

Информатика: семестр 3

Программирование на C++

Конспекты лекций

Лектор: Хирьянов Т.Ф.

2 января 2018 г.

Оглавление

4	Динамическая память. Продолжение.	3
4.1	Динамические массивы	3
4.2	Структуры в C++	4
4.3	Массивы и функции	6
4.4	Двумерный массив и передача его функции	7
4.5	Статические массивы	8
4.6	Динамические массивы	8

Лекция 4

Динамическая память. Продолжение.

4.1 Динамические массивы

Как было сказано, массивов в C++ нет. Повторим объявление указателей:

Код 4.1: Пример объявления указателей

```
1 double A[100]; // has type 'const double*'
2 double* p = A;
3 //or
4 double* p1 = &A[5]
5 //equals to
6 double* p1 = A + 5
7
8 double* p = new double(5.);
```

В строке 4 мы получаем указатель на **A[5]**. В строке 8 мы создаем число, да и еще присваиваем ему значение.

Код 4.2: Объявление массива через **new**

```
1 double* p = new double[50];
2
3 for(int k = 0; k < 50; k++)
4     p[k] = k*k;
5
6 delete[] p;
```

При объявлении массива через **new** нужно обязательно его удалять через **delete[]** – оператор удаления.

4.2 Структуры в C++

В C++ существует такая вещь, как структура – набор переменных под одной крышкой.

Код 4.3: Простейшая структура

```
1 #include<iostream>
2
3 struct Student {
4     int8_t age;
5     int16_t group;
6     std::string name;
7 };
8
9 void student_print(Student x) {
10     std::cout << x.name << " " << x.age
11     << " " << x.group << "\n";
12 }
13
14 int main()
15 {
16     Student vasya = {16, 649, "Vasilii Ivanov"};
17     Student masha = {18, 649, "Maria Vasilyeva"};
18
19     Student *p = nullptr;
20
21     p = &vasya;
22
23     std::cout << (*p).name << "\n";
24     // This code is equal to
25     std::cout << p -> name << "\n";
26     student_print(masha);
27
28     return 0;
29 }
30
31 void studentPrint1(Student *x) {
32     std::cout << x->name << " " << x->age
33     << " " << x->group << "\n";
34 }
35
36 void studentPrint2(const Student &x) {
37     std::cout << x.name << " " << x.age
38     << " " << x.group << "\n";
39 }
```

Обратите внимание на строки **23** и **25**: такое сокращение еще часто будет встречаться (для указателей вместо "." используется "–>"). Метод для распечатки мы вынесли

отдельно для удобства.

Подход к распечатке в самой первой функции имеет минус: при вызове функции мы создаем локальную копию `Student`. Метод **studentPrint1** лишен этого – там передается по ссылке. Но возникает проблема, ведь мы внутри функции случайно можем изменить переменную `x` и этот результат будет виден извне. Решение дает передача по константной ссылке, что сделано в **studentPrint2**.

Существует еще одно решение задачи, в котором мы сначала создаем объект `Student` в памяти, получаем на него указатель, а затем через него меняем свойства объекта:

Код 4.4: Еще одно решение со структурой

```
1 #include<iostream>
2
3 struct Student {
4     int8_t age;
5     int16_t group;
6     std::string name;
7 };
8
9 void studentPrint(const Student &x);
10
11 int main()
12 {
13     Student *p_student = new Student;
14     p_student->age = 17;
15     p_student->group = 649;
16     p_student->name = "Vasya Ivanov";
17
18     studentPrint(*p_student);
19     return 0;
20 }
21
22 void studentPrint(const Student &x) {
23     std::cout << x.name << " " << x.age
24     << " " << x.group << "\n";
25 }
```

Из структур можно составлять массивы, что неудивительно, учитывая, что массив - это просто несколько переменных. В качестве примера можно привести следующий код:

Код 4.5: Массив из структур

```
1 #include<iostream>
2
3 struct Student {
4     int8_t age;      // Here age is not shown, but age is kept
5     int16_t group;
6     std::string name;
7 };
8
9 void studentPrint(const Student &x);
10
11 int group_max_size = 10;
12 int main()
13 {
14     Student *my_group = new Student[group_max_size];
15     for(int k = 0; k < group_max_size; k++)
16     {
17         my_group[k].age = 17 + k;
18         my_group[k].group = 600 + k;
19         my_group[k].name = "Vasya Ivanov";
20     }
21
22     for(int k = 0; k < group_max_size; k++)
23     {
24         studentPrint(my_group[k]);
25     }
26     return 0;
27 }
28
29 void studentPrint(const Student &x) {
30     std::cout << x.name << " " << x.age
31     << " " << x.group << "\n";
32 }
```

4.3 Массивы и функции

В функциях можно менять массивы, но их нельзя возвращать. Код внизу – пример изменения массива внутри функции.

Код 4.6: Изменение массива внутри функции

```
1 #include<iostream>
2
3 void squarer(int* B, const int& B_size)
4 {
5     for(int k = 0; k < B_size; k++) {
6         B[k] = B[k] * B[k];
7     }
8 }
9
10 int main()
11 {
12     int A[5];
13     squarer(A, 5);
14     return 0;
15 }
```

4.4 Двумерный массив и передача его функции

Двумерные массивы работают так же, как в Питоне. Тем не менее, у них есть некоторые особенности, связанные с их способом представления в памяти (помним, что массив по факту лишь набор переменных). Одна из этих особенностей видна в коде ниже, где мы передаем двумерный массив в функцию:

Код 4.7: Передача массива в функцию

```
1 #include<iostream>
2
3 //void squarer(A);
4
5 void printer(double A[][4])
6 {
7     for(int k = 0; k < 3; k++)
8     {
9         for(int j = 0; j < 4; j++)
10             std::cout << A[k][j] << '\t';
11         std::cout << '\n';
12     }
13 }
14
15 int main()
16 {
17     double A[3][4];
18     for(int k = 0; k < 3; k++)
19         for(int j = 0; j < 4; j++)
20             A[k][j] = 1 + k*4 + j;
```

```
21  
22     printer(A);  
23  
24     return 0;  
25 }
```

Обратите внимание: в объявлении функции **printer** мы обязательно указываем длину (вторая квадратная скобка).

4.5 Статические массивы

Код 4.8: Указание на элемент массива

```
1 A[i][j] == *(A+i*y+j)
```

В коде выше видно, что для прохода по элементам массива в функции необходимо заранее знать его длину. Если посмотреть на выражение 4.8, то станет видно, что эта длина уходит в инструкции прохода по массиву еще на этапе компиляции, поэтому ее и нужно было передавать заранее, как это сделано в 4.7. Но можно ли сделать функцию **printer** без явного указания длины массива? Ответ дают динамические массивы.

4.6 Динамические массивы

Динамические массивы — это массивы, хранящиеся в так называемой динамической памяти (*heap*). Динамические массивы объявлять несколько сложнее: так, для двумерного случая необходимо делать указатель на область памяти с указателями, каждый указатель в которой, в свою очередь, будет указывать на область памяти с нашими объектами (к примеру, **int**). Звучит не очень, однако пример 4.10 может немного прояснить этот момент.

Код 4.9: Объявление динамического массива

```
1 #include<iostream>  
2  
3 //void squarer(A);  
4  
5 // TODO: func must take width and height, fix it  
6 void printer(double A[][4])  
7 {  
8     for(int k = 0; k < 3; k++)  
9     {  
10         for(int j = 0; j < 4; j++)  
11             std::cout << A[k][j] << '\t';
```



```
12         std::cout << '\n';
13     }
14 }
15
16 int main()
17 {
18     int width = 4, height = 3;
19     double **A = nullptr;
20     A = new double*[height];
21     for(int k = 0; k < height; k++)
22         A[k] = new double[width];
23
24     for(int k = 0; k < height; k++)
25         for(int j = 0; j < width; j++)
26             A[k][j] = 1 + k*width + j;
27
28     printer(A);
29
30     for(int k = 0; k < height; k++)
31         delete[] A[k];
32     delete A;
33
34     return 0;
35 }
```

A – это массив из указателей типа **double***. Каждый элемент этого массива - массив размера **width**. Таким образом, **A** есть двумерный массив. Обращение к его элементам происходит так же, как и при обычном объявлении массива, что видно в строке **26**. Мы сделали массив, распечатали его, а затем удалили, чтобы освободить память. При этом следует помнить такой трюк:

Код 4.10: Порядок разыменования

```
1 A[k][j] == (*(A + k) + j)
```

Исходные коды всех программ можно найти [здесь](#)