**Технологии программирования.**

**Лекция 2**

Достоинства языка С++:

* Гибкость и компактность.
* Эффективность, основанную на том, что семантика языка отражает архитектуру компьютера.
* Доступность.
* Переносимость. Степень сложности переноса с одной платформы на другую относительно проста по сравнению с другими языками (Конечно, если не считать Python)

Все в языке C++ основано на функциях. Основная программа описывается в функции в main.

*Int main(int argc, char \*argv[]) {*

*return 0;*

*}*

argc — количество аргументов, которые были переданные в программу

argv — указатель на массив аргументов

Объявление функции стоится по шаблону: тип название (аргументы) { Код }

Программа может содержать в себе различный директивы предкомпиляции. Они начитаются с #.

Существуют 2 типа файла:

* Заголовочный файл (.h)
* Файл исходного кода программы (.cpp).

Определение времени жизни и области видимости переменных

Время жизни переменной определяется следующими правилами:

* Переменные, объявленные на внешнем уровне, всегда имеют глобальное время жизни
* Переменные, объявленные на внутреннем уровне, имеют локальное время жизни. При помощи static возможно обеспечить глобальное время жизни переменной

Видимость переменной в программе определяется по правилам:

* Переменные, объявленные или определенные на внешнем уровне, видимы от точки объявления или определения до конца исходного файла
* Переменные, объявленные или определенные внутри блока, видимы только внутри этого блока

Модификаторы памяти

* **auto** — автоматическая переменная. Память выделяется в стеке и при необходимости инициализируется каждый раз при выполнении оператора, содержащего ее определение. Освобождение памяти — при выходе из блока
* **extern** — переменная определяется в другом месте программы
* **static** — статическая переменная. Время жизни — постоянное. Инициализируется один раз при первом выполнении оператора, содержащего определение переменной. В зависимости от расположения оператора описания статические переменный могут быть глобальными и локальными
* **register** — аналогично auto, но память выделяется по возможности в регистрах процессора.

Пространство имен

Использование пространство имен позволяет разрешать конфликты с именами в программе в случае различных сбоев. Для того чтобы определить пространство имен необходимо использовать namespace.

*namespace Jack*

*{*

*void fetch();*

*int pal;*

*}*

*namespace Jill*

*{*

*double fetch;*

*int pal;*

*}*

*Jack::pal = 12;*

Пространство имен можно объявлять и использовать. Для этого есть директива using.

*Namespace Jill*

*{*

*double fetch;*

*};*

*char fetch;*

*int main() {*

*using Jill::fetch;*

*dobule fetch;*

*cin >> fetch;*

*cin >> ::fetch;*

*}*

Для перевода всех переменных и функций из пространства имен в глобальную область используется *using namespace*

*std::cout << «Hello» << endl;*

*using namespace std;*

*cout << «Hello» << endl;*

Константы

* Десятичные. 10, 20, 30
* Восьмеричные. 010, 020, 030
* Шестнацатеричные 0xf, 0xab

Массивы

int mass[2];

float matr[6][7];

char data = «Это массив»; // data[10] = «\o»

Массивы в C++ нумеруются начиная с 0.

Многомерные массивы в С++ всегда передаются указатели.

Операции в С++

* Унитарные операции (++а, а++).
* Бинарные операции. Data += 10;
* Тернарная. Условие? Операнд1: Операнд2; max = (a > b) ? a : b;

Правила преобразования типов

Операнды char, usingned, char или short преобразуются к int:

* char расширяется нулем ил знаком в зависимости от умолчания
* unsigned char расширяется нулем, а знаковый — знаком
* Остальные не расширяются

Если один из операндов имеет больший тип, то другой преобразуется к нему

Если один операнд беззнаковый, то другой преобразуется к беззнаковому

Тип результата точно такой же как и участвующие операнды

В С++ возможно использовать явное или неявное преобразование типов:

* Явное — руками программиста выбирается тип, к которому преобразуется результат
* Неявное — выбирается автоматически в зависимости от типов

Функции

Вы и сами все знаете...

Особенности использования функции в языке С

Локализация переменных происходит так же как и в высокоуровневых языках программирования.

Передавать параметры возможно по значению, ссылке и по константной ссылке. Попытка изменить значение по переданной константной ссылке вызывает ошибку компиляции. В С имеется возможность часть параметров определить до вызова функции при ее описании. Такие параметры называются параметрами по умолчанию и при вызове функции они могут не указываться. Параметры по умолчанию являются необязательными и должны располагаться последними в списке формальных параметров.

*Int sum(int a, int b = 10) {*

*}*

*В данном примере b — установление со значением поумолчанию*

Такая возможность обеспечивается за счет порядка передачи параметров при вызове функции. Еще одной особенность является возможность использования функции с переменным числом параметров. В таких функциях количество параметров определяется в момент вызова, где фактические параметры подставляются на место многоточия в списке формальных параметров. Обязательным является указание в списке формальных параметров специального параметра, позволяющего определить либо количество фактических параметров, либо критерии по которому определяется последний параметр в списке фактических

*int sum(int x, ...) {*

*int result = 0;*

*int \*p = &x;*

*for(int i = 0; i < x; ++i) {*

*++p; // перемещение указателя*

*result += \*p; // Суммируем*

*}*

*return result;*

*}*

Перегрузка функции

На практике разным функциям присваивают разные имена, однако при решении задач, часто возникает ситуация, когда необходимо использовать похожие операции для разных типов или количества переменных. Для удобства работы эти операции делают одноименными. С этой целью используется перегрузка имен функции. 2 или более функции являются перегруженными, когда они имеют одинаковые имена, разные типы и\или количество формальных параметров.

Для определения конкретной функции подлежащей вызову, компилятор проверяет согласование типа и количество параметров и подставляет в точку вызова нужную функции. Говорят, что перегруженные функции различаются при помощи сигнатур. Для выполнения распознавания, компилятор в первом проходе осуществляет декорирование функции.

Декорирование функции предполагает, что:

* каждой функции присваивается декор, который определяет правила доступа к функции (публичный или приватный).
* Определяется имя функции перед которой ставиться @. # обозначает конец имя функции. q обозначает что у функции есть параметры передающиеся в функцию, затем идут типы аргументов функции (@sum#qii). Zc — char по значению, pi — указатель на int;

В первом проходе компилятор проверяет соответствие типов и имен. Во втором проходе в точку вызова записывается адрес точки вызова, декор которого соответствует списку фактических параметров.

При выполнении подстановки в адрес точки вызова выполняется согласование типов. Учитывается:

* Точное совпадение
* Правило приведения типов

Если привести параметры в соответствие невозможно, то вызов классифицируется как двусмысленный, возникает неоднозначность.

В случае использования перегруженных функции, классифицируют 3 типа неоднозначности:

* При преобразовании типов
* Использование аргументов по умолчанию
* Использование параметров-ссылок

Правила описания перегруженных функции

* Перегруженыне функции должны находиться в одной области видимости;
* Перегруженные функции могут иметь параметры по умолчанию, но при этом значения параметров должны совпадать. В различных вариантах перегруженных функции могут быть разное количество аргументов по умлочанию
* Функции не согут быть перегруженны, если описание их параметров отличается только модификатором const или ссылки.

Шаблоны функции

Шаблоны функции представляют собой обощенное определение функции, из которого компилятор автоматически создает представителя функции для заданного пользователя типа или типов параметров.

Когда компилятор создает по щаблону функции представителя, то говорят, что он порождает функцию.

Синтасис:

*template <class T>*

*void printarray(T\* array, const int count) {*

*for(int i = 0; i < count; i++) {*

*cout << array[i] << endl;*

*}*

*}*

Параметрами являются опеределение класса, параметрами на место которого подставляются пользователскими типами. Если их несколько, то они разделяются запятой. При описании функции, в качестве типов формальных параметров должны быть указаны либо имена шаблонных классов, либо реально существующие типы параметров. При порождении функции, шаблонный тип заменяется фактическим генерируется функция, создается ее декор и адрес порожденной функции, записывается в точку ее вызова.

Когда типовые шаблонны параметры задаются явно, ты выполняется неявное преобразование типов.

При определении шаблонов, можно указывать ключевое слово **export**, при этом шаблон будет перемещен в заголовочный файл с возможностью использования, но не все компиляторы поддерживают экспорт шаблонов.

Конкретизация шаблона предполагает, что может быть выполнено явное приведение типа шаблоного параметра, перед вызовом.

Указатель на функцию

В любой момент можно получить указатель на функцию, для того чтобы передать функцию в качестве параметра.

Void err(cgar\* p) {

}

*Void (\*p\_funct)(char\*);*

*int main() {*

*p\_funct = &err;*

*(\*p\_funct)(«data»);*

*}*