Информатика: семестр 3

Программирование на C++

Конспекты лекций

Лектор: Хирьянов Т.Ф.

24 октября 2017 г.

Оглавление

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 крышей.

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

```
#include<iostream>

struct Student {
    int8_t age;
    int16_t group;
    std::string name;
};

void studentPrint(const Student &x);

int main()
{
    Student *p_student = new Student;
    p_student->age = 17;
    p_student->group = 649;
    p_student->name = "Vasya Ivanov";

    studentPrint(*p_student);
    return 0;
}

void studentPrint(const Student &x) {
    std::cout << x.name << " " << x.age
    << " " << x.group << "\n";
}
```

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

```
#include<iostream>

struct Student {
    int8_t age;
    int16_t group;
    std::string name;
};

void studentPrint(const Student &x);

int group_max_size = 10;
int main()
{
    Student *my_group = new Student[group_max_size];
    for(int k = 0; k < group_max_size; k++)
    {
        my_group[k].age = 17 + k;
        my_group[k].group = 600 + k;
        my_group[k].name = "Vasya Ivanov";
    }

    for(int k = 0; k < group_max_size; k++)
    {
        studentPrint(my_group[k]);
    }
    return 0;
}

void studentPrint(const Student &x) {
    std::cout << x.name << " " << x.age
    << " " << x.group << "\n";
}
```

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

```
#include<iostream>

void squarer(int* B, const int& B_size)
{
    for(int k = 0; k < B_size; k++) {
        B[k] = B[k] * B[k];
    }
}

int main()
{
    int A[5];
    squarer(A, 5);
    return 0;
}
```

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

```
#include<iostream>

//void squarer(A);

void printer(double A[][4])
{
    for(int k = 0;k < 3; k++)
    {
        for(int j = 0; j < 4; j++)
            std::cout << A[k][j] << '\t';
        std::cout << '\n';
    }
}

int main()
{
    double A[3][4];
    for(int k = 0;k < 3; k++)
        for(int j = 0; j < 4; j++)
            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 прохождения 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 widh 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