Insper

SuperComputação

Aula 3 – C++ (STL e referências)

2019 – Engenharia

Luciano Soares <lpsoares@insper.edu.br>
Igor Montagner <igorsm1@insper.edu.br>

Parte 3 – STL e referências

STL

A STL (standard template library) oferece vários recursos interessantes para programação:

1. containers
2. smart pointers
3. geração de números aleatórios

Containers STL

Coleções de dados STL

Estruturas de dados prontas para uso

1. vector -- vetor que cresce dinamicamente
2. array -- tamanho fixo
3. pair -- tupla de dados
4. queue -- fila
5. stack -- pilha
6. unordered_map -- dicionário
7. string

Cada uma garante algumas propriedades

Exemplo Vector

exemplo_vector.cpp ▸ ...

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

int main() {
    std::vector<double> vec_dobl;
    vec_dobl.push_back(5.4);
    vec_dobl.push_back(1.2);

    std::cout << vec_dobl.size() << "\n";
    return 0;

}
```

Estruturas podem ser compostas: `vector<pair<int, string> >` é válido

Coleções de dados STL

- Estruturas de dados prontas para uso
- Cada uma garante algumas propriedades
- Documentação completa em

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

Coleções de dados STL - iteradores

- Objetos usados para percorrer coleções
- Não depende do tipo da coleção

```
for(auto it = vec_dobl.begin(); it != vec_dobl.end(); it++){
    std::cout << *it << "\n"; // acessa elemento
}
```

- Aritmética funciona (`vec_dobl.end() - 1` é o último elemento, `vec_dobl.begin() + 3` é o quarto elemento)

Coleções de dados STL - iteradores

- Para frente: begin()/end()
- Para trás: rbegin()/rend()
- Para frente const: cbegin()/cend()

Referências e memória

Smart pointers

- Gerenciamento de memória é difícil
    - para cada `new` tem que existir um `delete`
- É difícil controlar quando os objetos são criados em uma função e retornados para uso em outros lugares

- **smart pointer**: usa o escopo das variáveis para facilitar alocação de memória

Smart pointers

- `unique_ptr<T>`: Quando a variável deste tipo sair de escopo ela chama delete no seu ponteiro

```
{
    unique_ptr<double> p = unique_ptr<double>(new double);
    *p = 5;
    std::cout << *p << "\n";
    // quando acaba o bloco chama delete automaticamente
}
```

Smart pointers

- `shared_ptr<T>`: Conta quantas referências existem. Se chegar em 0 chama `delete`

```
{
    shared_ptr<double> p = shared_ptr<double>(new double);

    while (1) {
        auto p2 = p;
        // usa p2 aqui
        // p.use_count == 2
    }

    // deleta o dado pois p.use_count == 0 no fim do bloco
}
```

Números aleatórios

Números aleatórios

- Muito úteis em simulações
- Muitas vezes queremos que eles sigam uma certa distribuição estatística
    - Normal
    - Poisson
    - Exponencial
    - etc
- Cabeçalho `<random>`

Números aleatórios

- **Engine**: retorna números pseudo aleatórios baseado em um seed.
- **Distribution**: retorna uma sequência de números que está de acordo com uma distribuição

```
std::default_random_engine eng (100); // gera sempre a mesma ordem
std::normal_distribution<double> distr(10, 2);
for (int i = 0; i < 10; i++) {
    std::cout << distr(eng) << "\n";
}
```

Atividade

Exercício final deve ser entregue como um commit no seu repositório.

Insper

www.insper.edu.br