на переменную, то можем её использовать

// как отдельно взятую переменную

cout << ref << endl; // Вывод символа "А"

ref = 'C'; // Изменение на символ "C"

Использование ссылок и указателей оправдано, если вы передаете данные в функции или же в различные объекты. Они отлично подойдут для передачи большого объема данных. Используя их, вы лишь передаете адрес объекта, а не объект целиком, что значительно сокращает код.

**Задание к уроку**

1. Создайте указатель на указатель и измените значение первичной переменной на 59.4. Переменная: float a = 8.1;.  
  
2. Создайте ссылку на переменную типа double. Создайте еще одну ссылку на первую ссылку. Используя вторую ссылку, измените значение главной переменной и выведите на экран все три ссылки.

Решение задания:

// Первое задание

using namespace std;

int main() {

setlocale(LC\_ALL, "Russian"); // Русская локализация консоли

float a = 8.1;

cout << "First value is - " << a << endl;

float \*ptr = &a;

float \*ptr\_2 = ptr;

\*ptr\_2 = 59.4;

cout << \*ptr\_2 << " - " << \*ptr << " - " << a << endl;

return 0;

}

// Решение второй задачи

using namespace std;

int main() {

setlocale(LC\_ALL, "Russian"); // Русская локализация консоли

double a = 8.1;

cout << "First value is - " << a << endl;

double &ref = a;

double &ref\_2 = ref;

ref\_2 = 37.8;

cout << ref\_2 << " - " << ref << " - " << a << endl;

return 0;

}

**#7 - Функции. Перегрузка функций, а также операторы**

**Функции**

Функции - небольшие подпрограммы, которые можно создавать и вызывать в любом удобном месте программы.  
  
Пояснение: функции и методы - это две одинаковые конструкции. Функции принято называть методами, когда те находятся внутри класса или структуры. Если же они находятся вне класса, то их называют функциями. Они выполняют одну и ту же роль, но названия у них разные. Такой же принцип у полей и переменных. Переменные вне классов и структур, а поля - в классах или же структурах.  
  
Все функции необходимо записывать перед main(). Если так и делать, то код будет нечитабельный, плюс придется долго скролить до тех пор, пока не найдете главную функцию main(). В языке C++ было принято решение, записывать *"скелет"* функции в начале документа и в конце реализовывать сам *"скелет"*.  
  
Чтобы создать функцию, необходимо указать тип данных, возвращаемый обратно с функции, а также назвать саму функцию. В случае если функция ничего не возвращает, то тип данных необходимо ставить void.  
  
Таким образом, функция может выглядеть следующим образом:

void test () {

cout << "Вывод чего-либо в консоль" << endl;

}

Такая функция не принимает параметров и ничего не возвращает. Все что она делает, так это пишет в консоль предложение: "Вывод чего-либо в консоль". Разнообразим функцию и добавим один параметр:

void test (char symbol) {

cout << symbol << endl;

}

Теперь функция получает параметр, который будет отображен в консоли.  
  
Кроме того, было бы логично создать "скелет" функции в начале документа:

void test (char symbol);

Если функция должна что-либо вернуть, то мы прописываем подходящий тип данных перед названием функции. Для передачи данных с функции используйте ключевое слово return.

double test (double some\_number) {

some\_number \*= 2;

return some\_number;

}

**Задание к уроку**

1. Сделайте функцию, которая сравнивает введенные числа от пользователя и возвращает результат в виде знаков >, < или =.  
  
2. Создайте несколько функций (с одним именем), которые будут выводить объединение нескольких слов.

1. Функция принимает два слова и выводит их через пробел;
2. Функция принимает два целых числа и выводит каждое число с новой строки;
3. Функция принимает одно число и одно булевое выражение и выводить их через пробел.

Решение задания:

// Первое задание

#include <iostream>

#include <clocale> // Поддержка кириллицы

using namespace std;

// возвращает знаки равенства

char ravenstvo(int a, int b);

int main() {

// Установка поддержки кириллицы

setlocale(LC\_CTYPE, "rus");

int a1, b1; // переменные для чисел

cout << " Введите первое число: ";

cin >> a1;

cout << " Введите второе число: ";

cin >> b1;

cout << a1 << " " << ravenstvo(a1, b1) << " " << b1;

return 0;

}

char ravenstvo(int a, int b) {

if (a < b)

return('<');

else if (a > b)

return('>');

else

return('=');

}

// Решение второй задачи

#include <iostream>

#include <string>

using namespace std;

// Указываем про существование функций

void printWords (string first, string second);

void printWords (int a, int b);

void printWords (string word, bool logic);

int main() {

printWords ("Hello", "world!");

printWords (15, 93);

printWords ("Boolean variable is ", true);

return 0;

}

void printWords (string first, string second) {

cout << first << " " << second;

}

void printWords (int a, int b) {

cout << a << endl;

cout << b << endl;

}

void printWords (string word, bool logic) {

cout << word << " " << logic;

}

**#1 - Заголовочные файлы.**

**Заголовочные файлы**

Файлы кода C++ (*с расширением .cpp*) не являются единственными файлами в проектах и программах. Есть еще один тип файлов, который называется заголовочный файл (файл заголовка, подключаемый файл или header file). Они имеют расширение .h, но иногда их можно увидеть и с расширением .hpp или вообще без расширения. Целью заголовочных файлов является удобное хранение предварительных объявлений для использования другими файлами.

**Задание к уроку**

1. Создайте функцию добавления двух чисел и поместите её в файл math.cpp, также создайте функцию вывода информации на экран и поместите её в файл print.cpp. Добавьте соответствующие заголовочные файлы. Файл math.h должен подключаться в файле print.h.  
  
В главном файле подключите оба заголовка и используйте две функции из двух файлов для получения результатов.

Решение задания:

// Файл math.cpp

#include <stdio.h>

#include <iostream>

int summ(int a, int b) {

int res = a + b;

return res;

}

// Файл math.h

#ifndef multiply\_h

#define multiply\_h

int summ(int a, int b);

#endif

// Файл print.cpp

#include "print.hpp"

#include <string>

#include <iostream>

void print(int word) {

std::cout << word << std::endl;

}

void print(std::string word) {

std::cout << word << std::endl;

}

// Файл print.hpp

#ifndef print\_hpp

#define print\_hpp

#include <string>

#include "math.h"

void print(int word);

void print(std::string word);

#endif

// Файл main.cpp

#include <iostream>

#include <clocale>

#include "math.h"

#include "print.hpp"

using namespace std;

int main() {

setlocale(LC\_ALL, "Russian");

print(summ(4, 6));

return 0;

}

#### #2 - Введение в ООП. Классы и объекты

#### ООП простыми словами

Поскольку на примере все усвоить гораздо проще, то давайте за пример возьмем робота, которого постараемся описать за счёт классов в ООП.

**Класс** в случае с роботом – это его чертёж. Экземпляром класса (объектом) называет целый робот, который создан точно по чертежу.

**Наследование** – это добавление полезных опций к чертежу робота. К примеру, берем стандартный чертёж робота и дорисуем к нему лазеры, крылья и броню. Все эти дорисовки мы сделаем в классе наследнике, основной функционал которого взят из родительского класса.

**Полиморфизм** – это общий функционал для всех роботов и не важно что каждый робот может очень сильно отличаться друг от друга. К примеру, в главном классе мы указываем возможность передвижения для всех последующих роботов. Далее в классе наследнике мы можем дополнительно указать возможность левитации для робота, в другом же классе укажем возможность передвижения по воде и так далее. Получается, что есть общий функционал что записан в главном чертеже, но его можно переписать для каждого последующего робота (для каждого наследника).

А **инкапсуляция** является для нас бронёй, защищающей робота. Под пластырем брони находятся уязвимые элементы, вроде проводов и микросхем. После прикрытия брешей с помощью брони (protected или private), робот полностью защищён от внешних вмешательств. По сути, мы делаем доступ ко всем полям лишь за счёт методов, тем самым прямой доступ к полю будет закрыт.

***Инкапсуляция*** *— это свойство, позволяющее объединить в классе и данные, и методы, работающие с ними и скрыть детали реализации от пользователя.*

***Полиморфизм*** *— свойство классов, позволяющее использовать объекты классов с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.*

***Наследование*** *— это свойство, позволяющее создать новый класс-потомок на основе уже существующего, при этом все характеристики класса родителя присваиваются классу-потомку.*

У всех классов методы могут отличаться, как и поля с конструкторами. Каждый класс позволяет создавать любое количество разных объектов, все из них имеют собственные характеристики.

#### Создание классов

Для создания класса необходимо прописать ключевое слово class и далее название для класса. Общепринято начинать названия классов с буквы в верхнем регистре, но если этого не сделать, то ошибки не будет.

В любом классе можно создавать поля (переменные), методы (функции), а также конструкторы.

Создав новый класс и поместив туда какую-либо информацию мы можем создавать на основе него новые объекты. Объекты будут иметь доступ ко всем характеристикам класса, которые отмечены модификатором public.

Существует три модификатора доступа:

* public - данные будут видны повсюду, как в классе, так и вне его;
* protected - данные будут видны только в классе, где они были созданы, а также в классах наследниках;
* private - данные будут видны только в классе, где они были созданы.

Пример простого класса приведен ниже:

class Book {

public:

int pages;

char name;

float weight;

void getInfoBook () {

cout << "В книге " << name << " находиться " << pages << " страниц. " << endl;

cout << "При этом она весит " << weight << endl;

}

};

На основе такого класса мы можем создать множество объектов. Каждый объект в данном случае будет представлять из себя конкретную книжку. Для каждого объекта мы можем указать уникальные данные: количество страниц, название книги и её вес.

Чтобы создать объект нам потребуется следующий код:

Book sherlock\_holms; // Создание объекта

sherlock\_holms.getInfoBook(); // Вызов метода класса

Чтобы брать данные из класса через объект необходимо ставить точку и указывать имя переменной или функции, которую мы хотим взять.