

Работа с текст. STL string. Потоци. Текстови файлове

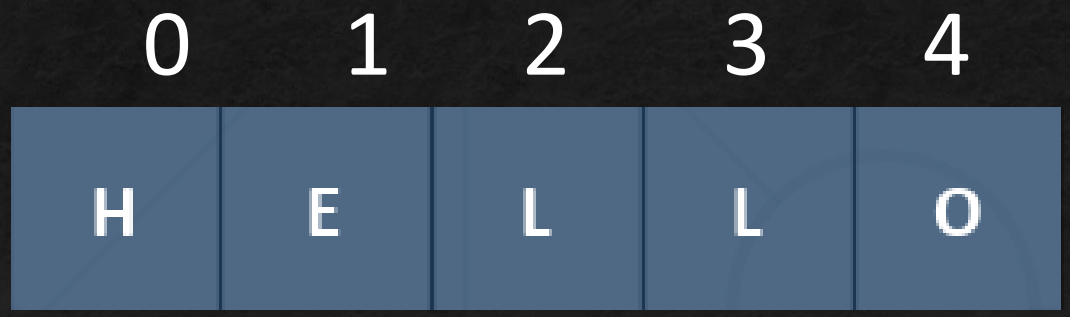
Изготвена от Никола Светославов

Lesson Outline

- Символен низ
- STL string
- Streams
- File – incoming data
- File – outgoing data

Символен низ

- ***Символен низ*** наричаме последователност от символи
- Последователност от 0 символи наричаме ***празен низ***
- Представяне в C++ - масив от символи (char), който завършва със символа '\0'



Примери

- `char word[] = { 'H', 'e', 'l', 'l', 'o', '\0' };`
- `char word[6] = { 'H', 'e', 'l', 'l', 'o' }; //обикновен масив от символи`
- `char word[5] = { 'H', 'e', 'l', 'l', 'o' }; //обикновен масив от символи`
- `char word[100] = "Hello";`
- `char word[5] = "Hello"; //невалидно, понеже " "включват в себе си '\0‘`
- `char word[6] = "Hello";`

Относно '\0'

- Първият символ в ASCII таблицата, с код 0 – т.е char(0)
- Брои се като елемент и влияе на размера на масив.
- Използва се като прекъсвач (терминатор) от много функции за символни низове, за да се определя края на низа.
- Може да се сложи в средата на масив от символи

```
char a []= {'H', 'e', 'l', 'l', '\0', 'o'}; //символният низ е “Hell”
```

Инициализация

- Чрез инициализатор
  - `char text[16] = {'C','+','+', '\0'};`
  - `char sameText[] = {'C','+','+', '\0'};`
- Чрез литерал[1]
  - `char sameTextAgain[] = "C++";`
  - `char sameTextYetAgain[16] = "C++";`

1. В информатиката под литерал се разбира константна стойност на променлива, зададена в сорс кода на дадена компютърна програма.

Quiz

- What will the following code print?

```
char line1[] = { 'a', 'b', 'c' };
char line2[4] = { 'd', 'e', 'f' };
cout << line1 << endl;
cout << line2 << endl;
```

A. It won't – there will be a compile-time error
B. behavior is undefined
C. First line abc. Second line def
D. First line undefined, second line is def

std::cin.getline()

- Фунцкия от библиотеката <iostream> , позволяваща да приемете цял ред като единичен input
- Синтаксис: istream& getline (char* s, streamsize n);
  - Първият параметър е пойнтър към паметта, където искаме да запазим данните.
  - Вторият параметър е броят символи, които най-много искаме да приемем от потребителя
- Функцията автоматично добавя терминираща нула към записания низа!

```
#include <iostream>

int main() {
	char a[5] = "abcd";
	std::cin.getline(a, 5);
 //реално можем да въведем само 4
 //символа, защото 5-ти е '\0'
	std::cout<<a;
	return 0;
}
```

Вход: wazza → wazz

https://stackoverflow.com/questions/18786575/using-getline-in-c/18786719#18786719

C-string

http://www.cplusplus.com/reference/cstring

- The C++ <cstring> header file declares a set of functions to work with C style string (null terminated byte strings).

- Functions
  - strlen(str)
    - връща колко символа има от началото до '\0':
  - strcat(destination, source)
    - Конкатенация
  - strstr(str ,seach)
    - Връща поинтер на първия намерен search в str
    - Ако не е намерен – връща NULL
  - Strcpy(destination, source)
    - Копира source в destination(т.е destination става със стойност на source)

std::string

- Това е обект от стандартната библиотека, който цели да улесни работата със символни низове
- За да се използва std::string, е необходима библиотеката <string> или някоя друга библиотека, която я включва
  - std::string <NAME> [= VALUE];
  - Ако не се подаде някаква стойност по подразбиране се приема стойност "", което е символен низ, съдържащ само '\0'
  - Като стойност може да се подават както друг std::string, така и обикновен символен низ

- std::string name1 = "Ivan";
- std::string name2 = name1;

Оператори

- Оператор ==
  - Оператор == сравнява двата стринга по елементно и връща true ако съвпадат напълно или false ако има разминаване в съдържанието им

```
std::string name1 = "Ivan", name2 = "Tony";
std::cout << (name1 == name2); //0
```

- Оператор +=
  - Позволява да се добави друг стринг към края на настоящия

```
std::string name = "Ivan";
name += " Ivanov";
std::cout << name; //Ivan Ivanov
```

- Оператор > , >= , < и <=
  - Лексикографско сравнение/наредба

Примери: "a" < "b" "a" < "ab"
"A" < "a" "cat" < "caterpillar"

Оператор []

- Както при масиви и std::string, връща символа, който се намира на указания индекс.

- Индексирането започва от 0

```
int main() {
    std::string name = "Ivan" ;

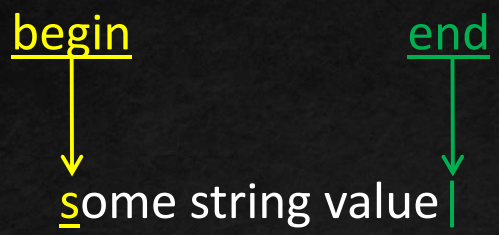
    std::cout << name[0]; //I
    std::cout << name [3]; //n
    return 0;
}
```

Функции

- size() & length()
  - Функцията на класа std::string size връща броя на елементите в стринга, като не брои терминиращата нула.
  - Много е удобна за обхождане на целия стринг.
- insert()
  - Добавя текст след посочения индекс или итератор
- erase()
  - Изтрива символ на подадената позиция
  - Изтрива поредица от символи на подадената позиция
  - Работи само с итератори
- clear()
  - Изтрива всички елементи от стринга като оставя само '\0'
- pop_back()
- begin() и end()

begin() и end()

- begin () и end () са функции, които връщат итератори
- Накратко итераторите са обекти за по-лесна работа със STL обекти



- Към итераторите може да се добавят стойности, които казват колко символа след итератора

Работа с итератори

- Някои функции, които знаят как да работят с итератори

- Ако искаме да вземем стойността на итератора трябва да използваме * пред името му:

```
std::string name = "Pesho";
std::cout << name.begin() + 1;     //Грешка при компилация
std::cout << *(name.begin() + 1); //e
std::cout << *(name.end() - 1);    //o
```

Пример – size()

```
int main() {
    std::string name = "Ivan";
    for(int i = 0; i < name.size(); i++) {
        std::cout << name[i];
    }
    return 0;
}
```

Пример – clear()

```
int main() {
    std::string name = "Ivan";
    name.clear();
    std::cout << name;
    return 0;
}
```

Пример – erase()

```
int main(){
  std::string str ("This is an example sentence.");
  std::cout << str << '\n';
  // "This is an example sentence."
  str.erase(10,8);
  std::cout << str << '\n';
  // "This is an sentence."
  str.erase(str.begin()+9);
  std::cout << str << '\n';
  // "This is a sentence."
  str.erase(str.begin()+5, str.end()-9);
  std::cout << str << '\n';
  // "This sentence."
  return 0;
}
```

Пример – pop_back()

```
int main()
{
std::string str = "hello world!";
str.pop_back();
std::cout << str << '\n';
return 0;
}
```

String като параметър на функция

- Подаването на стринг като параметър на функция е лоша практика, за която ще учите в курса по ООП
- Използвайте референции (когато се налага константни) за параметри на функции, когато ще използвате обекти
  - `void print(const std::string &);`
  - `void trim(std::string &);`



C++ Streams

- Потоците предлагат абстракция над входящи/изходящи данни
  - Напр. cin и cout са абстракции на входа / изхода на конзолата
- На практика, потоците са начини за четене/запис на данни
- Поток може да бъде конструиран за всеки тип контейнер за данни
  - Масиви, низове, памет
  - Файлове, мрежови връзки, буфер на клавиатурата и др.

Поток върху string

- #include <sstream>
- Can read data from a string, can write data to a string
- ***istringstream/ostringstream*** - that only read/write respectively

- Полезна за работа върху низ „дума по дума“
- Полезна за четене на числа от низ.
- Създаване на низ от числа и текст.

Пример

```
string str = "3 -2";

istringstream in(str);

int num1, num2;

in >> num1 >> num2;

int sum = num1 + num2;
```

Пример - ostringstream

```
using namespace std;
//Initialize ostringstream like a normal variable
ostringstream output;

From then on, use the stream just like cout
output << “The sum is “ << num1 + num2 << endl;

//To get the string when you're done, call str()
cout << output.str().
```

Какво представлява файл

- Файлът е ресурс за съхранение на данни на дадено устройство

- Има два основни типа файлове:
  - Текстови:
    - Съхраняват данните под формата на четим текст (plain text)
    - Използват се за съхранение на конфигурации, код и др.
  - Двоични:
    - Съхраняват данните в двоична бройна система (0,1)
    - Използват се за съхранение на всякакви данни, както и за стартиране на нови процеси/програми

Интерпретиране на данни от файл

- Програмистът избира по какъв начин да съхранява данните във файлове
- Има много стандарти за тази цел:
  - Xml
  - Json
  - Yaml
  - Други
- Много често е достатъчно да се импровизира даден прост формат за нуждите на програмата

Пример

- Да кажем, че искаме да съхраним в текстов файл всички студенти от курса заедно с факултетните им номера

- Възможно е да използваме някой от стандартизираните формати за съхранение на данни:

XML

```
<student>
    <fn>12345</fn>
    <name>Pesho</name>
</student>
```

YAML

```
- student:
    fn: '12345'
    Name: 'Pesho'
```

Пример

- Тези формати са се доказали и се използват много в различни сфери
- Те са подходящи за работа с голямо количество разнообразна информация
- Нашата нужда е да се съхраняват само студенти и то само с име и факултетен номер:

12345:Pesho

Пример

- Тъй като знаем каква информация ще съхраняваме, не ни е нужно да уточняваме това в текстовия файл
- Нужно ни е единствено да имаме информацията, която ни трябва - факултетен номер и име
- За да можем да разграничим двете полета едно от друго ни трябва разделител - символ или комбинация от символи, които знаем, че няма как да са част от информацията, която съхраняваме (например : или |, или ||)
- Така можем на всеки ред да съхраняваме по точно един запис за студент:

12345:Pesho

12346:Gosho

12321:Grisho

12353:Tony

.....

Този формат е изключително лесен за обработка и ни върши необходимата работа

Отваряне на файл

- Работата с файлове се осъществява чрез файлови дескриптори - std::fstream
- Това са системни единици, на които отговарят числа, които осъществяват връзката между текущия процес и файла, за който отговарят
- Файловите дескриптори имат 2 основни роли - четене и писане, като може да извършват и двете
- Избира се какъв тип файлов дескриптор ще се използва за работа с файла
- Препоръчително да се използва точно такъв, какъвто ни трябва без излишни привилегии, а не да се използва такъв за писане и четене постоянно
- Ако файлът е отворен само за четене, то друг процес ще може да го отвори за писане ако му се налага, докато ако се отвори за четене и писане без да ни трябва, другият процес ще трябва да изчака ненужно, за да може да пише

Пример

```
#include <fstream>
ifstream input;
input.open("input.txt");

int a, b;
input >> a >> b;
input.close();

ofstream output;
output.open("output.txt");

output << a + b << endl;
output.close();
```

Параметри при отваряне на файл

member constant	stands for	access
in	input	File open for reading: the internal stream buffer supports input operations.
out	output	File open for writing: the internal stream buffer supports output operations.
binary	binary	Operations are performed in binary mode rather than text.
ate	at end	The output position starts at the end of the file.
app	append	All output operations happen at the end of the file, appending to its existing contents.
trunc	truncate	Any contents that existed in the file before it is open are discarded.

Пример

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

int main()
{
   std::fstream myFile1;
   myFile1.open( "hello.txt", std::fstream::out);
   //out => ще изнася информация
   myFile1 << "Hello file world";
   return 0;
}
```

Пояснение

- std::fstream myFile1 създава файлов дескриптор
- myFile1.open("hello.txt", std::fstream::out) кара файловия дескриптор да отвори файла hello.txt в режим на писане, защото std::fstream::out означава, че файловият дескриптор ще изнася информация към файла
- Ако такъв файл не съществува при отваряне, то той ще бъде създаден като празен файл
- myFile1 << "Hello file world" записва стринга във файла

Повторно изпълнение на примера

- При първото изпълнение на примера се създава файл със съдържание "Hello file world"
- Какво ще стане при повторно изпълнение на програмата?
- Отговор: отново ще получим файл със съдържание "Hello file world"
- Причината за това е, че по подразбиране параметъра за отваряне на файл за писане е trunc (truncate), който унищожава цялата информация във файла по време на отваряне

Пример - fixed

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

int main() {
    std::fstream myFile1;
    myFile1.open("hello.txt", std::fstream::out | std::fstream::app);
    myFile1 << "Hello file world";
    return 0;
}
```

Пояснение

- Параметрите за отваряне на флаг (наричани още флагове) се подават разделени с |
- Сега при повторно изпълнение на програмата файлът има съдържание:
- "Hello file worldHello file world"
- Това се дължи на факта, че не сме сложили символ за нов ред в края на стринга, който добавяме

Пример - fixed

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

int main() {
    std::fstream myFile1;
    myFile1.open("hello.txt", std::fstream::out | std::fstream::app);
    myFile1 << "Hello file world\n” ;
    return 0;
}
```

Проверка за успешно отваряне на файл

- Добра практика е да се проверява дали файлът е бил отворен успешно

- Това става с фунцкията is_open()

```
if (myFile.is_open() == false)
{
std::err << "Failed to open file";
return 1;
}
```