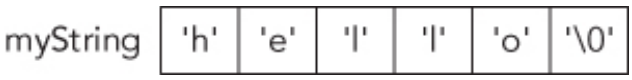
Manipulação de Strings

Uma das atividades mais comuns no dia a dia de um programador é manipulação de strings. Cada linguagem tem sua peculiaridade sobre o tratamento de strings. Em C++ string é definido como um tipo abstrato, ou seja uma classe.
Este material tem o objetivo de apresentar de forma sucinta os principais métodos desta classe.

C ++ tem em sua definição uma maneira de representar uma sequência de caracteres como um objeto de classe. Essa classe é chamada std::string. A classe String armazena os caracteres como uma sequência de bytes com uma funcionalidade de permitir acesso a caracteres de byte único.

vetor de caracteres versus std::string

- Em linguagem C, strings são representadas como um vetor de caracteres. O último caractere de uma string é um caractere nul (\0), e assim que o código onde termina a string. (nul é diferente do null de um ponteiro.
-



- Uma string é uma classe que define objetos que são representados como fluxo de caracteres.
- O tamanho do vetor de caracteres precisa ser alocado estaticamente, não sendo possível alocar mais espaço de memória em tempo de execução). A memória alocada e não utilizada é desperdiçada no caso de um vetor de caracteres.
- No caso de strings, a memória é alocada dinamicamente. Mais memória pode ser alocada em tempo de execução sob demanda. Como nenhuma memória é pré-alocada, nenhuma memória é desperdiçada.
- Existe uma ameaça de deterioração do vetor no caso de um vetor de caracteres (Acontece quando passamos um vetor ou ponteiro de um vetor como parâmetro por valor). Como as strings são representadas como objetos, essa deterioração não acontece.
- A implementação de vetor de caracteres é mais rápida que std::string. As strings são mais lentas quando comparadas à implementação do vetor de caracteres.
- O vetor de caracteres não oferece funções embutidas para manipular seqüências de caracteres. A classe String define um número de funcionalidades (funções/métodos) que permitem operações múltiplas em strings.

Operações sobre strings

Função e métodos de entrada

1. getline ( ): - Esta função é usada para armazenar um fluxo de caracteres (stream) conforme digitado pelo usuário na memória do objeto.

2. push_back ( ): - Este método é usado para inserir um caractere no final da string.

3. pop_back ( ): - Este é um método introduzido a partir do C ++11 (para strings), e é usado para excluir o último caractere da string.

```
#include<iostream>
#include<string> // classe string
using namespace std;
int main()
{
    // Declarando string
    string str;

    // Comando de entrada para uma string usando getline()
    getline(cin,str);

    // Mostrando uma string
    cout << "O inicio da string e : ";
    cout << str << endl;

    // Usando push_back() para inserir um caractere no fim
    // inserindo 's' neste caso
    str.push_back('s');

    // Mostrando a string
    cout << "A string depois da operacao push_back e : ";
    cout << str << endl;

    // Usando pop_back() para deletar um caractere do inicio
    // retirando 's' neste caso
    str.pop_back();

    // Mostrando a string
    cout << "A string depois da operacao after pop_back e : ";
    cout << str << endl;

    return 0;

}
```

Métodos de capacidade

4. capacity ( ): - Este método retorna a capacidade alocada para a sequência, que pode ser igual ou superior ao tamanho da sequência. Espaço adicional é alocado para que, quando os novos caracteres forem adicionados à sequência, as operações possam ser realizadas com eficiência.

5. resize ( ): - Este método altera o tamanho da string, o tamanho pode ser aumentado ou diminuído.

6.length ( ): - Este método encontra o comprimento da string

7.shrink_to_fit ( ): - Este método diminui a capacidade da string e a torna igual ao seu tamanho. Esta operação é útil para economizar memória adicional se tivermos certeza de que nenhuma adição adicional de caracteres precisa ser feita.

```
#include<iostream>
#include<string> // classe string
using namespace std;
int main()
{
    // Inicializando a string
    string str = "Tecnicas de programacao"; //21 letras e dois espacos

    // Mostrando string
    cout << "A string inicial e : ";
    cout << str << endl;

    // Redimensionando a string usando resize()
    str.resize(13);

    // Mostrando string
    cout << "A string depois do redimensionamento e  : ";
    cout << str << endl;

    // Mostrando a capacidade da string
    cout << "A capacidade da string e : ";
    cout << str.capacity() << endl;

    // Mostrando o comprimento da string
    cout<<"O comprimento da string e :"<<str.length()<<endl;

    // Decrementando a capacidade da string string usando shrink_to_fit()
    str.shrink_to_fit();

    // Mostrando string
    cout << "The new capacity after shrinking is : ";
    cout << str.capacity() << endl;

    return 0;

}
```

Métodos de iteração

8. begin ( ): - Este método retorna um iterador para o início da string.

9. end ( ): - Este método retorna um iterador para o final da string.

10. rbegin ( ): - Este método retorna um iterador reverso apontando para o final da string.

11. rend ( ): - Este método retorna um iterador reverso apontando no início da string.

```
#include<iostream>
#include<string> // classe string
using namespace std;
int main()
{
    // Inicializando string`
    string str = "Tecnicas de Programacao";

    // Declarando iterator
    std::string::iterator it;

    // Declarando iterator reverso
    std::string::reverse_iterator it1;

    // Mostrando a string
    cout << "A string usando iterador com avanço e : ";
    for (it=str.begin(); it!=str.end(); it++)
    cout << *it;
    cout << endl;

    // Mostrando a string reversa
    cout << "A string reversa usando o iterador reverso e : ";
    for (it1=str.rbegin(); it1!=str.rend(); it1++)
    cout << *it1;
    cout << endl;

    return 0;

}
```

Mais métodos de Manipulação

12. copy (“char array”, len, pos): - Este método copia a substring no vetor de caracteres de destino mencionado em seus argumentos. São necessários 3 argumentos, vetor de caracteres de destino, comprimento a ser copiado e posição inicial na string para iniciar a cópia.

13. swap ( ): - Este método troca uma string por outra.

```
#include<iostream>
#include<string> // classe string
using namespace std;
int main()
{
    // Inicializando a primeira string
```

```
    string str1 = "Tecnicas de Programacao";

    // Declarando a segunda string
    string str2 = "Estrutura de Dados";

    // Declarando o vetor de caracteres
    char ch[80];

    // usando copy() para copiar elementos dentro do vetor char
    // copia "Tecnicas"
    str1.copy(ch,8,0);

    // Mostrando o vetor char
    cout << "O novo vetor de caracteres copiado e : ";
    cout << ch << endl << endl;

    // Mostrando as strings antes da troca
    cout << "A primeira string antes da troca e : ";
    cout << str1 << endl;
    cout << "A segunda string antes da troca e : ";
    cout << str2 << endl;

    // usando swap() para trocar o conteudo da string
    str1.swap(str2);

    // Mostrando as string depois da troca
    cout << "A primeira string depois da troca e : ";
    cout << str1 << endl;
    cout << "A segunda string depois da troca e : ";
    cout << str2 << endl;

    return 0;

}
```

Aplicações da Classe string

A classe string faz parte da biblioteca C++ que suporta muitas funcionalidades em relação às strings no estilo da linguagem C (vetor de caracteres).
A classe de string C++ usa internamente um vetor char para armazenar caracteres, mas todo o gerenciamento de memória, alocação e terminação nul é tratado pela própria classe de strings, e é por isso que é fácil de usar. O comprimento da cadeia de caracteres C++ pode ser alterado em tempo de execução devido à alocação dinâmica de memória semelhante a vetores. Como a classe string é uma classe contêiner, podemos iterar todos os seus caracteres usando um iterador semelhante a outros contêineres como vector, set e maps, mas geralmente usamos um loop for simples para iterar sobre os caracteres e indexá-los usando o operador [ ].

A classe string C++ possui várias funções para manipular facilmente a string. Os mais úteis deles são demonstrados no código abaixo:

```
#include <bits/stdc++.h>
#include <iostream>
using namespace std;

int main()
{
    // Varios construtores da classe string

    // incializando por a string em linha
    string str1("primeira string");

    // inicializando por outra string
    string str2(str1);
    // inicializacao por caractere com numero de ocorrencias
    string str3(5, '#');

    // inicializacao com parte de outra string
    string str4(str1, 6, 6); //    a partir do sexto (segundo parametro)
                             // 6 caracteres (terceiro parametro)

    // inicializacao por parte de outra string : versao iterator
    string str5(str2.begin(), str2.begin() + 5);

    cout << str1 << endl;
    cout << str2 << endl;
    cout << str3 << endl;
    cout << str4 << endl;
    cout << str5 << endl;

    //  associando via operador
    string str6 = str4;

    // funcao clear deleta todos caracteres a partir da string
    str4.clear();

    //  ambos size() e length() retornam o comprimento da string e
    //  eles trabalham como sinonimos
    int len = str6.length(); // Mesmo que "len = str6.size();"

    cout << "Comprimento da string e : " << len << endl;

    // um caractere particular pode ser acessado usando at /
    // o operador []
    char ch = str6.at(2); //  Mesmo que  "ch = str6[2];"
```

```
    cout << "Terceiro caractere da string e : " << ch << endl;

    //  funcao front retorna o primneiro caractere
    //  e a funcao back retorna o ultimo caractere da string

    char ch_f = str6.front();  // Mesmo que "ch_f = str6[0];"
    char ch_b = str6.back();   // Mesmo que "ch_b = str6[str6.length() -
1];"

    cout << "Primeiro char e : " << ch_f << ", Ultimo char e : "
         << ch_b << endl;

    // c_str retorna uma versao da string como vetor terminado com nul
    const char* charstr = str6.c_str();
    printf("%s\n", charstr);

    // O metodo append adiciona o argumento no fim da string
    str6.append(" extensao");
    //  mesmo que str6 += " extensao"

    // outra versao do append, acrescenta parte de outra string
    str4.append(str6, 0, 6);  // na posicao 0 6 caracteres

    cout << str6 << endl;
    cout << str4 << endl;

    //  find retorna indice onde o padrao e encontrado.
    //  Se o padrao nao for encontrado, ele retornara a constante npos
    //  predefinida com valor -1

    if (str6.find(str4) != string::npos)
        cout << "str4 foi encontrado em str6 no indice " <<
str6.find(str4)
             << " pos" << endl;
    else
        cout << "str4 nao foi encontrado em str6" << endl;

    //  funcao substr(a, b) retorna uma substring de comprimento b
    //  iniciando em a
    cout << str6.substr(7, 3) << endl;

    //  se o segundo argumento nao for passado, uma substring até o fim
    //  da string será retornada
    cout << str6.substr(7) << endl;

    //  erase(a, b) deleta b caracteres a partir de a
    str6.erase(7, 4);
```

```
    cout << str6 << endl;

    //  versao iterator do erase
    str6.erase(str6.begin() + 5, str6.end() - 3);
    cout << str6 << endl;

    str6 = "Este e um exemplo ";

    //  replace(a, b, str)  substitui b caracteres a partir de a por str
    str6.replace(2, 7, "apenas um teste");

    cout << str6 << endl;

    return 0;
}
```