String em C++

Em linguagem C, *strings* são implementadas através de vetores de caracteres, cujo último elemento deve ser um caractere nulo (*\0*). **Já em C++** emprega-se a classe **string** para declarar uma *string* de tamanho variável (incluir a biblioteca **<cstring>**).

Inicialização e atribuição de strings

Na declaração de strings pode-se ou não já inicializá-las, como mostra o exemplo a seguir:

```
#include <iostream>
#include <cstring>              // Necessario para usar strings
using namespace std;
int main(){
  string nome1("Fulano");       // Inicializa nome1
  string nome2, nome3;          // Não inicializadas
  cout << "Nome1 " << nome1 << endl;
  nome2 = "Ciclano";            // Inicialização posterior
  nome3 = nome1;
  cout << "Outros nomes sao " << nome2 << " e " << nome3 << endl;
   return 1; }
```

Acesso a caracteres individuais de uma string

O operador **[..]**, usado pelos vetores, também permite o acesso a caracteres individuais dentro de *strings*. **Em C++** existe também o operador **at**, que não permite o acesso a posições inválidas - observe o exemplo abaixo:

```
#include <iostream>
#include <cstring>
using namespace std;
int main(){
  string nome1("Fulano");       // Inicializa nome1
  string nome2;                 // Não inicializada
  nome2 = nome1;
  nome1[5] = 'a';               // Troca nome1 para "Fulana"
  nome2[0] = '*';               // Troca inicial do nome2 "*ulano"
  cout << "Os nome sao: " << nome1 << " e " << nome2 << endl;
  // Sem usar string.at:
  cout << "O caractere 10 do segundo nome eh: " << nome2[10] << endl;
   // ERRO! Não existe o caractere 10! Não mostra valor nem para o progr.
   // Usando string.at:
  cout << "O caractere 10 do 2o nome eh: " << nome2.at(10) << endl;
   // ERRO, aborta o programa, melhor!!!
   return 1; }
```

Em ***nome2.at(10)*** o programa termina de forma anormal - gerando uma *exceção* que pode ser tratada futuramente, o que é bom! A vantagem é não correr o risco de acessar memória inválida – o que frequentemente levava a resultados desastrosos em C, com o uso do [ ]!

Funções de caracteres úteis

As seguintes funções estão na biblioteca **<cctype>**

- **toupper(var)** – (`to+upper`) retorna a maiúscula de uma letra, tem um argumento (o caractere).
- **tolower(var)** – (`to+lower`) o mesmo comportamento que `toupper()`, porém o resultado em minúscula.

Funções que verificam o caractere: recebem apenas um argumento (o caractere) e retornam um valor booleano.

Função	Descrição
`isalpha`	Retorna `true` se o argumento é uma letra do alfabeto; `false` em caso contrário.
`isalnum`	Retorna `true` se o argumento é uma letra do alfabeto ou um dígito; `false` em caso contrário.
`isdigit`	Retorna `true` se o argumento é um dígito; `false` em caso contrário.
`islower`	Retorna `true` se o argumento é uma letra minúscula, `false` em caso contrário.
`isprint`	Retorna `true` se o argumento é um caractere imprimível (incluindo espaços); `false` em caso contrário.
`ispunct`	Retorna `true` se o argumento é um sinal de pontuação (caracteres imprimíveis que não sejam letras, dígitos ou espaço); `false` em caso contrário.
`isupper`	Retorna `true` se o argumento é uma letra maiúscula; `false` em caso contrário.
`isspace`	Retorna `true` se o argumento é um espaço, tabulação ou nova linha; `false` em caso contrário.

Comparação de strings

Em C++, usa-se os operadores **==, !=, >, >=** ou **<** para fazer comparações alfabéticas entre duas *strings*. Também pode-se empregar a função **compare** para comparar 2 strings, retornando um valor numérico igual a 0 (iguais), menor que 0 (1ª string vem antes) ou maior que 0 (1ª string vem depois), da mesma forma que a função *strcmp* em C.

Exemplo:

```
#include <iostream>
#include <cstring>
using namespace std;
int main(){
  string nome1("Fulano");      // Inicializa nome1
  string nome2("Beltrano");    // Inicializa nome2
  if(nome1 > nome2)
         cout << nome1 << " vem depois de " << nome2 << endl;
  else if(nome1 == nome2)
         cout << "Os nomes são iguais" << endl;
  else
         cout << nome1 << " vem antes de " << nome2 << endl;
   return 1;
}
```

Manipulando o tamanho de uma string

Em C++ *strings* têm tamanho variável, assim existem uma série de funções que podem ser utilizadas para controlar e manipular essa propriedade. Por tamanho variável, deve-se entender que uma *string* reserva uma certa quantidade de memória (sua *capacidade*), mas não necessariamente utiliza toda essa memória. Caso a *string* cresça mais do que a sua capacidade, então mais memória é reservada e assim por diante. A seguir apresenta-se as principais funções:

- **size**(): retorna o tamanho corrente da *string*;
- **capacity**(): retorna a capacidade corrente da *string*, ou seja, quantos elementos ela poderá conter antes de necessitar mais memória;
- **max_size**(): retorna o tamanho máximo possível em uma *string*, geralmente dependente da máquina e do compilador.

Exemplo de utilização da função **size**():

```
#include <iostream>
#include <cstring>
using namespace std;
int main(){
 string s1, s2("valor inicial");
 cout << "tamanho inicial de s1 eh: " << s1.size() << endl;
 cout << "capacidade de s1 eh: " << s1.capacity() << endl;
 cout << "tamanho máximo possivel p/s1 eh: " << s1.max_size() << endl;
  s1 = s2 + " copiado de s2";        // concatenacao e atribuicao
 cout << s1 << endl;
 cout << "tamanho inicial de s1 mudou para: " << s1.size() << endl;
 for(unsigned int i=0; i<s1.size(); i++)
        cout<<s1.at(i)<<" ";         // mostra cada elemento, ou s1[i]
 cout << endl;
 return 1; }
```

DEMAIS FUNÇÕES => http://www.cplusplus.com/reference/string/string/

Member functions

(constructor)	Construct string object (*public member function*)
(destructor)	String destructor (*public member function*)
operator=	String assignment (*public member function*)

Iterators:

begin	Return iterator to beginning (*public member function*)
end	Return iterator to end (*public member function*)
rbegin	Return reverse iterator to reverse beginning (*public member function*)
rend	Return reverse iterator to reverse end (*public member function*)
cbegin	Return const_iterator to beginning (*public member function*)
cend	Return const_iterator to end (*public member function*)
crbegin	Return const_reverse_iterator to reverse beginning (*public member function*)
crend	Return const_reverse_iterator to reverse end (*public member function*)

Capacity:

size	Return length of string (*public member function*)
length	Return length of string (*public member function*)
max_size	Return maximum size of string (*public member function*)
resize	Resize string (*public member function*)
capacity	Return size of allocated storage (*public member function*)
reserve	Request a change in capacity (*public member function*)
clear	Clear string (*public member function*)
empty	Test if string is empty (*public member function*)
shrink_to_fit	Shrink to fit (*public member function*)

String operations:

c_str	Get C string equivalent (*public member function*)
data	Get string data (*public member function*)
get_allocator	Get allocator (*public member function*)
copy	Copy sequence of characters from string (*public member function*)
find	Find content in string (*public member function*)
rfind	Find last occurrence of content in string (*public member function*)
find_first_of	Find character in string (*public member function*)
find_last_of	Find character in string from the end (*public member function*)
find_first_not_of	Find absence of character in string (*public member function*)
find_last_not_of	Find non-matching character in string from the end (*public member function*)
substr	Generate substring (*public member function*)
compare	Compare strings (*public member function*)

Element access:

operator[]	Get character of string (*public member function*)
at	Get character in string (*public member function*)
back	Access last character (*public member function*)
front	Access first character (*public member function*)

Member constants

npos	Maximum value for size_t (public static member constant)

Non-member function overloads

operator+	Concatenate strings (function)
relational operators	Relational operators for string (function)
swap	Exchanges the values of two strings (function)
operator>>	Extract string from stream (function)
operator<<	Insert string into stream (function)
getline	Get line from stream into string (function)