Основы программирования на C++

Лектор – Мигинский Денис Сергеевич.

Задачи курса (1 семестр)

- Изучить синтаксис языка C++
- Освоить структурный подход к программированию
- Изучить основные приемы программирования
- Выработать хороший стиль программирования

Почему C++?

1. Хорошая стандартизованность
2. Реализации на большинстве платформ
3. Переносимость
4. Высокая производительность
5. Гибкость
6. Поддержка большинства современных концепций программирования
7. Адекватность большинству задач биоинформатики
8. Хорошая база для изучения других современных языков

Краткая история C++

Создан в конце 70-х годов XX века Бьерном Страуструпом на основе языков C, Simula67, BCPL и Algol68.

Первоначальное назвние языка – C с классами.

Название C++ появилось в 1983 году.

C++ создавался как расширение языка C – фактически большинство программ на C являются также и программами на C++ (обратное неверно), хотя формально есть некоторая несовместимость.

В данный момент C++ является одним из наиболее строго стандартизованных языков (наряду с C и FORTRAN)

Hello, world!

```
//файл hello.cpp
#include <stdio.h>     //подключение библиотеки


int main (void)          //точка входа программы
{
        printf ("Hello, world!\n");  //печать на экран
        return 0;          //код возврата, выход
}
```

Комментарии

```
//Однострочный комментарий

/*Многострочный
  комментарий*/
```

Переменные

```
int main (void)
{
        int counter;
        double x = 1.2, y;
        ...
}
```

тип имя1 = значение1, имя2 = значение2, …;

Базовые типы: целые

bool	//boolean, 1 byte, true or false
char	//character, 1 byte
wchar_t	//unicode character, 4 bytes
short int (short)	//2 bytes, -2^{15} … $2^{15}-1$
int	//integer, 4 bytes, -2^{31} … $2^{31}-1$
long int (long)	//4 or 8 bytes

1 = sizeof(char) <= sizeof(short) <= sizeof(int) <= sizeof(long

Беззнаковые целые – unsigned

unsigned применим к char, short, int, long, wchar_t

```
unsigned int a;        //4 bytes, 0 ... 2^32-1

unsigned short b;

unsigned long int c;
```

sizeof (X) == sizeof (unsigned X)

Особенности целочисленной арифметики

char: -128…127 unsigned char: 0…255

```
char a = 127, b=1;
char c = a + b;
//c == -128
```

```
unsigned char a = 255, b=1;
unsigned char c = a + b;
//c == 0
```

char: 0 1 … 127 -128 -127 … -1
unsigned char: 0 1 … 127 128 129 … 255

```
int a = 1 / 2;          //a == 0
int b = 9 / 10;         //b == 0
```

Базовые типы: вещественные

```
float          //floating point, 4 bytes
double         //8 bytes
long double    //12 or 16 bytes
```

Обычно используется double

Целые константы

```
125          //целое десятичное

0126         //целое восьмеричное

0x2FE        //целое шестнадцатеричное

12U          //целое беззнаковое

12L          //целое типа long int

true
false        //логические константы
```

Вещественные константы

1.86 //по умолчанию все вещественные
//константы типа double

.25 == 0.25

12. != 12 //12. – вещественное, а 12 – целое!

3.8e-6 == 3.8E-6 //== $3.8*10^{-6}$

68.13F //константа типа float

Символьные и строковые константы

```
'a'          //символьная константа (char)
             //тождественна коду символа

"name"       //строковая константа

"name\n"     //строковая константа со
             //специальным символом \n
"long"
"Name"       // == "longName"
```

Специальные символы

\n	//новая строка
\r	//возврат каретки
\t	//табуляция
\\	//back-slash
\’	//одинарная кавычка
\”	//двойная кавычка

Имена и идентификаторы

Алфавит имен: A-Z, a-z, _, 0-9

Первый символ имени: A-Z, a-z, _

Правильные имена: loopCounter, i, _98a, x15

Неправильные имена: loop$Cnt, 9x

A != a

Перечисления

```
enum имя_типа {знач1=/*int*/, знач2=/*int*/, …}
```

```
enum Color {blue=1, green, red, yellow=14, white};
Color c = red;
//(int) c == 3;

//Color c = 3 – ошибка!

c = white;
//(int) c == 15;
```

Синонимы типов

typedef тип имя_синонима;

```
//объявление size_t, как синонима unsigned long
typedef unsigned long size_t;

unsigned long a = 10;
size_t b = 0;

//a и b имеют одинаковый тип
a = b;
```

Условные обозначения

expr – любое допустимое выражение, составленное из переменных, констант и операторов

type – имя типа

lvalue – (left value) изначально – выражение, которое может стоять в левой части оператора присваивания.
Более точно – объект, занимающий память.

Арифметические операторы

```
expr + expr        //сложение
expr - expr        //вычитание
expr * expr        //умножение
expr / expr        //деление
expr % expr        //деление по модулю (целые)
- expr             //унарный минус
+ expr             //унарный плюс
```

Битовые операторы

```
expr >> expr        //битовый сдвиг
expr << expr        //битовый сдвиг
expr & expr         //битовое AND
expr | expr         //битовое OR
expr ^ expr         //битовое XOR
~ expr              //битовое отрицание
```

Операторы присваивания

```
lvalue = expr          //присваивание

lvalue += expr         //lvalue = lvalue + expr

-=, *=, /=, %=, |=, &=, ^=, >>=, <<=

++lvalue               //lvalue += 1 префиксный
                       //инкремент
lvalue++               //постфиксный инкремент

--lvalue               //префиксный декремент

lvalue--               //постфиксный декремент
```

Особенности операторов присваивания

```
int a = 0;
int b = 0;
a = b++;
//a==0
//b==1
```

```
int a = 0;
int b = 0;
a = ++b;
//a==1
//b==1
```

```
int a = 0;
int b = 0;

//плохой стиль
a = b++ + b++;
//a==0
//b==2
```

```
int a = 0;
int b = 0;
a = b = 1;
//a==1
//b==1
```

```
int a = 0;
int b = 0;
(a = b) = 1;
//a==1
//b==0
```

Логические операторы

```
expr == expr        //сравнение на равенство
expr != expr        //сравнение на неравенство
expr > expr         //больше
expr < expr         //меньше
expr >= expr        //больше или равно
expr <= expr        //меньше или равно
```

Логические операторы

```
expr && expr        //логическое AND

expr || expr        //логическое OR

! expr              //логическое отрицание
```

Преобразование типов

Неявное преобразование:

- char->short->int->long->float->double->long double

- signed->unsigned (плохой стиль)
- enum->int
- все типы в bool

```
double a = 5 + 10;
double b = 5. + 10;
//int c = 5. + 10    ошибка
```

Явное преобразование: (type) expr

```
double a = 12.7;
int b = (int) a; //b == 12 (усечение)
```

Другие операторы

sizeof expr
sizeof (type) //размер типа или объекта

```
int a = sizeof (double);    //a == 8
int b = sizeof a;           //b == 4
```

() //управление приоритетом

```
int a = 10*(2+3);           //a == 50
```

expr ? expr : expr //условное выражение

```
int a = 10, b = 15;
int max = a > b ? a : b;    //max == 15
```

Приоритет операторов

- [] ++ --(пост-) ()(вызов функции) . ->
- sizeof ++ --(пре-) + -(унарные) ! *(разыменование) &(адрес) new delete ()(преобразование)
- * / %
- + -
- << >>
- < > <= >=
- == !=
- &
- ^
- |
- &&
- ||
- ?:
- = *= /= += -= %= >>= <<= &= |= ^=

Инструкции

Инструкция это:

1. Выражение, построенное из имен,
   констант и операторов в соответствии с синтаксисом.
   ```
   a = b + c*10;
   30;
   ```

2. Объявление переменных, типов, функций и т.д..
   ```
   int a = 1;
   double b;
   ```

3. Набор инструкций, объединенных в составную.

4. Одна из специальных управляющих инструкций.

5. Инструкция препроцессора.

Составная инструкция

```
int main (void)
{
        int a = 1;
        int c = 4;
        //a == 1, c == 4
        {                   //открытие составной инструкции
                int a = 2;
                //a == 2 – другой a!
                //c == 4 – тот же c!
                int b = 3;
                //b == 3
        }                   //закрытие составной инструкции
        //a == 1, c == 4
        //b не определен!
        return 0;
}
```

Условный выбор

if (expr) инструкция1 else инструкция2

Если expr == true, то выполняется инструкция1, иначе – инструкция2

```
if (a > 0)
        printf (“positive\n”);
else
        printf (“negative\n”);
```

```
if (a > 0)
{
        a--;
        printf (“success\n”);
}
else
{
        a = 0;
        printf (“failure\n”);
}
```

Цикл while

while (expr) инструкция

1. Если expr == false – выход из цикла
2. Выполняется инструкция
3. Переход к пункту 1

```
int a = 1;
int i = 2;
while (i <= 10)
{
        a *= i;
        ++i;
}
//a == 10!
```

```
//бесконечный цикл
while (true);
```

Цикл do while

```
do инструкция while (expr);
```

1. Выполняется инструкция
2. Если expr == false – выход из цикла
3. Переход к пункту 1

```
int a = 1;
int i = 2;
do
{
        a *= i;
        ++i;
} while (i <= 10);
//a == 10!
```

Цикл for

for (инструкция1; expr; инструкция2) инструкция

1. Выполняется инструкция1
2. Если expr == false – выход из цикла
3. Выполняется инструкция
4. Выполняется инструкция2
5. Переход к пункту 2

```
int a, i;
for (a = 1, i = 2; i <= 10; ++i)
        a *=i;
//a == 10!
```

```
//бесконечный цикл
for (;;);
```

```
int a, i;
for (a = 1, i = 1; i <= 10; ++i, a *= i /*a *=++i */);
//a == 10!
```

Управление циклами

break – безусловный выход из цикла

continue – пропускает текущую итерацию цикла

```
int a = 1;
int i = 1;
while (true)
{
        if (++i > 10) break;
        a *= i;
}
//a == 10!
```

```
int a = 1;
int i = 1;
while (++i <= 20)
{
        if (i > 10) continue;
        a *= i;
}
//a == 10!
```

Безусловный переход

```
goto метка;          //переход к метке
метка:               //метка
```

Избегайте goto!!!

```
int i, j;
for (i = 0;/*i < 10*/; ++i)
{
        for (j = 0; j < 10; ++j)
        {
                if (i >= 10) goto postLoop; //”break”
        }
}
postLoop:
```

Ветвление

1. Вычисляется значение expr.
2. Если значение expr совпадает с одним из значений в метках case, переход к метке.
3. В противном случае переход к метке default.
4. Инструкции выполняются до конца ветвления, либо до break

```
switch (expr)
{
case знач1:
        набор_инструкций1 //break;
case знач2:
        набор_инструкций2 //break;
...
default:
        набор_инструкцийD
};
```

Ветвление: пример

```
enum LightSignal {red, yellow, green} light = red;
switch (light)
{
case red:
        printf (“Red light\n”);
        break;
case yellow:
        printf (“Yellow light\n”);
        break;
case green:
        printf (“Green light\n”);
        break;
default:
        printf (“Other light\n”);
}
```

Функция вывода на экран

```
#include <stdio.h>
int main (void)
{
        int a = 28;
        double b = 12.52345436
        printf (“a(dec)=%d, a(hex)=%x, a(oct)=%o\n”,
                a, a, a);
        printf (“%s%f\n”, “b=”, b);
        return 0;
}
```

```
a(dec)=28, a(hex)=1c, a(oct)=34
b=12.52345
```

Спецификации преобразования в printf

%d - int (dec)
%x - int (hex)
%o - int (oct)
%u - unsigned int (dec)
%hd - short
%ld - long
%f - float ([-]ddd.ddd)
%e - float ([-]d.ddde+/-ddd)
%g - %f or %e
%lg - double
%Lg - long double

%c - char (symbol)
%s - строка