Строки текста как массивы и работа с файлами

Текстовые строки как массивы символов

- В языке C++ текстовая строка представляет собой набор символов, обязательно заканчивающийся нулевым символом (‘**\0**’) Текстовые строки представляются с помощью одномерных массивов символов.
- Для хранения текстовых строк из **m** символов необходимо определить одномерный массив символов с количеством элементов, равным

  m + 1:

```
const  m = 20;
char  Str [m + 1];
```

или

```
char  Str [21];
```

- Переменная **Str** может использоваться для хранения текстовых строк, содержащих не более **20** символов (максимум **20** обычных символов + нулевой символ **‘\0’**).
- Нулевой символ позволяет определить границу между содержащимся в строке текстом и неиспользованной частью строки.

Инициализация текстовых строк

- При определении строковых переменных их можно инициализировать конкретными значениями с помощью строковых литералов:

char S1 [15] = “This is text”;

char S2 [ ] = “Это пример текста”;

char S3 [20] = “”;

char S4 [5] = “Error”;

- Строка **S3** инициализируется “пустой” строкой **“”** и содержит один нулевой символ **‘\0’**.
- Попытка инициализировать строку **S4** словом **“Error”** закончится неудачей, так как в переменную **S4**, имеющую размер в **5** символов, можно записывать тексты с максимальным количеством символов не более **4**

Программное формирование строк

- Если строка формируется программно, необходимо в конец ее обязательно записать нулевой символ **‘\0’**:

```
char Str[21];
int i = 0;
for (char c = ‘A’; i < 6; i++, c++)
    Str [i] = c;
Str [i] = ‘\0’;              // “закрываем” строку нулевым символом
cout  <<  Str  <<  endl;   // на экран выведено: ABCDEF
```

- Если не добавить нулевой символ, то на экран будут выведены все 21 символов массива **Str**. Значения последних 14 символов предсказать невозможно.

Вывод текста на экран

Вывод текстовых строк на экран сложностей не вызывает:

char Str[21] = “Это пример текста”;

cout << setw(40) << right << Str << endl;

cout << “Это текстовый литерал.” << endl;

При выводе строк можно использовать форматирование (манипуляторы или функции потока вывода).

Непосредственное чтение текстовых строк из потока вывода

char Str[21];

cin >> Str; // Пусть введена строка **“Это пример текста”**

cout << Str << endl; // На экран будет выведено только слово **“Это”**

Такой способ чтения (с помощью оператора **>>**) обеспечивает ввод символов до первого пробельного символа (по словам), а не до конца строки. Остальные символы введенного с клавиатуры предложения остаются в потоке ввода и могут быть прочитаны из него следующими операторами >>.

Функции gets и gets_s

- Для того чтобы прочесть всю строку полностью, можно воспользоваться одной из функций **gets** или **gets_s** (для этого в программу должен быть включен заголовочный файл **<stdio.h>**):
- **const int N = 21;**
- **char Str [N];**
- **gets (Str);** **//** Пусть введена строка **“Это пример текста”**
- **// gets_s (Str, N);** Альтернативный вариант
- **cout << Str << endl; //** На экран будет выведено **“ Это пример текста”**
- Функция **gets** имеет один параметр, соответствующий массиву символов, в который осуществляется чтение. Вторая функция (**gets_s)** имеет второй параметр, задающий максимальную длину массива символов **Str**.
- И та и другая при вводе текста, длина которого (вместе с нулевым символом) превышает значение второго параметра (то есть длины символьного массива **Str**), приводит к возникновению ошибки при выполнении программы

Использование функции потока ввода cin.getline

- В этой функции первый параметр **Str** соответствует массиву символов, в который должна быть записана взятая из потока ввода текстовая строка. Второй параметр - задает максимальное количество символов, которое может быть помещено в массив **Str** (с учетом завершающего нулевого символа, который добавляется в конец введенного текста автоматически).
- Если длина введенного с клавиатуры текста превышает максимальную длину массива **Str**, в него будет записано (в нашем примере) 20 символов вводимого текста и нулевой символ. Остальные символы введенного текста остаются во входном потоке и могут быть взяты из него следующими инструкциями ввода.

Использование функции потока ввода cin.getline

const int N = 11;

char Str [N];

cin.getline (Str, N); // Пусть введена строка **“Это пример текста”**

cout << Str << endl; // На экран будет выведено **“Это пример”** – 10 символов

cin.getline (Str, N); // Ожидается чтение остальных символов: **“ текста”**

cout << Str << endl; // Однако на экран будет выведена пустая строка

•Для того чтобы продолжить чтение из потока, необходимо восстановить его нормальное состояние. Этого можно достигнуть с помощью функции потока **cin.clear()**, которая сбрасывает состояние потока в нормальное

Использование функции потока ввода cin.getline и cin.clear

- **const int N = 11;**
- **char Str [N];**
- **cin.getline (Str, N);** // Пусть введена строка **“Это пример текста”**
- **cout << Str << endl;** // На экран будет выведено **“Это пример”** – 10 символов
- **cin.clear();** // Сбрасываем состояние потока в норму
- **cin.getline (Str, N);** // Не останавливаясь дочитываем оставшиеся в потоке символы
- **cout << Str << endl;** // На экран выведено: **“ текста”**

- Если забирать остатки данных из потока ввода не надо, то следует очистить его с помощью функции **cin.sync()**:
-
- **const int N = 11;**
- **char Str [N];**
- **cin.getline (Str, N);** // Пусть введена строка **“Это пример текста”**
- **cout << Str << endl;** // На экран будет выведено **“Это пример”** – 10 символов
- **cin.clear();** // Сбрасываем состояние потока в норму
- **cin.sync();** // Очищаем поток от оставшихся символов
- **cin.getline (Str, N);** // Ждем очередного ввода данных. Введено: **“Слово”**
- **cout << Str << endl;** // На экран выведено: **“Слово”**

Потоки для работы с файлами

- Для работы с файлами в языке C++ используются потоки трех видов:
- поток ввода (класс **ifstream**);
- поток вывода (класс **ofstream**);
- поток ввода-вывода (класс **fstream**).
- Класс **ifstream** используется для выполнения чтения данных из файлов. Поток **ofstream –** для записи данных в файлы. Поток **fstream –** для чтения и записи данных в файлы.
- Для использования этих классов потоков необходимо в программу включить заголовочный файл <**fstream**>.

Создание потока, открытие и закрытие файла

ofstream File; - создали поток

File.open ("E:\\test.txt"); - связали поток с файлом (открыли файл)

Либо так:

ofstream File ("E:\\test.txt"); - создали поток и открыли файл

После открытия файла необходимо обязательно проверить открылся ли файл.

Если файл открыть не удалось, то переменная потока (**File**) принимает значение **false**, если файл открыт – **true**. Следовательно, проверку успешного открытия файла можно выполнит так:

if (! File) // Ошибка

Еще один способ – использовать функцию потока **is_open ()**, которая также возвращает логическое значение в зависимости от результата операции открытия файла:

if (! File.is_open ()) // Ошибка

Файл закрывается с помощью функции потока **close ()**:

File.close ();

Прототипы функции Open

```
void ifstream::open ( const char * FileName,
                      ios::openmode Mode = ios::in );

void ofstream::open ( const char * FileName,
    ios::openmode Mode = ios::out | ios::trunc );

void fstream::open ( const char * FileName,
          ios::openmode Mode = ios::in | ios::out );
```

Режимы открытия файла

- **ios::app** – при открытии файла на запись (поток **ofstream**) обеспечивает добавление всех выводимых в файл данных в конец файла;
- **ios::ate –** обеспечивает начало поиска данных в файле начиная с конца файла;
- **ios::in –** файл открывается для чтения из него данных;
- **ios::out –** файл открывается для записи данных в файл;
- **ios::binary –** открытие файла в двоичном режиме (по умолчанию все файлы открываются в текстовом режиме);
- **ios::trunc –** содержимое открываемого файла уничтожается (его длина становится равной 0).

Эти флаги можно комбинировать с помощью побитовой операции ИЛИ (|).

Запись и чтение данных в текстовых файлах

- Запись и чтение данных в текстовых файлах ничем не отличается от способов ввода-вывода данных с помощью потоков **cin** и **cout**. Методы форматирования вывода и вводы данных остаются такими же (флаги форматирования, манипуляторы, функции потоков).
- Необходимо помнить, что при использовании операции >> для чтения данных из текстовых файлов, процесс чтения останавливается при достижении пробельного символа (так же как и в потоках **cin** и **cout**). Поэтому для чтения строки текста, содержащей несколько слов, необходимо, как и раньше, использовать, например, функцию **getline ()**.

Запись и чтение данных в двоичном режиме

- Для работы с файлом в двоичном режиме его необходимо открыть с флагом **ios :: binary**.
- Чтение и запись двоичных данных в этом режиме можно осуществлять двумя способами:

по одному байту – функции файловых потоков **get ()** и **put ()**;

блокам определенной длины - функции файловых потоков **read ()** и **write ()**.

Функции Get() и Put()

ifstream & get (char & ch);

ofstream & put (char ch);

Функция **get ()** берет один символ из потока ввода, помещает его в символьный параметр **ch** и возвращает ссылку на поток ввода. Когда достигается конец файла, значение ссылки на поток становится равным 0.

Функция **put ()** помещает символ **ch** в поток вывода и возвращает ссылку на поток вывода.

Пример записи массива в файл

```
int main ()
{ setlocale (0, "");
  // Запись массива A в_файл "E:\test.dat"
  ofstream o_File;  // Создали поток вывода для записи данных в файл
  o_File.open ( "E:\\test.dat", ios::binary );  // Открыли файл
  if (! o_File.is_open())  // Проверили, удалось ли открыть файл
  {          cout << "Открыть файл не удалось! \n" ;
        return 0;
  }
  // Записываем данные из массива A в файл
    int A[5] = {1, 2, 3, 4, 5};
  char *ch = (char *) A;  // ch – ссылка на массив A, как на массив символов (байт)
  for (int I = 0; I < sizeof(A); ++ I)
        o_File.put(ch[I]);
  o_File.close();  // Закрываем файл
```

Чтение данных из файла в массив B

```
ifstream  i_File;   // Создали поток ввода для чтения данных из файла
i_File.open ( "E:\\test.dat", ios::binary );  // Открыли файл
if (! i_File.is_open())  //  Проверили, удалось ли открыть файл
{       cout << "Открыть файл не удалось! \n" ;
        return 0; }
// Читаем данные из файла в массив B
int B[5];
ch = (char *) B;  //  ch – ссылка на массив B, как на массив символов (байт)
int I = 0;
while (! i_File.eof())
        i_File.get(ch[I++]);
i_File.close();   // Закрываем файл
// Выводим массив B на экран
for (I = 0; I < 5; ++ I)
        cout << B[I] << " ";
cout << endl;
```

Функции Read() и Write()

- **ifstream & read (char * buf, streamsize n);**
- **ofstream & write (const char * buf, streamsize n);**
- Первый параметр этих функций определяет адрес буфера (некоторого массива символов - байт) для чтения или записи данных в соответствующий файловый поток. Второй параметр задает количество символов – байт, которые необходимо взять из потока или записать в поток (тип данных **streamsize –** целый тип данных). Размер буфера должен соответствовать значению второго параметра.

- Функция **write ()** обеспечивает запись из буфера, адрес которого указан в первом параметре функции, **n** символов данных в поток вывода и возвращает ссылку на поток.

- Функция **read ()** обеспечивает запись из потока ввода **n** символов данных в память по адресу, указанному в первом параметре **buf**. При достижении конца файла функция возвращает ссылку на поток, равную 0, а фактическое количество взятых из потока символов может быть меньше, чем значение **n** второго параметра (буфер заполнен не полностью).
- Фактическое количество считанных из потока ввода символов после выполнения последней операции чтения можно определить с помощью функции потока ввода **gcount()**.