АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ

Филипп Рухович
Дмитрий Беляков

ЦЕЛИ КУРСА

- Курс находится на стыке математики и программирования

- После изучения курса Вы должны знать все базовые алгоритмы, а также уметь применять их для решения различных задач

- Кроме этого, в процессе обучения алгоритмам Вы узнаете, как их реализовывать на языке программирования C++

ОТЧЕТНОСТЬ ЗА КУРС

- 3-4 больших задания в течение семестра (на реализацию)
- На каждом семинаре ~5-6 задач
- В конце семестра - дифф. зачет

ВВЕДЕНИЕ В C++

- Эволюционное развитие языка C
- C++ совместим с C
- C++ более «близок к поставленной задаче»

HELLO, WORLD!

```
#include <iostream>

int main(int argc, char** argv)
{
  std::cout << "Hello, world!" << std::endl;
  return 0;
}
```

ПЕРЕМЕННЫЕ

- Каждое имя и каждое выражение обязано иметь тип `int index;`

- Тип определяет операции, которые могут быть применимы

ТИПЫ ДАННЫХ

- char
- short

- представляют целые числа

- int
- long

ТИПЫ ДАННЫХ

- float
- double
  - представляют числа с плавающей точкой
- long double
- bool - логическое значение (true или false)

МОДИФИКАТОРЫ ТИПА

- signed
- unsigned
- short (short int)
- long (long int)

РАЗМЕР ТИПОВ ДАННЫХ

```
#include <iostream>

int main()
{
   std::cout << std::numeric_limits<int>.max() << std::endl;
   return 0;
}
```

НЕЯВНОЕ ПРЕОБРАЗОВАНИЕ ТИПОВ

```cpp
#include <iostream>

int main()
{
    long a = 10e10;
    int b = a;
    std::cout << a << " " << b << std::endl;
    return 0;
}
```

ОПЕРАЦИИ СО ВСТРОЕННЫМИ ТИПАМИ ДАННЫХ

- Присваивание (=)
- Арифметические (+, -, *, /, %, ++, - -, +=, -=, %=)
- Отношений (==, !=, <, <=, >, >=)
- Логические (&&, ||, !)

ОБЛАСТИ ВИДИМОСТИ

- Глобальная область видимости - самая внешняя часть, доступна везде.

- Локальная область – это часть исходного текста программы, содержащаяся в определении функции (или в блоке).

- Имя должно обозначать одну сущность внутри одной области видимости.

ПРОИЗВОДНЫЕ ТИПЫ

- * - указатель (int *i)
- & - ссылка (int &i)
- [] - массив (int i[10])

УКАЗАТЕЛИ

- Для типа T переменная типа T* может хранить адрес объекта данного типа
- * - операция разыменования указателя (*p)
- -> - обращение к объекту (p->smth())
- Арифметика указателей: p = p + 1; ++p;

МАССИВЫ

- float v[3]
- float v[] = {1, 2, 3}; v[0] == 1; v[2] == 3
- float s[] = «hello, world»;
- Имя массива можно использовать как указатель на его первый элемент

ССЫЛКИ

- Ссылки – особый тип данных, являющийся скрытой формой указателя, который при использовании автоматически разыменовывается. Ссылка может быть объявлена как другим именем, так и как псевдоним переменной, на которую ссылается.

```
int a = 1;
int& b = a;
```

ФУНКЦИИ

```
T1 someFunc(T2 obj1, T3 obj2, ...)
{
   ...
}
```

- помогают организовать программу в виде совокупности независимых частей

- функция main - точка входа в программу

- вызов функции:

```
T2 o1 = ...;
T3 o2 = ...;

T1 returnedValue = someFunc(o1, o2);
```

ПАРАМЕТРЫ ФУНКЦИИ

- Значение параметров можно передавать: по значению, по указателю и по ссылке.

```
void someFunc(int a, int* b, int& c);
```

- Параметру можно задать значение по умолчанию

```
int calculate(int a, int b = 2);
```

- Возврат значения: ключевое слово return

КЛЮЧЕВОЕ СЛОВО CONST

- применительно к значениям

```
const int pi = 3.14;
```

- применительно к ссылкам

```
const int& pi_val = pi;
```

- применительно к указателям

```
int *const p1;
int const* p2;
const int* p3;
const int* const p4;
```

УСЛОВНЫЕ ОПЕРАТОРЫ

- if

```
if (smth) operator_1; else operator_2;
```

```
if (smth) { operator_1;  operator_2; … }
else { operator_1; operator_2; … }
```

- switch

```
switch (variable)
{
case value_1: operators_1; break;
case value_2: operators_2; break;
…
case value_n: operators_n; break;
default: operators; break;
}
```

ЦИКЛЫ

- for

```
for (int i = 0, n = 10; i < n; ++i) {
...
}
```

- while

```
while (i < n) {
...
}
```

- do while

```
do {
...
} while (i < n);
```

ВВОД-ВЫВОД (C-STYLE)

- scanf

```
char str[80];
int i;
std::scanf("%s%d", str, &i);
```

- printf

```
char str[] = "Hello, user";
int i = 1;
std::scanf("%s%d", str, i);
```

ВВОД-ВЫВОД (C++-STYLE)

- std::cin

```
int i;
double j;
std::cin >> i >> j;
```

- std::cout

```
int i = 1;
double j = 3.14;
std::cin << i << j;
```

#INCLUDE

Указывает препроцессору, что содержимое заданного файла необходимо обработать так, как если бы оно находилось в исходной программе в той точке, в которой располагается эта директива.

```
#include <iostream>

#include "local_file.h"
```