Программирование на C++

Минцифры России

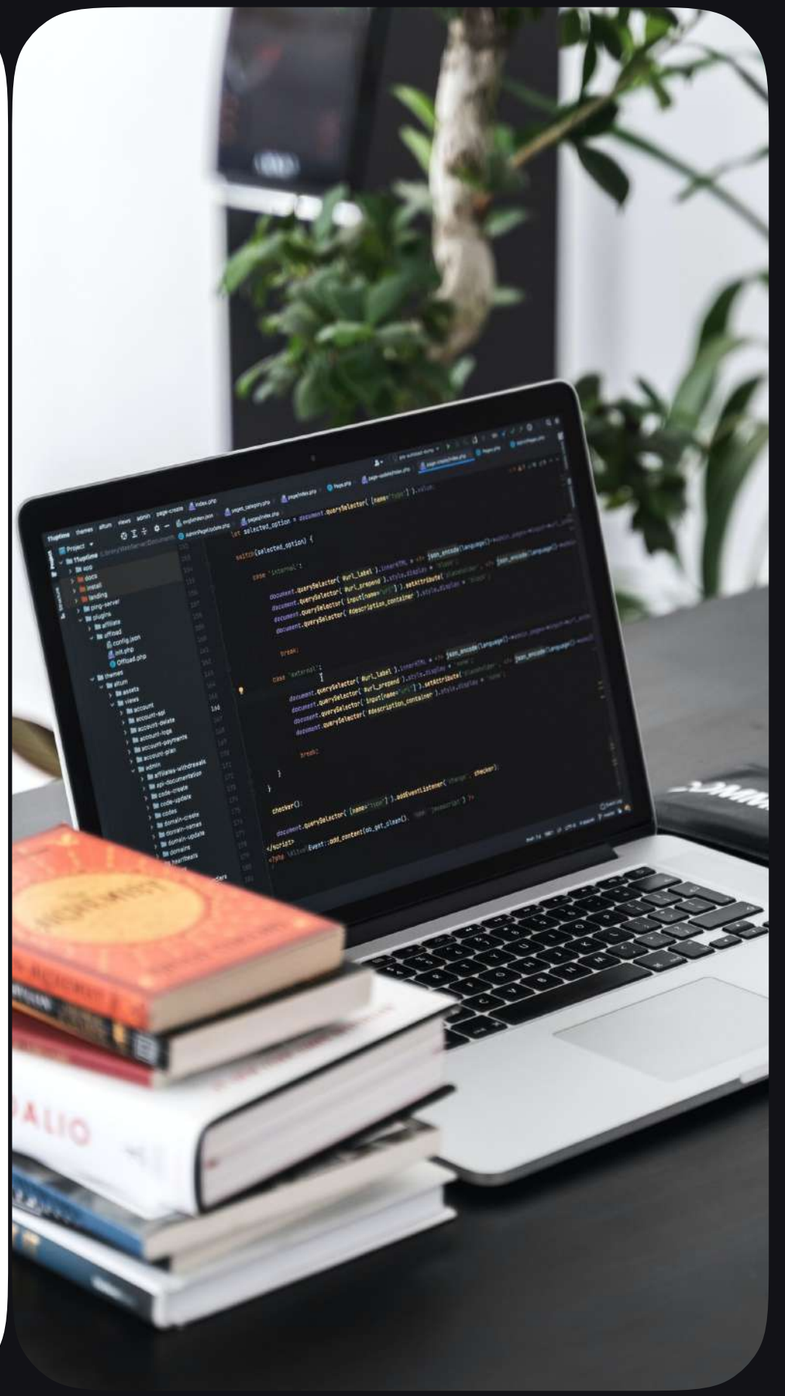
UCHi DOMA

20.35 УНИВЕРСИТЕТ

Урок 1 Модуль 3

C и C++

Полезные материалы



Цели урока

- изучить отличия C и C++
- отработать на практике составление алгоритмов с вводом/выводом, выделением динамической памяти на C++

Отличия C и C++

	C	C++
Появление	Разработал Деннис Ричи в 1969г. в Лабаратории Белла	Разработал Бьярне Страуструп в 1979г
Класс языка	Процедурный	Объектно-ориентированный, мультипарадигмальный, процедурный, функциональный, обобщённый
ООП	Поскольку C не поддерживает концепцию ООП, то он не поддерживает полиморфизм, инкапсуляцию и наследование	C++ поддерживает полиморфизм, инкапсуляцию и наследование, т. к. является объектно-ориентированным языком
Особенности	Не поддерживает перегрузку функций и операторов	C++ поддерживает как и перегрузку функций, так и перегрузку операторов, а также пространства имен и ссылки, обработка исключений, богатая библиотека

Отличия C и C++

	C	C++
Ключевые слова	32 ключевых слова	52 ключевых слова
Функции	Не поддерживаются в структуре, не могут быть «друзьями» и виртуальными	Поддерживаются в структуре, могут быть «друзьями» и виртуальными
Память	Функции **malloc()** и **calloc()** для динамического выделения памяти, а также **free()** для освобождения	Для этих же операций используются операторы **new** и **delete**
Вход/Выход	Используются **scanf()** и **printf()**	Используются **cout** и **cin**

Поточный ввод-вывод в C++

- В C++, как и в C, нет встроенных в язык средств ввода-вывода
- В C для этих целей используется библиотека **stdio.h.**
- В C++ разработана новая библиотека ввода-вывода **iostream**, использующая концепцию объектно-ориентированного программирования:

```
#include <iostream>
```

Поточный ввод-вывод в C++

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)

Поточный ввод-вывод в C++

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

- >> получить из входного потока
- << поместить в выходной поток

Поточный вывод в C++

cout << значение;

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

```
cout << s;
```

Возможно многократное назначение потоков:

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

Пример поточного вывода

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5  int a=10,b=2,sum=0;
6  sum=a+b;
7  cout << a<<" + "<< b<< " = " << sum;
8  return 0;
9  }
```

Результат работы программы:

```
10 + 2 = 12
```

Поточный ввод информации в C++

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

Возможно многократное назначение потоков:

cin >> переменная1 >> переменная2 >>...>> переменнаяn;

Пример поточного ввода информации

```
1   #include <iostream>
2   using namespace std;
3   int main()
4   {
5   int a,b,sum=0;
6   cin>>a>>b;
7   sum=a+b;
8   cout << a<<" + "<< b<< " = " << sum;
9   return 0;
10  }
```

Результат работы программы:

```
2
3
2 + 3 = 5
```

или

```
23
2 + 3 = 5
```

Ввод символьных строк

По умолчанию потоковый ввод **cin** вводит строку до пробела, символа табуляции или перевода строки

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5    char s[80];
6    cin >> s;
7    cout << s << endl;
8    return 0;
9  }
```

Результат работы программы:

```
Привет, пользователь
Привет,
```

Ввод символьных строк

Для ввода текста до символа перевода строки используется манипулятор потока **getline()**

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5    char s[80];
6    cin.getline(s, 80);
7    cout << s << endl;
8    return 0;
9  }
```

Результат работы программы:

```
Привет, пользователь
Привет, пользователь
```

Манипуляторы потока

Функцию-манипулятор потока можно включать
в операции помещения в поток и извлечения из потока (<<, >>)

Манипулятор	Описание
endl	Помещение в выходной поток символа конца строки **'\n'**
dec	Установка основания 10-ой системы счисления
oct	Установка основания 8-ой системы счисления
hex	Установка основания 16-ой системы счисления
setbase	Вывод базовой системы счисления
width(ширина)	Устанавливает ширину поля вывода
fill('символ')	Заполняет пустые знакоместа значением символа

Манипуляторы потока

Манипулятор	Описание
precision(точность)	Устанавливает количество значащих цифр в числе (или после запятой) в зависимости от использования **fixed**
fixed	Показывает, что установленная точность относится к количеству знаков после запятой
showpos	Показывает знак + для положительных чисел
scientific	Выводит число в экспоненциальной форме
get()	Ожидает ввода символа
getline(указатель, количество)	Ожидает ввода строки символов. Максимальное количество символов ограничено полем **количество**

Программы на Си в C++

- ✓ Код, написанный на Си, будет работать в C++
- ✓ C++ обратно совместим с C

Пример программы на C и C++

Программа ввода-вывода значения переменной

Программа на Си

```c
#include <stdio.h>
int main()
{
int n;
  printf("Введите n:");
  scanf("%d", &n);
  printf("Значение n равно: %d\n", n);
  return 0;
}
```

Программа на C++

```cpp
#include <iostream>
using namespace std;
int main()
{
  int n;
  cout << "Введите n:";
  cin >> n;
  cout << "Значение n равно: " << n << endl;
  return 0;
}
```

Динамическое выделение памяти в C++

В Си работать с динамической памятью можно при помощи соответствующих функций распределения памяти (**calloc**, **malloc**, **free**), для чего необходимо подключить библиотеку **malloc.h**

Динамическое выделение памяти в C++

C++ использует новые методы работы с динамической памятью при помощи операторов **new** и **delete**:

- **new** — для выделения памяти
- **delete** — для освобождения памяти

Оператор **new** используется в следующих формах:

- **new тип;** — для переменных
- **new тип[размер];** — для массивов

Результатом выполнения операции **new** будет указатель на отведенную память, или исключение **std::bad_alloc** в случае ошибки.

Динамическое выделение памяти в C++

```
int *ptr_i;
double *ptr_d;
struct person *human;
…
ptr_i = new int;
ptr_d = new double[10];
human = new struct person;
```

Динамическое выделение памяти в C++

Память, отведенная в результате выполнения **new**, будет считаться распределенной до тех пор, пока не будет выполнена операция **delete.**

Освобождение памяти связано с тем, как выделялась память — для одного элемента или для нескольких. В соответствии с этим существует и две формы применения **delete**:

- **delete указатель;** — для одного элемента
- **delete[] указатель;** — для массивов

Освобождаться с помощью **delete** может только память, выделенная оператором **new**.

Динамическое выделение памяти в C++

```
int *ptr_i;
double *ptr_d;
struct person *human;
…
ptr_i = new int;
ptr_d = new double[10];
human = new struct person;

delete ptr_i;
delete[] ptr_d;
delete human;
```

Пример создания динамического массива

```
1   #include <iostream>
2   using namespace std;
3   int main()
4   {
5     int size;
6     int *mas;
7     cout << "Ввести размерность массива : ";
8     cin >> size;
9     mas = new int[size];
10    for (int i = 0; i<size; i++) {
11      cout << "mas[" << i << "] = ";
12      cin >> mas[i];
13    }
14    for (int i = 0; i<size; i++)
15      cout << mas[i] << " ";
16    delete[] mas;
17    return 0;
18  }
```

Результат работы программы:

```
Ввести размерность массива : 3
mass [0] = 1
mass [1] = 2
mass [2] = 3
1 2 3
```