Информатика: семестр 3

Программирование на C++

Конспекты лекций

Лектор: Хирьянов Т.Ф.

24 октября 2017 г.

Оглавление

Лекция 5

Строки и файлы в C++

5.1 Хранение строк

Строки делятся на *C-строки* и *ANSI-строки*. В разных языках строки реализованы по-разному. Так, в Pascal они хранятся как *ANSI-строки*: в первой ячейке памяти записана длина строки, а затем последовательно идут символы строки. Главный минус такого подхода — ограниченность длины строки (255 символов). Этот недостаток отсутствует в *C-строках*.

В *C-строках* используется следующая схема хранения: в памяти просто идет последовательность символов и завершается **нулем** (типа **int**, не "0"). Так, строка "Hello" будет записана как последовательность 'H', 'e', 'l', 'l', 'o', 0x0 (строки такого типа называются *zero-ending string*). НО! Если мы сделаем обращение и изменим какой-либо символ на 0 (типа целочисленное), то строка поделится на две, ведь теперь у нее два символа окончания. Это — один из минусов такого подхода.

5.2 Операции со строками

Конкатенация строки можно делать с помощью стандартной библиотеки **cstring**.

Код 5.1: Работа со строками

```
1  #include <cstring>
2  #include <iostream>
3
4  using namespace std;
5
6  int main()
7  {
8   char s1[10000], s2[1000];
9   s1[0] = 0;
10  std::cin.getline(s2, 1000);
11  strcat(s1, s2); // concatanate
```

```
12 }
```

В этом коде мы считываем строки, а затем их конкатенируем. Но Т.Ф. не любит эту библиотеку. Дело в том, что **strcat** сначала проходит и за линейное время находит ноль в конце строки, дабы узнать, где ее конец, а затем уже объединяет обе. Если такой код засунуть в цикл, то (из свойств арифметической прогрессии) мы подарим нашему коду квадратичную асимптотику.

В MacOS сделано несколько интереснее: используется как число, равное длине строки, в ее начале, так и символ окончания в конце (комбинированный подход).

При работе со строками следует быть осторожными: при операции `+=` можно уйти за аллоцированную (выделенную) память для данной строки и стереть ее содержимое. Кстати, в C++ строка — это объект. Для нас сейчас это значит, что операции, которые нам кажутся простыми (наподобие `+=`), на деле такими не являются — они реализованы по уму. К тому же, у строк есть свои методы и поля. К примеру, метод **s.size()** вернет длину строки за $O(1)$. У строк автоматическая работа с памяти — язык автоматически выделяет столько памяти, сколько нужно.

Т.Ф. рекомендует пользоваться C++ – строками вместо C–строк.

5.3 C++ – строки

В C++ строки входят в окружение **std**.

Код 5.2: Объявление строки в C++

```
1 #include <string>
2 #include <iostream>
3
4 int main()
5 {
6  std::string s("Hello");
7  for(int k=0; k < s.size(); k++)
8   std::cout << s[k] << std::endl;
9 }
```

В Python строки кодируются стандартом UTF-8. В таком подходе память используется в разных количествах для разных символов, поэтому при вставке символа в середину строки возникают проблемы переноса (одна из причин, почему в Python строки неизменяемы). C++ лишен такого недостатка и здесь строку можно менять посимвольно.

C++ не разделяет типы символа и *кода* символа. Символ = код символа. Вообще, подробности можно изучить здесь.

5.4 Считывание строки

Существует несколько способов читать строки:

Код 5.3: Считывание строки

```
while(cin >> s) // (cin >> s) returns true or false
{
 cont << s << '*';
}

// Another way
getline(cin, s)
```

Обратите внимание: **cin** ≫ **s** возвращает **true** или **false**, в зависимости от того, закончился ли ввод.

Другой способ заключался в методе **getline**, которому на вход поступает поток ввода (см. ниже) и строка s, куда будет записано. Различие следующее: **cin** читает до символа пробела, тогда как **getline** читает до символа переноса строки.

5.5 Файловые потоки

Стандартная библиотека для работы с потоками — это **fstream**. Она дает три потока для работы: **ifstream** для ввода и **ofstream** для вывода и **fstrim** для обоих вариантов.

Код 5.4: Пример работы с потоками

```
#include <fstream>
#include <iostream>

using namespace std;

void read_array(int *A, int A_size, string f_name)
{
 ifstream fin(f_name); // input stream based on file
 for(int k=0; k < A_size; k++)
  fin >> A[k];
}
```

В коде выше было бы неплохо закрыть файл после окончания работы с ним. Более того, операционная система может дать отказ, если мы затребуем открытие слишком большого количества файлов. Однако C++ умный, он закроет файл после окончания существование объекта (в ходе выполнения деструктора, о них будет сказано в следующих лекциях).

Во время открытия потока можно указать для него *режим*: только для чтения и т.п.

Код 5.5: Режимы потока

```
fstream f("1.txt", "rwt")
```

Подробнее о режимах можно почитать в документации.

5.6 Строковые потоки

Работа со строковыми потоками тоже проходит через стандартную библиотеку:

Код 5.6: Строковые потоки

```
#include <sstream>

using namespace std;

int main()
{
 std::stringstream t;
 t << "x = " << 7 << "." << true << std::endl; // any type is compatable
 string s;
 s = sout.str(); // cast stream to string
 return 0;
}
```

5.7 Односвязный список

Попробуем сделать односвязный список в C++ по аналогии с тем, что мы делали в Python.

Код 5.7: Реализация односвязного списка

```
#include <iostream>

using namespace std;

typedef std::string DataType;

struct tNode
{
    DataType data;
    tNode* next = nullptr;
};

tNode * insert_node(tNode *p_begin, DataType data);
```

```
14
15  int main()
16  {
17
18      tNode* p = new tNode;
19      p->data = "Hello";
20      // Sequently add nodes
21      p = insert_node(p, "World");
22      p = insert_node(p, "Reverse order");
23
24      while(p) {
25          std::cout << p->data << std::endl;
26          p = p->next;
27      }
28
29      return 0;
30  }
31
32  tNode * insert_node(tNode *p_begin, DataType data)
33  {
34      tNode *p = new tNode;
35
36      p->data = data;
37      p->next = p_begin;
38      return p;
39  }
```

1. **typedef** — это создание синонима типа **string**. Это сделано только для того, чтобы не писать каждый раз **std::string**, другого назначения у такой конструкции для нас нет.

2. **struct tNode** — объявление структуры ноды. В ней всего две переменные: свое значение и ссылка на следующую ноду.

3. **tNode * ...** — сигнатура функции *insert_node*. Ее мы опишем позже.

4. В методе **main** мы создаем ноду через **new**, затем сохраняем ссылку на созданный объект. После этого мы записываем туда какую-то информацию ("Hello"в примере), далее прикрепляем к ней другие ноды.

5. В цикле мы прогуливаемся по всем нодам и выводим их содержимое. Обратите внимание на переход к следующей ноде в последней строке цикла.

6. Функция **insert_node** — изюминка нашего кода. Ей передается указатель на нашу ноду. Внутри функции мы создаем новую ноду, заполняем в нее данные из **data**, а в качестве следующей ноды записываем ту, что мы передали функции.

Из этого получается, что мы добавляем ноды в обратном порядке, так как только что созданная нода встает впереди старой.

Исходные коды всех программ можно найти здесь