PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ
ESCOLA POLITÉCNICA
CURSO DE TECNOLOGIA EM JOGOS DIGITAIS

ANDRÉ LUÍS JELLER SELLETI

STANDARD TEMPLATE LIBRARY

CURITIBA
2016

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ
ESCOLA POLITÉCNICA
CURSO DE TECNOLOGIA EM JOGOS DIGITAIS

ANDRÉ LUÍS JELLER SELLETI

STANDARD TEMPLATE LIBRARY

Artigo apresentado ao curso de Jogos Digitais como TDE para a disciplina Estrutura de Dados, sob a orientação do Professor Edson Jose Rodrigue Justino.

CURITIBA

2016

1) Introdução ao STL e sua finalidade.

A STL é uma biblioteca C++ genérica de containers, algoritmos e iteradores, e devido a isso quase todos seus componentes são e podem ser utilizados como modelo. E com isso simplificando o trabalho com as estruturas de dados.

2) Propriedades, recursos e flexibilidade da STL.

Esta biblioteca contem na categoria “Containers” as classes “*vetor, lista, deque, conjunto, multiset, mapa, multimap, hash_set, hash_multiset, hash_map e hash_multimap*” de modelo e podem ser utilizadas de várias maneiras diferentes dependendo do tipo de objeto que se quer instanciar.

Como exemplo de praticidade e simplicidade podemos utilizar a classe Vector que pode facilmente substituir um array comum em que não se necessita perder tempo fazendo a alocação dinâmica deste mesmo.

```cpp
#include <iostream>
#include <vector>

class sorvete {

};

int main(){

    std::vector<int> vetor; //Declarando um vetor sem tamanho definido
    vetor[0] = 5; // dando um valor a posição 0 do vetor de maneira comum a um array normal
    vetor.push_back.at(3) = 5; // Outra maneira de colocar um valor dentro de vetor na posição escolhida
    vetor.push_back(7); // Deste modo ele apenas insere um valor no final da fila independente de seu tamanho;
    vetor.pop_back.at(2); // Assim ele apaga o valor na posição desejada e ao mesmo tempo ja rearranja o array
    //de maneire que o proximo valor (posição3) vire a posição 2

    std::vector<sorvete>* teste; // Podendo também fazer um Vector de classes, de ponteiro para obejetos
    teste->push_back.at(0) = new sorvete();//Incluindo um novo objeto da classe sorvete no final da lista
    //que no caso seria o começo

}
```

Outro exemplo fácil de se mostrar, na categoria “Algoritmos” é a inversão do Array/Vetor, utilizando a função *Reverse(~,~)*, demonstrado assim.

```cpp
#include <iostream>
#include <vector>

int main(){

    std::vector<int> vetor;
    vetor[0] = 5; //Posiçao 0 contem valor 5
    vetor.push_back.at(1) = 20; // Pos 1 valor 20
    vetor.push_back(17); // no final (pos 2) valor 17
    vetor.push_back(14); // e no final (pos 3) valor 17
    // com recurso Reverse podemos inverter todo o vetor
    //ou apenas de uma posição a outra, dependendo dos parametros que se passar na função
    reverse(vetor.begin(), vetor.end());
    //apos isso o vetor vira
    //vetor[0] = 14
    //vetor[1] = 17
    //vetor[2] = 20
    //vetor[3] = 5

    //@AndreJeller
}
```

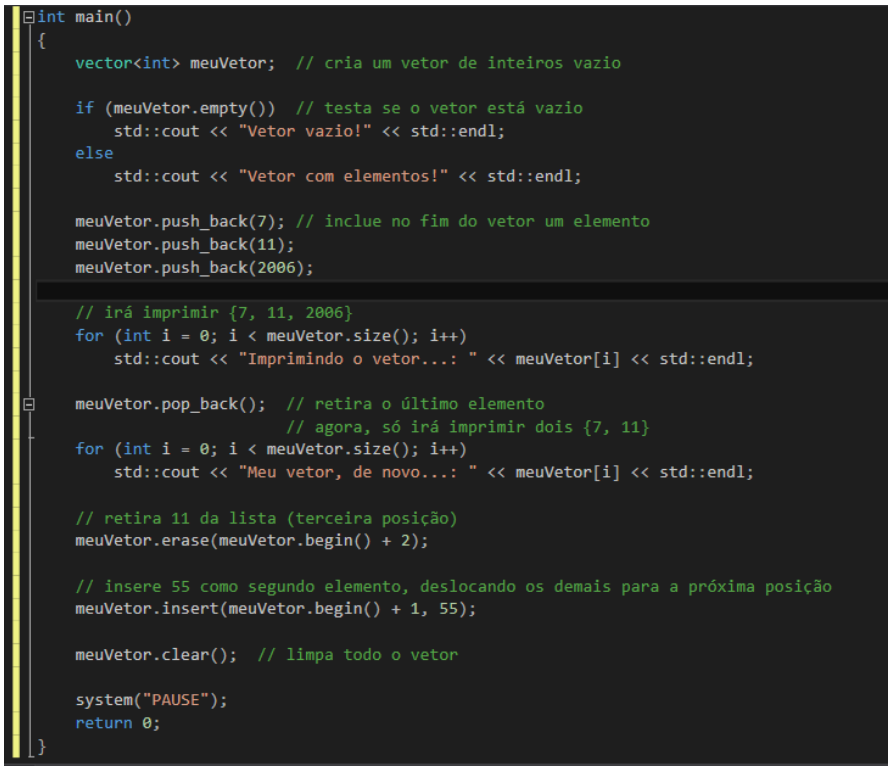
E por ultimo, temos também a categoria dos *“iteradores”* que são iguais a ponteiros, armazenam informações especificas as categorias/dados em que operam. Agora utilizando parte do exemplo anterior para complementar este.

turaDeDados (Global Scope)

```cpp
#include <iostream>
#include <vector>

int main(){

    std::vector<int> vetor;
    vetor[0] = 5;
    vetor.push_back.at(1) = 20;
    vetor.push_back(17);
    vetor.push_back(14);

    reverse(vetor.begin(), vetor.end());
    //.begin() e .end() são exemplos de utilização de iterador
    // pois são implementados com o mesmo tipo do contêiner a percorrer
}
```

3) Funções STL e exemplos de uso.

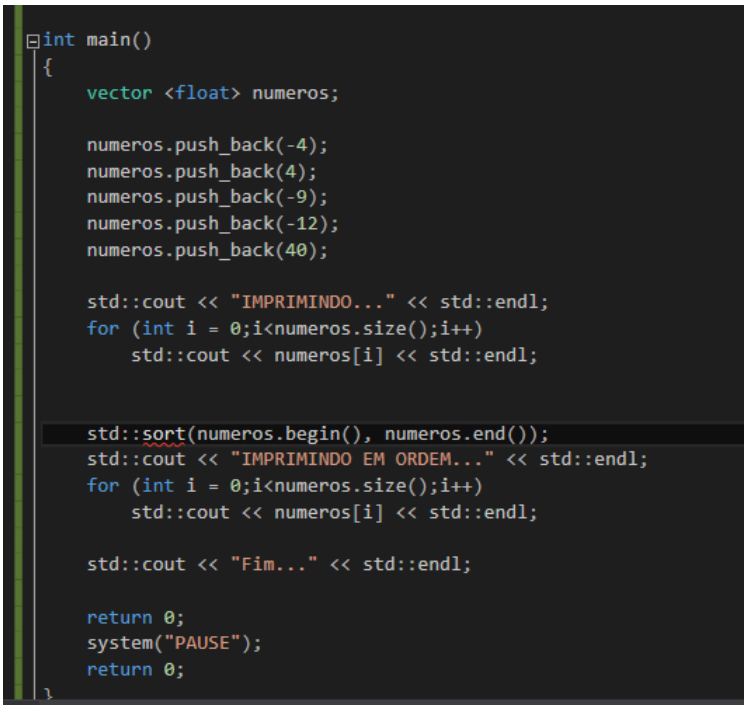
- Funções básicas da classe Vector.



```cpp
int main()
{
    vector<int> meuVetor;  // cria um vetor de inteiros vazio

    if (meuVetor.empty())  // testa se o vetor está vazio
        std::cout << "Vetor vazio!" << std::endl;
    else
        std::cout << "Vetor com elementos!" << std::endl;

    meuVetor.push_back(7); // inclue no fim do vetor um elemento
    meuVetor.push_back(11);
    meuVetor.push_back(2006);

    // irá imprimir {7, 11, 2006}
    for (int i = 0; i < meuVetor.size(); i++)
        std::cout << "Imprimindo o vetor...: " << meuVetor[i] << std::endl;

    meuVetor.pop_back();  // retira o último elemento
                          // agora, só irá imprimir dois {7, 11}
    for (int i = 0; i < meuVetor.size(); i++)
        std::cout << "Meu vetor, de novo...: " << meuVetor[i] << std::endl;

    // retira 11 da lista (terceira posição)
    meuVetor.erase(meuVetor.begin() + 2);

    // insere 55 como segundo elemento, deslocando os demais para a próxima posição
    meuVetor.insert(meuVetor.begin() + 1, 55);

    meuVetor.clear();  // limpa todo o vetor

    system("PAUSE");
    return 0;
}
```

Exemplo disponível na referencia 4.

- Ordenar um Vector numérico.



```cpp
int main()
{
    vector <float> numeros;

    numeros.push_back(-4);
    numeros.push_back(4);
    numeros.push_back(-9);
    numeros.push_back(-12);
    numeros.push_back(40);

    std::cout << "IMPRIMINDO..." << std::endl;
    for (int i = 0;i<numeros.size();i++)
        std::cout << numeros[i] << std::endl;


    std::sort(numeros.begin(), numeros.end());
    std::cout << "IMPRIMINDO EM ORDEM..." << std::endl;
    for (int i = 0;i<numeros.size();i++)
        std::cout << numeros[i] << std::endl;

    std::cout << "Fim..." << std::endl;

    return 0;
    system("PAUSE");
    return 0;
}
```

Exemplo disponível em referência 4 e 6.

- Vector com Classes. Classe Nomes

```cpp
#include <iostream>
#include <algorithm>
#include <vector>
//using std::vector;
using namespace std;

class Pessoa {
    string nome;
    int idade;
public:
    Pessoa(string no, int id)
    {
        idade = id;
        nome = no;
    }
    string getNome()
    {
        return nome;
    }
    int getIdade()
    {
        return idade;
    }
};
```

```cpp
int main() {

    vector <Pessoa> listaNomes;
    vector <Pessoa>::iterator ptr;

    listaNomes.push_back(Pessoa("Joao", 25));
    listaNomes.push_back(Pessoa("Maria", 32));
    listaNomes.push_back(Pessoa("Carla", 4));
    listaNomes.push_back(Pessoa("Abel", 30));
}
```

```cpp
    sort(listaNomes.begin(), listaNomes.end(), ordena_por_nome);

    // percorrendo a lista com um ITERATOR
    for (ptr = listaNomes.begin(); ptr != listaNomes.end(); ptr++)
    {
        std::cout << "Nome: " << ptr->getNome();
        std::cout << " - Idade: " << ptr->getIdade() << std::endl;
    }
    system("pause");

}
```

```cpp
bool ordena_por_nome(Pessoa A, Pessoa B)
{
    if (A.getNome() > B.getNome())
        return true;
    return false;
}
```

Exemplo disponível na referencia 4.

4) Conclusão

Considero a biblioteca STL algo completamente útil, usável e facilmente aplicável, uma vez que já a utilizei para concluir um trabalho no semestre anterior que sem ela eu não conseguiria fazer devido a parte de alocação dinâmica e outras coisas.

Com isso concluo que a Standard Template Library é algo que eu já conhecia mas não tinha ideia do que realmente era, e com esta trabalho consegui entender melhor sua funcionalidade e proposito dentro da matéria.

5) Referencias:.

1. **http://sweet.ua.pt/joao.p.silva/stl.html**
2. **http://www.inf.pucrs.br/~pinho/PRGSWB/STL/stl.html#PUCRS**
3. **http://www.sgi.com/tech/stl/stl_introduction.html**
4. **http://www.inf.pucrs.br/~pinho/PRGSWB/STL/stl.html#Iteradores**
5. **https://www.youtube.com/watch?v=Cq1h1KPoGBU**
6. **http://www.cplusplus.com/reference/algorithm/sort/**