Universidade de Brasília

Vetores de caracteres (Strings)

Computação para Engenharia — Tópico 6 — Parte 2

Daniel Guerreiro e Silva

Departamento de Engenharia Elétrica (ENE), Faculdade de Tecnologia (FT)

Roteiro

Cadeias de caracteres

Lendo e escrevendo cadeias

Manipulando cadeias de caracteres

Cadeias de caracteres

Cadeias de caracteres

Uma cadeia de caracteres, mais conhecida como *string*, é uma sequência de letras e símbolos, onde os símbolos podem ser espaços em branco, dígitos e vários outros como pontos de exclamação e interrogação, símbolos matemáticos, etc.

Há duas formas de representar strings em C++:

1. Por um vetor de variáveis do tipo `char` e é terminada com o caractere especial `'\0'`. Essa representação é também chamada de C-strings.
2. Por uma variável do tipo `std::string`, definido na biblioteca padrão do C++. É neste tipo que focaremos nosso estudo.

Declarando uma variável `std::string`

No cabeçalho do programa, deve-se incluir a diretiva para inclusão da biblioteca:

```
#include <string>
```

Exemplo de declaração

```
std::string texto;
```

Veja que, diferentemente dos outros tipos de vetores, não é necessário definir o tamanho, isto ocorre porque o espaço em memória para variáveis `std::string` é definido em tempo de execução, isto é, o espaço vai sendo alocado à medida que for necessário.

Acesso indexado

Tanto as C-strings como as variáveis `std::string` tem acesso indexado, isto é, podemos referenciar/acessar um determinado caractere da cadeia com o índice entre colchetes:

```
texto[4] = 'f';
```

Lendo e escrevendo cadeias

Lendo uma string do teclado

Podemos ler uma cadeia caractere a caractere, como faríamos com qualquer outro vetor, mas é mais simples ler a cadeia inteira, ou seja

```
cin >> texto;
```

Exemplo: `leia.cpp`.

```cpp
#include <iostream>
#include <string>
using namespace std;

int main () {

  string texto;

  /* Interrompe a leitura no primeiro espaco em branco */
  cin >> texto;
  cout << texto << endl;

  return 0;
}
```

Lendo uma cadeia do teclado

Infelizmente, a leitura a partir do teclado utilizando `cin >>` lê somente até o primeiro espaço, ou seja, lê somente uma palavra, o que torna o seu uso um pouco restrito.

- A solução é usar a função `getline` para `std::string`

```
getline(cin, texto);
```

Ela faz a leitura até encontrar o caractere de fim de linha (*enter*).

Exemplo: `getline.cpp`.

```cpp
#include <iostream>
#include <string>
using namespace std;

int main () {

  string texto;

  getline(cin, texto);
  cout << texto << endl;

  return (0);
}
```

Escrevendo uma cadeia na tela

Podemos escrever uma cadeia na tela caractere a caractere, mas é mais simples escrever de forma direta:

```
cout << texto;
```

Manipulando cadeias de caracteres

Manipulando objetos `string`

A manipulação de variáveis do tipo `std::string` é relativamente fácil. Daí vem um dos benefícios em usar esse tipo de dado da biblioteca padrão do C++ no lugar de um vetor `char` (C-strings).

Considere as variáveis a seguir previamente declaradas do tipo `std::string`.

Manipulando objetos `string`

Para obter o tamanho da string `texto`, basta invocar no código

```
texto.size()
```

Manipulando objetos `string`

Para copiar a string `fonte` para a string `destino`, basta fazer

```
destino = fonte;
```

Manipulando objetos `string`

Para concatenar a string `fonte` no fim da string `destino`, basta fazer

```
destino = destino + fonte;
```

Exemplo: funcoesstring.cpp.

```cpp
#include <iostream>
#include <string>//cabecalho de strings do C++
using namespace std;

int main () {

  //pode-se declarar E inicializar uma string como aqui embaixo
  string feliz = "Eu sou um texto feliz.";
  string triste = "Eu sou um texto triste.";
  string indeciso;
  int l;

  l = feliz.size();//obtendo tamanho da string
  cout << "Comprimento do texto feliz: " << l << endl;

  cout << "Valor de indeciso antes de qualquer operacao:\n";
  cout << indeciso << '\n';

  indeciso = feliz; //copia de strings no C++ pode ser por atribuicao direta

  cout << "Valor de indeciso depois da copia:\n";
  cout << indeciso << '\n';

  indeciso = indeciso + triste; //concatenacao pode ser feita com o +

  cout << "Valor de indeciso depois da concatenacao:\n";
  cout << indeciso;

  return 0;
}
```

Manipulando objetos `string`

A comparação entre variáveis `std::string` é relativamente fácil, também. Basta usar os operadores relacionais já vistos para tipos numéricos.

`str1 == str2`: dá `true` se str1 e str2 forem iguais;

`str1 < str2`: dá `true` se str1 for menor que str2;

`str1 > str2`: dá `true` se str1 for maior que str2;

A comparação pode ser usada, por exemplo, em

1. um programa que faça uma busca sequencial por um caractere ou
2. em um programa que ordena um vetor de variáveis `string`.

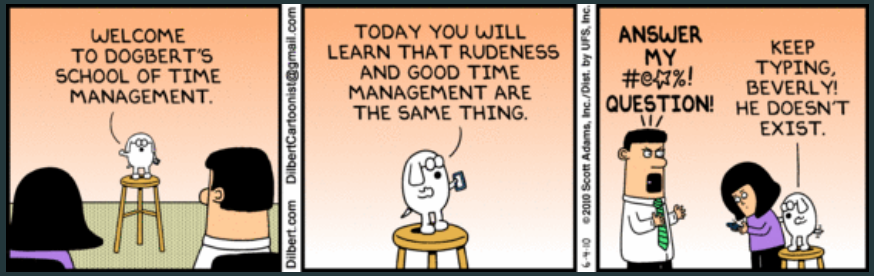
Busca sequencial por um caractere numa string

```cpp
//Procura um elemento em uma cadeia de caracteres - busca linear.
#include <iostream>
#include <string>
using namespace std;

int main() {
  string str;
  char c;
  int i;
  bool achou = false;

  cout << "Digite uma cadeia de caracteres: ";
  cin >> str;
  cout << "Digite um caractere: ";
  cin >> c;

  //laco de busca sequencial
  for (i = 0; !achou && i<str.size(); i++)
    if(str[i] == c)
      achou = true;

  if (achou) //varreu e nao achou
    cout << "O caractere " << c << " esta presente na string " << str << "\n";
  else
    cout << "O caractere " << c << " nao esta presente na string " << str << "\n";

  return 0;
}
```

Exemplo: ordenastring.cpp.

```cpp
#include <iostream>
#include <string>
#include <iomanip>
#define N_FLORES 5
using namespace std;

int main() {
  int i, j, k;
  //repare que abaixo declaramos um vetor de strings
  string vetor[] = {"rosa", "cravo", "margarida", "violeta", "amor-perfeito"};
  string aux;

  for (k = 0; k < N_FLORES-1; k++)
    cout << setw(13) << vetor[k] << ", ";
  cout << vetor[N_FLORES-1] << "\n";

  for(i = 1; i < N_FLORES; i++) {
    aux = vetor[i];
    //algoritmo de ordenacao insertion-sort
    for(j = i-1; (j >= 0) && (aux < vetor[j]); j--)
      vetor[j + 1] = vetor[j];
    vetor[j + 1] = aux;

    //imprime lista parcialmente ordenada
    for (k = 0; k < N_FLORES-1; k++)
      cout << setw(13) << vetor[k] << ", ";
    cout << vetor[N_FLORES-1] << "\n";
  }
  return 0;
}
```

Dilbert


WELCOME TO DOGBERT'S SCHOOL OF TIME MANAGEMENT.
Dilbert.com DilbertCartoonist@gmail.com
TODAY YOU WILL LEARN THAT RUDENESS AND GOOD TIME MANAGEMENT ARE THE SAME THING.
6-4-10 ©2010 Scott Adams, Inc./Dist. by UFS, Inc.
ANSWER MY #@☆%! QUESTION!
KEEP TYPING, BEVERLY! HE DOESN'T EXIST.