9-11 классы

Программирование
на C++

Презентация занятия

Указатели.
Массивы.

4 занятие

КОД БУДУЩЕГО | Минцифры России | инжиниринум МГТУ им. Н.Э. Баумана | НАЦИОНАЛЬНЫЕ ПРОЕКТЫ РОССИИ ЦИФРОВАЯ ЭКОНОМИКА | 20.35 УНИВЕРСИТЕТ

2020

Программирование
на C++

Теоретическая часть

Указатели.
Массивы.

4 занятие

инжиниум®
МГТУ им. Н.Э. Баумана

2020

Тема: Указатели. Массивы.

Оператор адреса (&)

При выполнении инициализации переменной, ей автоматически присваивается свободный адрес памяти, и, любое значение, которое мы присваиваем переменной, сохраняется в этом адресе памяти

```cpp
#include <iostream>

int main()
{
    int a = 7;

    // выводим значение переменной a
    std::cout << a << '\n';

    // выводим адрес памяти переменной a
    std::cout << &a << '\n';

    return 0;
}
```

Тема: Указатели. Массивы.

Оператор разыменования (*) позволяет получить значение по указанному адресу:

```cpp
#include <iostream>

int main()
{
    int a = 7;
    // Выводим значение переменной a
    std::cout << a << std::endl;

    // Выводим адрес переменной a
    std::cout << &a << std::endl;

    // Выводим значение ячейки памяти переменной a
    std::cout << *(&a) << std::endl;

    return 0;
}
```

Тема: Указатели. Массивы.

Указатель — это переменная, значением которой является адрес (ячейка) памяти. Указатели объявляются точно так же, как и обычные переменные, только со звёздочкой между типом данных и идентификатором:

```
// указатель на значение типа int
int *iPtr;
// указатель на значение типа double
double *dPtr;

// корректный синтаксис (допустимый, но не желателен)
int* iPtr3;
// корректный синтаксис (не делайте так)
int * iPtr4;
// объявляем два указателя для переменных типа int
int *iPtr5, *iPtr6;
```

Тема: Указатели. Массивы.

Поскольку указатели содержат только адреса, то при присваивании указателю значения — это значение должно быть адресом. Для получения адреса переменной используется оператор адреса:

```
int value = 5;
// инициализируем ptr адресом значения переменной
int *ptr = &value;
```

Вот почему указатели имеют такое имя: ptr содержит адрес значения переменной value, и, можно сказать, ptr *указывает* на это значение.

Тема: Указатели. Массивы.

```cpp
#include <iostream>

int main()
{
    int value = 5;
    // инициализируем ptr адресом значения переменной
    int *ptr = &value;

    // выводим адрес значения переменной value
    std::cout << &value << std::endl;
    // выводим адрес, который хранит ptr
    std::cout << ptr << std::endl;

    return 0;
}
```

Тема: Указатели. Массивы.

```
int iValue = 7;
double dValue = 9.0;

int *iPtr = &iValue;
double *dPtr = &dValue;

// неправильно: указатель типа int не может указывать
на адрес переменной типа double
iPtr = &dValue;
// неправильно: указатель типа double не может
указывать на адрес переменной типа int
dPtr = &iValue;
```

Тема: Указатели. Массивы.

Следующее не является допустимым:

```
int *ptr = 7;
```

Это связано с тем, что указатели могут содержать только адреса, а целочисленный **литерал 7 не имеет адреса памяти.**

C++ также не позволит вам напрямую присваивать адреса памяти указателю:

```
// не ок: рассматривается как присваивание
целочисленного литерала
double *dPtr = 0x0012FF7C;
```

Тема: Указатели. Массивы.

Оператор адреса возвращает указатель

Стоит отметить, что оператор адреса (&) не возвращает адрес своего операнда в качестве литерала. Вместо этого он возвращает указатель, содержащий адрес операнда, тип которого получен из аргумента (например, адрес переменной типа int передаётся как адрес указателя на значение типа int):

```
#include <iostream>
#include <typeinfo>

int main()
{
    int x(4);
    std::cout << typeid(&x).name();

    return 0;
}
```

Результат выполнения программы выше:

```
int *
```

Тема: Указатели. Массивы.

Разыменование указателей

Как только у нас есть указатель, указывающий на что-либо, мы можем его разыменовать, чтобы получить значение, на которое он указывает. Разыменованный указатель — это содержимое ячейки памяти, на которую он указывает:

```cpp
#include <iostream>

int main()
{
    int value = 5;
    std::cout << &value << std::endl; // выводим адрес value
    std::cout << value << std::endl; // выводим содержимое value

    int *ptr = &value; // ptr указывает на value
    std::cout << ptr << std::endl; // выводим адрес, который хранится в ptr, т.е. &value
    std::cout << *ptr << std::endl; // разыменовываем ptr (получаем значение на которое указывает ptr)

    return 0;
}
```

Результат:

```
0034FD90
5
0034FD90
5
```

Тема: Указатели. Массивы.

Вот почему указатели должны иметь тип данных. Без типа указатель не знал бы, как интерпретировать содержимое, на которое он указывает (при разыменовании). Также, поэтому и должны совпадать тип указателя с типом переменной. Если они не совпадают, то указатель при разыменовании может неправильно интерпретировать биты (например, вместо типа double использовать тип int).

Одному указателю можно присваивать разные значения:

```
int value1 = 5;
int value2 = 7;

int *ptr;

ptr = &value1; // ptr указывает на value1
std::cout << *ptr; // выведется 5

ptr = &value2; // ptr теперь указывает на value2
std::cout << *ptr; // выведется 7
```

Тема: Указатели. Массивы.

Когда адрес значения переменной присвоен указателю, то выполняется следующее:

- ptr – это то же самле, что и &value;
- *ptr – обрабатывается так же, как и value.

Поскольку *ptr обрабатывается так же, как и value, то мы можем присваивать ему значения так, как если бы это была бы обычная переменная, например:

```
int value = 5;
int *ptr = &value; // ptr указывает на value

*ptr = 7; // *ptr - это то же самое, что и value,
которому мы присвоили значение 7
std::cout << value; // выведется 7
```

Тема: Указатели. Массивы.

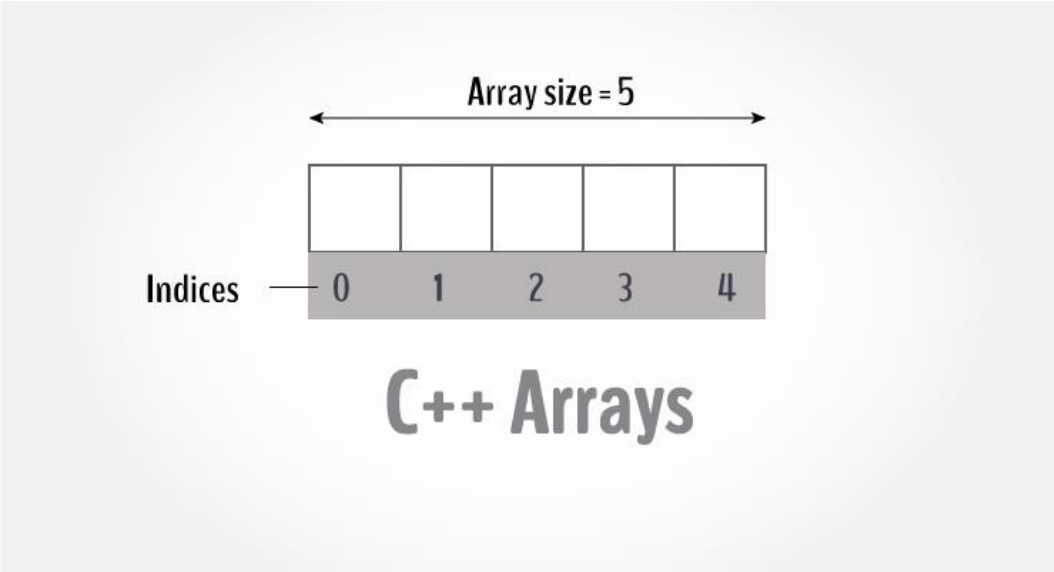
```cpp
#include <iostream>

int main()
{
    int *p; // создаём неинициализированный указатель
    (содержимым которого является мусор)

    std::cout << *p; // разыменовываем указатель с мусором

    return 0;
}
```

Тема: Указатели. Массивы.

Массивы

Массивы используются для хранения коллекций информации, но может быть полезным представлять массив как коллекцию переменных одинакового типа. Вместо объявления множества переменных и хранения в них индивидуальных значений вы можете объявить один массив для хранения всех этих значений. При объявлении массива укажите тип его элементов, а также и количество хранимых им элементов.

Например:



Тема: Указатели. Массивы.

В примере, переменная a объявлена как массив пяти значений целочисленного типа [указанных в квадратных скобках]

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

Тема: Указатели. Массивы.

Массивы

Вы можете инициализировать массив указав все его значения:

Значения представлены в форме списка, разделены запятыми, закрыты внутри {фигурных скобок}.

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       int b[5] = {11, 45, 62, 70, 88};
7
8       return (0);
9   }
```

Количество значений между фигурных скобок { } не должно превышать число элементов, объявленных в квадратных скобках [ ].

Тема: Указатели. Массивы.

Инициализация массивов

Если вы опустите размер массива, то будет создан массив достаточно большого размера для хранения инициализации.
Например:

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       int b[] = {11, 45, 62, 70, 88};
7
8       return (0);
9   }
```

Таким образом создается массив идентичный созданному в прошлом примере.

Тема: Указатели. Массивы.

Индексация

Каждый элемент, или член массива имеет свой индекс, который отмечает конкретную позицию каждого элемента.
Первый элемент массива имеет индекс равный 0, второй имеет индекс равный 1.
Для массива b, который мы объявили выше:

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       int b[] = {11, 45, 62, 70, 88};
7
8       return (0);
9   }
```

11	45	62	70	88
[0]	[1]	[2]	[3]	[4]

Тема: Указатели. Массивы.

Индексация

Для доступа к элементам массива, проиндексируйте имя массива путем подстановки индекса элемента в квадратные скобки после имени массива. Например:

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       int b[] = {11, 45, 62, 70, 88};
7
8       cout << b[0] << endl;
9       // Outputs 11
10
11      cout<< b[3] << endl;
12      // Outputs 70
13
14      return (0);
15  }
```

Тема: Указатели. Массивы.

Получение доступа к элементам массива

Индексы могут быть также использованы для присвоения нового значения элементу.

В этой программе присваивается число 100 третьему элементу массива.

В этой программе присваивается число 100 третьему элементу массива.

```
1   #include <iostream>
2   using namespace std;
3
4   int main()
5   {
6       int b[] = {11, 45, 62, 70, 88};
7
8       cout << b[2] << endl;
9       // Outputs 62
10
11      b[2] = 100;
12
13      cout<< b[2] << endl;
14      // Outputs 100
15
16      return (0);
17  }
```

Программирование
на C++

Практическая часть

Указатели.
Массивы.

4 занятие

инжиниум®
МГТУ им. Н.Э. Баумана

2020

Тема: Указатели. Массивы.

```
int value = 45;
int *ptr = &value; // объявляем указатель и
инициализируем его адресом переменной value
*ptr = &value; // присваиваем адрес value для ptr
```

Тема: Указатели. Массивы.

Будут ли одинаковы все три адреса в памяти, отображаемые следующей программой? Объясните, что происходит в этом коде.

```
#include <iostream>
using namespace std;
int main()
{
    int a = 10;
    int& b = a;
    int* c = &b;
    cout <<&a << endl;
    cout <<&b << endl;
    cout <<&(*c) << endl;
    return 0;
}
```