ОСНОВНЫЕ АЛГОРИТМИЧЕСКИЕ СТРУКТУРЫ

Дальневосточный государственный университет путей сообщения,
кафедра «Вычислительная техника и компьютерная графика»,
к.т.н., доцент Белозеров Олег Иванович.

Вопросы лекции:

1. Интегрированная среда для программирования
2. Линейный вычислительный процесс
3. Реализация разветвляющихся алгоритмов
4. Реализация циклических алгоритмов.

Язык СИ++

- Компилируемый статически типизированный язык программирования общего назначения.
- Совместим с языком С (но не стандартом языка С - С99).
- Поддержка разных парадигм:

  процедурное, объектно-ориентированное, обобщенное, функциональное, метапрограммирование.

Для работы с языком программирования нужна специальная среда:

- Microsoft Visual Studio.
- Xcode.
- NetBeans.
- Eclipse.
- CodeLite.
- Qt Creator.
- Code::Blocks.
- Dev-C++

Dev-C++ — это интегрированная среда для программирования на языках C и C++, работающая под управлением операционной системы Windows.

Среда Dev-C++ распространяется свободно с исходными кодами (на Delphi) по лицензии GPL.

Цель GNU GPL — предоставить пользователю права копировать, модифицировать и распространять (в том числе на коммерческой основе) программы, а также гарантировать, что и пользователи всех производных программ получат вышеперечисленные права. Принцип «наследования» прав называется «копилефт» (copyleft) и был придуман Ричардом Столлманом.

Достоинства оболочки Dev-C++:

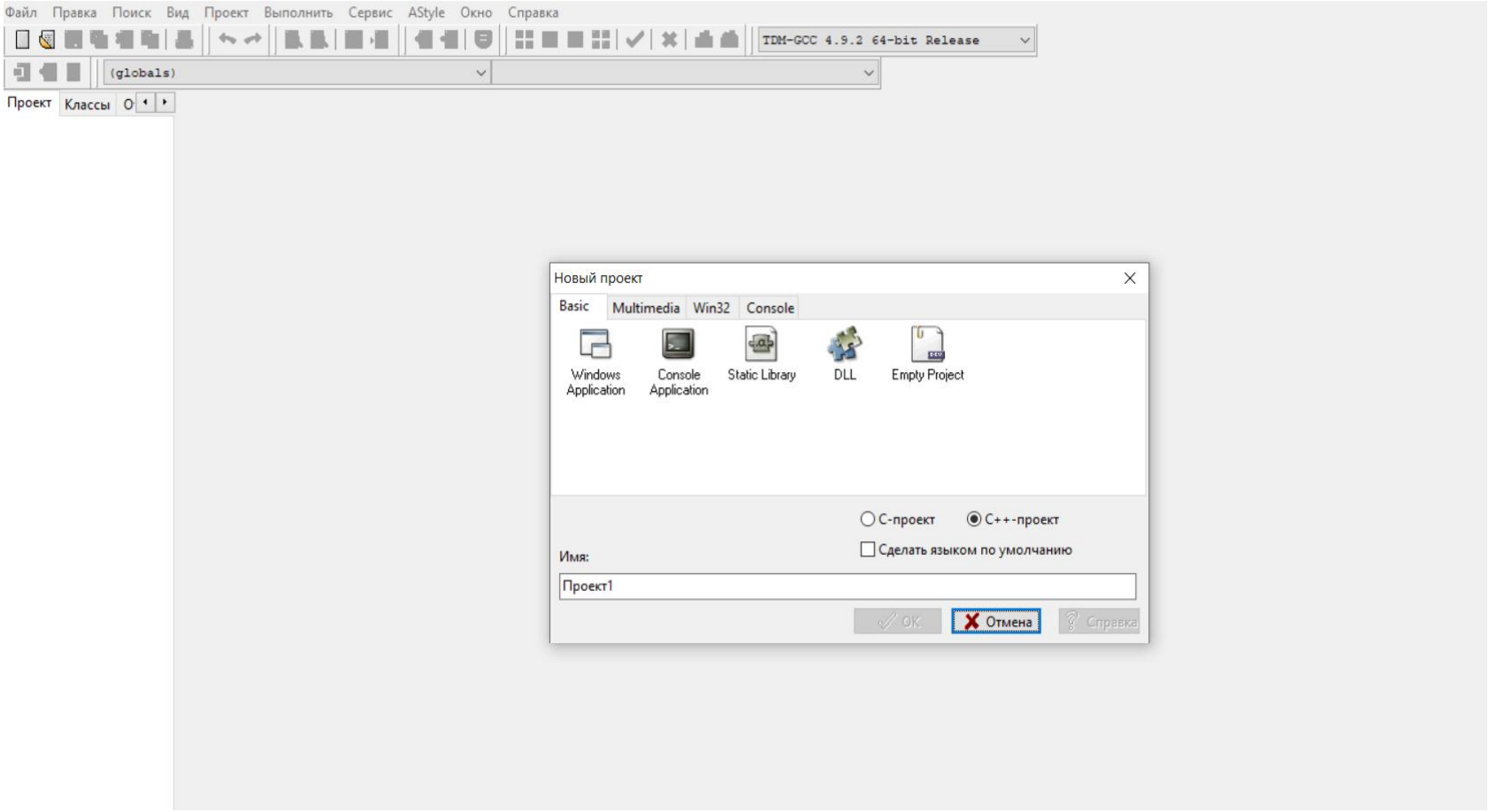
1) графический интерфейс;
2) русскоязычное меню;
3) встроенный отладчик;
4) возможность создавать консольные и графические программы.

Основные возможности отладчика GDB (GNU Debugger):

- символьная отладка программ как в терминах языков высокого уровня (СИ, СИ++), так и языка ассемблера (уровень машинных команд);
- запуск программ под управлением отладчика, анализ посмертного дампа (core-файла) аварийно завершившихся программ, "подключение" к уже выполняющейся программе;
- пошаговое выполнение программ;

- установка точек останова (возможно, условных) в программе;
- просмотр стеков (стек — абстрактный тип данных, список элементов, организованный по принципу LIFO (последний пришедший уходит первым) вложенных вызовов функций (процедур);
- установка точек слежения за переменными программы;
- изменение естественного хода вычислительного процесса в программе;
- изменение исполняемого кода программы;
- работа практически во всех распространенных универсальных ОС и на всех машинных архитектурах;
- кросс-отладка программ.

Создание нового проекта



Создание нового проекта

В главном окне выбрать пункты меню «Файл» - «Создать» - «Проект» - «Console Application» (см. рис.).

Задаем вводим имя нашего проекта (обязательно латинскими буквами) и место его размещения на диске. Более подробно процесс работы с программой для компиляции скриптов C++ можно изучить на видеоуроке в канале «ПРО ВСЕ».

Файлы и расширения

- Файл проекта .dev
- Файл с текстом программы .cpp
- Объектный файл .o
- Запускаемый файл .exe

Файлы и расширения

O-файлы содержат код C или C ++, используемый компилятором для создания файла для обработки других файлов исходного кода.

Файлы O содержат более компактный код. С помощью так называемого компоновщика такие файлы могут быть связаны для генерации библиотеки или исполняемого файла.

Файл с форматом O называется общим объектным файлом. Это означает, что он связан с различными компиляторами и программами.

Линейный вычислительный процесс

Алфавит языка C/C++ состоит из: прописных и строчных букв латинского алфавита, арабских цифр и специальных символов.

Линейный вычислительный процесс

К сложным типам данных относятся массивы, структуры (***struct***), объединения (***union***) и перечисления (***enum***).

К основным (базовым) типам данных относятся целый (***int***), вещественный (***float***, ***double***) и символьный (***char***) типы. В свою очередь, данные целого типа могут быть короткими (***short***) и длинными (***long***), со знаком (***signed***) и беззнаковыми (***unsigned***).

Запись самоопределенных констант

Тип данных	Общий формат записи	Примеры
Десятичные целые	**±*n***	**22 –15 176 –1925**
Вещественные с фиксированной десятичной точкой	**±*n.m***	**1.0 –3.125 –0.001**
Вещественные с плавающей точкой	**±*n.mE*±*p*** смысл записи $\pm n,m'10^{\pm p}$	**1.01E–10 0.12537*e*+4**
Символьные	**' * '**	**'*A*' '*x*' '0' '<'**
Строковые	**“ ****** ”**	**“*Хабаровск*” “*Press any key*”**

В таблице * – любой символ, набранный на клавиатуре.

Декларация объектов

Внимание! В языке C/C++ строчные и прописные буквы имеют различные коды, т.е. *PI*, *Pi* и *pi* – различные идентификаторы.

При декларации объектам можно задавать начальные значения (инициализировать), например:

```
int k = 10, m = 3, n;
double pi = 3.1415926;
```

Принято использовать в *ID* переменных строчные буквы, а в именованных константах – прописные, например:

```
const double PI = 3.1415926;
```

Разделителями *ID* являются пробелы, символы табуляции, перевода строки и страницы, а также комментарии.

Комментарий – любая последовательность символов, начинающаяся парой символов /* и заканчивающаяся парой символов */ или начинающаяся // и до конца текущей строки.

Директивы препроцессора

Перед компиляцией программы с помощью директив препроцессора выполняется предварительная обработка текста программы.

Директивы начинаются с символа # (шарп), за которым следует наименование операции препроцессора. Чаще всего используются директивы ***include*** и ***define***.

Директивы препроцессора

#include - это **директива препроцессора.**

*Директива **#include** используется для подключения к программе заголовочных файлов с декларацией стандартных библиотечных функций, например:*

<iostream> - это просто **аргумент**, поставляемый в дополнение к этой директиве, которая в данном случае является именем файла (функции ввода и вывода).

#include <stdio.h> – стандартные функции ввода-вывода;

#include <conio.h> – функции работы с консолью;

#include <math.h> – математические функции.

Директива **#define** (определить) создает макроконстанту и ее действие распространяется на весь файл, например:

#define PI 3.1415927

в ходе препроцессорной обработки идентификатор *PI* везде заменяется указанным значением 3,1415927.

Некоторые операции языка СИ++

*	**Умножение**
/	**Деление**
%	**Получение остатка**
+ (-)	**Сложение (вычитание)**
<<	**Сдвиг влево**
>>	**Сдвиг вправо**
<	**Меньше**
<=	**Меньше или равно**
>	**Больше**
>=	**Больше или равно**
==	**Равно**
!=	**Не равно**
&	**Побитовое И**
^	**Побитовое исключ. ИЛИ**
|	**Побитовое ИЛИ**
&&	**Логическое И**
||	**Логическое ИЛИ**

Что выведет данный код?

```
#include <iostream>
#define PI 3.14
using namespace std;

int main()
{
    int a = 205;
    int b = 0;
    b = a % 10;
    cout << "Hello World!\n" << a << endl << PI << endl << b;
    return 0;
}
```

Операция присваивания имеет полную и сокращенную формы записи.

Полная форма: ***ID = выражение;***

- – выполняется справа налево, т.е. сначала вычисляется *выражение*, а затем его результат присваивается указанному *ID*, например: y=(x+2)/(3*x)–5;
- В одном операторе можно присвоить значение нескольким переменным, например: x = y = z = 0;

или z = (x = y) * 5; – сначала переменной ***x*** присваивается значение переменной ***y***, далее вычисляется выражение ***x*5**, и результат присваивается переменной ***z***.\

Сокращенная форма: *ID* ***операция*** = *выражение;*

- где ***операция*** – одна из арифметических операций + , –, *, /, %; например: s += 7; (s = s+7;) или y *= x+3; (y = y*(x+3));
- Сокращенная форма применяется, когда переменная используется в обеих частях ее полной формы.

В языке C/C++ существуют операции ***инкремента*** и ***декремента***, т.е. уменьшения или увеличения значения переменной на 1. Операции могут быть ***префиксные*** и ***постфиксные.*** При использовании данной операции в выражении в префиксной форме, сначала выполняется сама операция (изменяется значение *i*), и только потом вычисляется выражение. В постфиксной форме – операция применяется после вычисления выражения.

Операции инкремента (--) и декремента (++).

Операции могут быть ***префиксные*** (++*i* и --*i*) и ***постфиксные*** (*i*++ и *i*--).

Например, для значений $b = 7$ и $n = 1$ будут получены следующие результаты:

c = b*++n;

*– порядок выполнения: n = n+1, c = b*n, т.е. c = 14;*

c = b*n++;

*– в этом случае: c = b*n, n = n+1, т.е. c = 7.*

Стандартные функции вывода:

Поточный ввод-вывод в C++ выполняется с помощью функций сторонних библиотек. В C++, как и в C, нет встроенных в язык средств ввода-вывода.

В C для этих целей используется библиотека **stdio.h**.
В C++ была разработана новая библиотека ввода-вывода `iostream`, использующая концепцию объектно-ориентированного программирования:

```
#include <iostream>
```

Стандартные функции вывода:

***puts*(*S*);**

***cout* << *n*;**

printf (*управляющая строка, список объектов вывода*);

Стандартные функции вывода: puts

***Puts* (*S*)** – функция выводит строку в стандартный поток вывода. После вывода строки производится переход на новую строку (вывод символа «новая строка»). Символ конца строки (нулевой символ) не выводится.

Синтаксис:
#include < stdio.h >
int puts (const char *s);

Аргументы:
s – указатель на строку, которую необходимо вывести.

Возвращаемое значение:
EOF - в случае ошибки.
Не отрицательное число, если вывод прошел успешно.

Стандартные функции вывода: puts

```
#include < stdio.h > // Для puts
int main (void)
{
const char *str = “Проверка работы функции puts.”;
//Вывод строки
puts (s);
return 0;
}
```

Стандартные функции вывода: puts

```
#include < stdio.h > // Для puts
#include <string.h>

int main(void)
{
const char* str = "Проверка работы функции puts.";
//Вывод строки
puts(str);
return 0;
}
```

Стандартные функции вывода: puts

≟ЁюткЁър ЁрсюЄѵ ЇєэъЎшш puts.

Стандартные функции вывода: puts

```
#include < stdio.h > // Для puts
#include <string.h>

int main(void)
{
setlocale(LC_ALL, "Russian");
const char* str = "Проверка работы функции puts.";
//Вывод строки
puts(str);
return 0;
}
```

Стандартные функции вывода: puts

```
#include < stdio.h > // Для puts
#include <string.h>
#include <iostream>
int main(void)
{
setlocale(LC_ALL, "Russian");
const char* str = "Проверка работы
функции puts.";
//Вывод строки
puts(str);
return 0;
}
```

Стандартные функции вывода: puts

Проверка работы функции puts.

Стандартные функции вывода: cout

Библиотека `iostream` определяет три стандартных потока:

- `cin` стандартный входной поток (`stdin` в C)
- `cout` стандартный выходной поток (`stdout` в C)
- `cerr` стандартный поток вывода сообщений об ошибках (`stderr` в C)

Для их использования в Microsoft Visual Studio необходимо прописать строку:

```
using namespace std;
```

Стандартные функции вывода: cout

Для выполнения операций ввода-вывода переопределены две операции поразрядного сдвига:

>> получить из входного потока

<< поместить в выходной поток

Вывод информации

cout << значение;

Здесь значение преобразуется в последовательность символов и выводится в выходной поток:

```
cout << n;
```

Возможно многократное назначение потоков:

cout << 'значение1' << 'значение2' << ... << 'значение n';

Пример:

```
cout << "Значение n равно" << n << "j=" << j;
```

Стандартные функции вывода: printf

printf (*управляющая строка*, *список объектов вывода*);

управляющая строка – заключенная в кавычки строка, содержащая спецификации преобразования объектов вывода, управляющие символы (признак «\») и любой набор символов, использующийся в качестве поясняющего текста – указывает компилятору вид выводимой информации;

список объектов вывода – печатаемые объекты (константы, переменные или выражения, вычисляемые перед выводом). Данные, указанные в списке, выводятся в соответствии со спецификациями управляющей строки.

Стандартные функции вывода: printf

Спецификации преобразования имеют вид

% <*флаг*> <*размер поля . точность*> *спецификация*

флаг: – (минус) выравнивание влево (по умолчанию выполняется выравнивание вправо); + (плюс) выводится знак положительного числа;

размер поля – задает ширину поля вывода (количество символов), при недостаточном значении выполняется автоматическое расширение;

точность – задает количество цифр в дробной части числа;

спецификация – формат преобразования выводимого объекта.

Стандартные функции вывода: printf

При необходимости вывода управляющих символов (**%** \ и т.п.) их нужно указать 2 раза, например:

```
printf("Только %d%% предприятий не работало. \n",5);
```

получим:

Только 5% предприятий не работало.

Стандартные функции вывода

Приведем основные форматы печати:

%d – десятичные целые (*int*);

%c – один символ (*char*);

%s – строка символов (*string*);

%f – данные типа *float*;

%ℓd – длинное целое;

%ℓf – данные типа *double*;

%x – шестнадцатеричные данные;

%o – восьмеричные данные.

Стандартные функции вывода

***Управляющие символы*:**

\n – переход на новую строку;

\t – горизонтальная;

\v – вертикальная табуляция;

\b – возврат назад на один символ;

\r – возврат в начало строки;

\a – звуковой сигнал;

\f – прогон бумаги до начала новой страницы;

\? – знак вопроса.

Стандартные функции ввода:

gets(S);

cin << n;

scanf (*управляющая строка, список адресов объектов ввода*);

Стандартные функции ввода: gets

Функция ***gets*** **(*S*)** обеспечивает ввод строки символов *S* до нажатия клавиши *Enter*, т.е. позволяет ввести строку, содержащую пробелы.

Функция gets() читает строку символов, введенных с клавиатуры и помещает их по адресу, указанному в аргументе. Можно набирать символы, пока не будет нажат ввод. Символ, соответствующий клавише ввод - возврат каретки, - не станет частью строки. Вместо этого в конце строки появится нулевой символ, и gets() закончит работу. Фактически невозможно использовать gets() для получения возврата каретки (можно использовать getchar() и ее варианты). Если при вводе допущены ошибки, то они могут быть исправлены нажатием на клавишу BACKSPASE перед нажатием ввода.

Стандартные функции ввода: gets

Имеется проблема, связанная с gets(), о которой следует знать: используя gets(), можно перейти границы массива, с которым она вызывалась. Это возможно, поскольку не существует способа указать gets(), где находится граница массива. Например, если вызвать gets() с массивом длиной в 40 байт, а затем ввести 40 или более символов, то произойдет выход за пределы массива. Это, естественно, вызывает проблемы и часто может привести к краху системы.

Стандартные функции ввода: gets

В качестве альтернативы можно использовать функцию fgets(), которая позволяет указать максимальную длину. Единственная проблема, связанная с fgets(), заключается в том, что она сохраняет символ новой строки. Если этот символ не нужен, то следует удалить его вручную.

Стандартные функции ввода: gets

Если функция gets() в вашем компиляторе не работает, все-таки она считается устаревшей и не везде поддерживается, используйте функцию gets_s().

Стандартные функции ввода: gets_s

```c
#include <stdio.h>
#include <string.h>

int main(void)
{
char str[80];
gets_s(str);
printf("Length is %d", strlen(str));
return 0;
}
```

Стандартные функции ввода: cin

cin >> идентификатор;

При этом из входного потока читается последовательность символов до пробела, затем эта последовательность преобразуется к типу идентификатора, и получаемое значение помещается в **идентификатор**:

```
int n;
cin >> n;
```

Возможно многократное назначение потоков:
cin >> переменная1 >> переменная2 >>...>> переменнаяn;

Стандартные функции ввода: cin

При наборе данных на клавиатуре значения для такого оператора должны быть разделены символами (пробел, \n, \t).

```
int n;
char j;
cin >> n >> j;
```

Стандартные функции ввода: cin

Особого внимания заслуживает ввод символьных строк. По умолчанию потоковый ввод cin вводит строку до пробела, символа табуляции или перевода строки.

Пример:

```
#include <iostream>
using namespace std;
int main()
{
  char s[80];
  cin >> s;
  cout << s << endl;
  system("pause");
  return 0;
}
```

Стандартные функции ввода: scanf

scanf (*управляющая строка*, *список адресов объектов ввода*);

в *управляющей строке* указываются только спецификации преобразований, а в *списке объектов ввода* – ***адреса*** вводимых скалярных переменных, для чего перед *ID* переменной указывается операция &, обозначающая «взять адрес». Для ввода значений строковых (составных) переменных символ & не указывается. При использовании формата %s строка вводится до первого пробела.

scanf ("%d %lf %s", &a, &b, str);

Вводить данные можно как в одной строке через пробел, так и в разных строках.

Стандартные функции ввода: scanf

Например, для считывания двух целых чисел функция вызывается так:

scanf("%d%d", &a, &b);

Основное отличие функции scanf в том, что при считывании чисел (или значений типа char) ей необходимо передавать **адреса** переменных (в языке С все параметры передаются по значению, поэтому чтобы функция scanf могла модифицировать переменную, необходимо передать в функцию адрес этой переменной). Поэтому перед названиями переменных мы пишем знак амперсанда ("&").

Стандартные функции ввода: scanf

В функции scanf могут быть явно записаны какие-то символы, кроме форматных строк. Например, вызов

scanf("%d:%d")

можно использовать для считывания времени, заданного в виде hh:mm - функция считает число, затем символ двоеточия, затем опять число.

Если в форматной строке встречается пробел, то он означает, что будет считано любое число пробельных символов, возможно даже нулевое.

Стандартные функции ввода: scanf

```
#include <stdio.h>
#include <string.h>
using namespace std;

int main(void)
{
char buf[5] = { '\0' };
scanf("%s", buf);

return 0;
}
```

Стандартные функции ввода: scanf_s

Если имеются проблемы с функцией scanf, есть смысл воспользоваться функцией scanf_s.

Функция scanf () - это исходная функция ввода, а функция scanf_s () - уникальная функция Microsoft VS.

Оба имеют одинаковые функции, но последний более безопасен и надежен.

Стандартные функции ввода: scanf_s

```
scanf_s("%s", buf, 5);
```

Стандартные математические функции:

Для использования математических функций необходимо подключить файл ***math.h (cmath)***.
Параметры и возвращаемые результаты большинства математических функций имеют тип ***double***.
Аргументы тригонометрических функций задаются в радианах. Напомним, что 2*p* (рад) равно 360^{o} (град).

Стандартные математические функции:

ID функции
sqrt(x)
fabs(x)
exp(x)
pow(x, y)
log(x)

Оператор условной передачи управления:

if **(***выражение***)** *оператор* **1;**
else *оператор* **2;**

вычисляется *выражение*, и если его значение не равно 0 (истинно), то выполняется *оператор* **1**, иначе – *оператор* **2**, например:

if(x>y) max = x; else max = y;

Если *операторы* **1, 2** содержат более одного оператора, то они заключаются в фигурные скобки **{ }**, т.е. применяется ***блок***.

Конструкция ***else…*** может отсутствовать и такую форму называют *сокращенной*, тогда в случае ложности условия управление передается на следующий за *if* оператор.

Оператор условной передачи управления:

Если *операторы* **1, 2** в свою очередь являются операторами *if*, то такой оператор называют *вложенным*, при этом ключевое слово *else* принадлежит ближайшему предшествующему *if*.

Например, найти наибольшее значение из трех чисел *x*, *y*, *z*: if (x>y)

if (x>z) max=x;

else max=z;

else if(y>z) max=y;

else max=z;

Операция сравнения:

Операции сравнения ***бинарные***, их общий вид:

Операнд 1 ***знак операции*** *Операнд* 2

Операндами операций сравнения могут быть данные любых базовых типов, значения которых перед сравнением преобразуются к одному типу.

Операция сравнения:

< (меньше),

<= (меньше или равно),

> (больше),

>= (больше или равно),

!= (не равно),

= = (равно).

Операция сравнения:

Логические операции используются в качестве условий при составлении более сложных выражений. Приведем их перечень в порядке убывания приоритета:

! (отрицание или логическое НЕ – унарная операция), **&&** (конъюнкция или логическое И), **||** (дизъюнкция или логическое ИЛИ).

Например: (0<x)&&(x<=100)
((!x)&&(y>0)||((z==1)&&(k>0))

Выражения вычисляются слева направо, причем их вычисление прекращается, как только результат становится известен.

Операция сравнения:

Тернарная (условная) операция ?:

Ее общая форма:

Операнд 1 **?** *Операнд* 2 **:** *Операнд* 3

Если значение *операнда* 1 истинно (не равно 0), то результатом операции является

операнд 2, иначе – *операнд* 3.

Например, найти наибольшее из двух чисел: max = a>b ? a : b;

Конструкция **switch-case** — это удобная замена длинной if-else конструкции, которая сравнивает переменную с несколькими константными значениями, например int или char.

```
switch ( <переменная> ) {
case значение1:
    Выполнить если <переменная> == значение1
    break;
case значение2:
    Выполнить если <переменная> == значение2
    break;
...
default:
    выполнить, если ни один вариант не подошел
    break;
}
```

Переменная в скобках сравнивается со значениями, описанными после ключевого слова **case**.
После двоеточия находится код, который будет выполнен в случае, если переменная оказалась равной текущему значению.
break необходим для того, чтобы прервать выполнение **switch**.

```
int a=1;
switch(a)
{
        case 1:
                a++;
        case 2:
                a++;
        case 3:
                a++;
}
cout<<"a= "<<a;
```

Данная программа выведет a = ?

Следующий вариант использования switch-case — неверен (в case не указано конкретное значение):

```
int a = 10;
int b = 10;
int c = 20;
 switch ( a ) {
case b:
    // Code
    break;
case c:
    // Code
    break;
default:
    // Code
    break;
}
```

Сообщение об ошибке:

test.cpp:9: error: 'b' cannot appear in a constant-expression

На следующем слайде пример правильного использования конструкции.

```
#include <iostream>
using namespace std;
void playgame()
{
        cout << "Play game called";
}
void loadgame()
{
        cout << "Load game called";
}
void playmultiplayer()
{
            cout << "Play multiplayer game
called";
}
 int main()
{
    int input;
   cout<<"1. Play game\n";
    cout<<"2. Load game\n";
    cout<<"3. Play multiplayer\n";
    cout<<"4. Exit\n";
    cout<<"Selection: ";
cin>> input;
   switch ( input ) {
   case 1:
      playgame();
      break;
   case 2:
      loadgame();
      break;
   case 3:
      playmultiplayer();
      break;
   case 4:
      cout<<"Thank you for playing!\n";
      break;
   default:
      cout<<"Error, bad input, quitting\n";
      break;
   }
     cin.get(); // это еще один вызов
функции, которая считывает данные из
входного потока данных и ожидает
нажатия клавиши ENTER.
        }
```

Для справки:

Тип данных **void** *- "универсальный" указатель, который просто соответствует указателю на какой-либо адрес в памяти компьютера, безотносительно типа данных, на которые указывает этот указатель. Функция* **void**, *которая используется в данной программе, не возвращает никаких значений. Это единственная такая функция в СИ.*

Сравнение switch-case с if-else

```
if ( 1 == input )
{
        playgame();
}
else if ( 2 == input )
{
        loadgame();
}
else if ( 3 == input )
{
        playmultiplayer();
}
else if ( 4 == input )
{
        cout << "Thank you for playing!\n";
}
else
{
        cout << "Error, bad input, quitting\n";
}
```

Реализация циклических алгоритмов:

Под циклом понимается многократное выполнение одних и тех же операторов при различных значениях промежуточных данных. Число повторений может быть задано в явной или неявной формах. Для организации повторений в языке C++ используются три различных оператора цикла.

Реализация циклических алгоритмов:

- Оператор цикла с предусловием ***while***
- Оператор цикла с постусловием ***do while***
- Оператор с предусловием и коррекцией ***for***

Реализация циклических алгоритмов: while

Оператор цикла с предусловием

while **(***выражение***)**

код цикла

организует повторение операторов *кода цикла* до тех пор, пока *выражение* истинно (не равно 0), если выражение = 0 (ложно) при первом входе, то код цикла не выполнится ни разу. Если код цикла состоит более чем из одного оператора, то организуется *блок*.

Реализация циклических алгоритмов: while

```
while (i <= k) // пока i меньше или равно k
{
// добавляем значение i к сумме
sum = sum + i;
// увеличиваем i на 1
i++;
}
```

Реализация циклических алгоритмов: do while

Оператор цикла с постусловием

do

код цикла

while **(***выражение***);**

организует повторение *кода цикла* до тех пор, пока выполняемое *выражение* истинно, после чего управление передается следующему за циклом оператору. Данный оператор гарантирует выполнение *кода цикла* хотя бы один раз.

Реализация циклических алгоритмов: do while

```
do {
// приглашение пользователю
printf("Введите число от 0 до 10: ");
scanf("%d", &num); // ввод числа
// повторяем цикл пока num<0 или num>10
} while ((num < 0) || (num > 10));
// выводим введенное значение num - от 0 до 10
printf("Вы ввели число %d", num);
```

Реализация циклических алгоритмов: for

Оператор с предусловием и коррекцией

***for* (*выражение* 1; *выражение* 2; *выражение* 3)**

код цикла

где *выражение* **1** – начальное значение параметра цикла; *выражение* **2** – проверка условия на продолжение цикла; *выражение* **3** – изменение (коррекция) параметра цикла.

Вначале вычисляется *выражение* 1, затем проверяется *выражение* 2, если оно – истинно, то выполняется код цикла, затем производится коррекция в *выражении* 3, и так до тех пор, пока *выражение* 2 не примет значение «*ложь*».

Досрочный выход из операторов цикла выполняет оператор ***break***, а оператор ***continue*** выполняет передачу управления в головной оператор цикла.

Реализация циклических алгоритмов: for

```
// цикл для переменной i от 1 до k с шагом1

for(int i=1; i<=k; i++)
{
 sum = sum + i; // добавляем значение i к сумме
 }
```

Сначала учите науку программирования и всю теорию. Далее выработаете свой программистский стиль. Затем забудьте все и просто программируйте.

George Carrette

Спасибо за внимание!