Conceitos básicos de C++ para maratona de programação

Victor Emanuel Almeida

UNIOESTE

24 de agosto de 2022

Conteúdo

Conceitos I

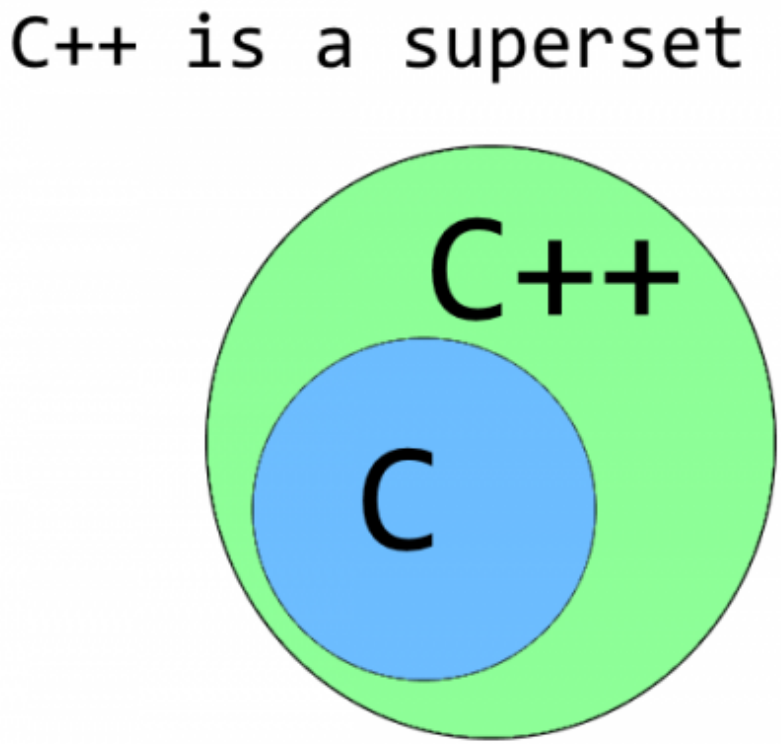
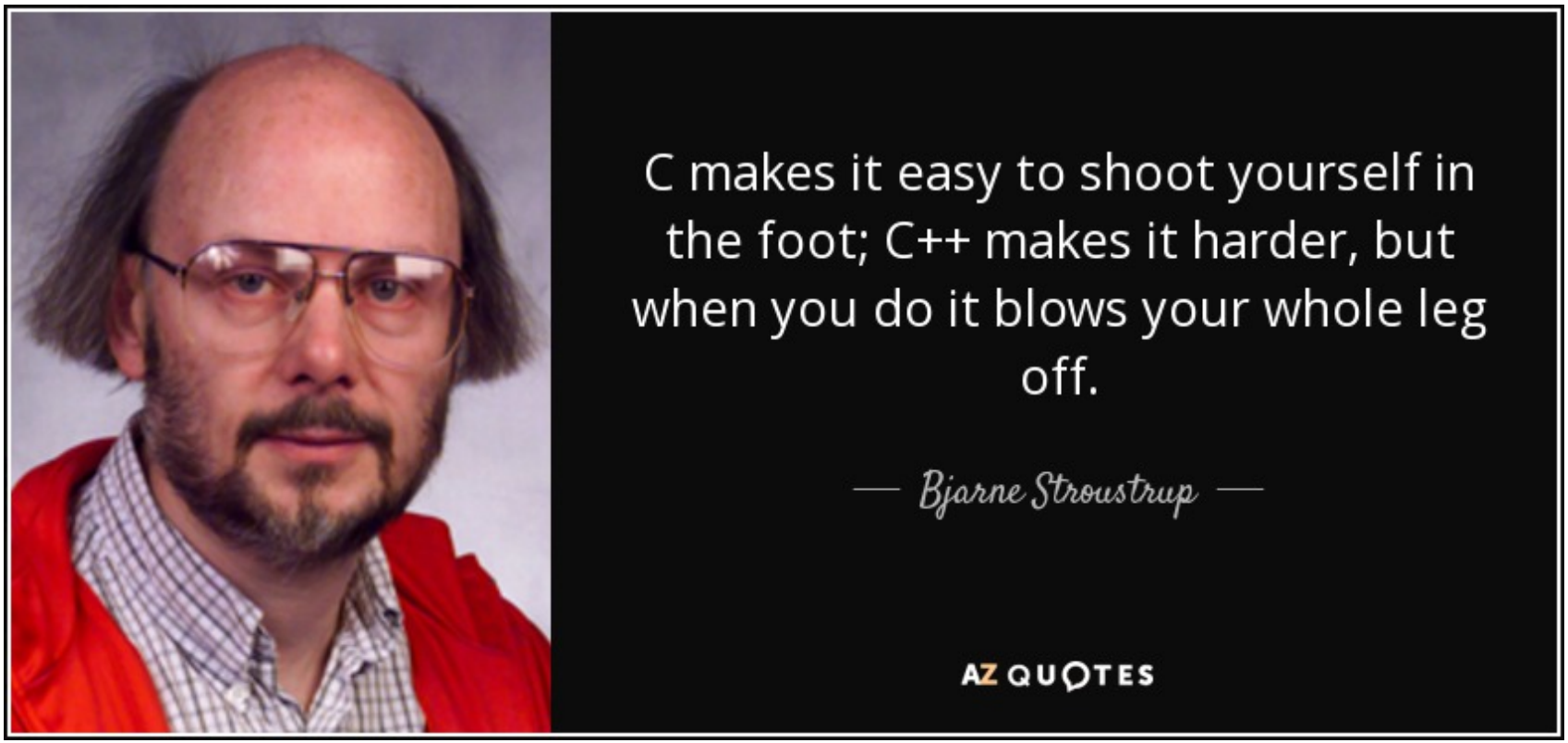


Figura 1: Diagrama de Venn C/C++

Conceitos II



Conceitos III

- Linguagem compilada (g++);
- Fortemente tipada[1];
- Multiparadigma: Imperativa e orientada a objetos[1];
- Linguagem complexa com muitas instruções e palavras reservadas[1].

Vantagens do C++ sobre o C

- Orientação a objetos;
- Entrada e saída de dados;
- Sobrecarga de operadores;
- Referências;
- Alocação de memória e smart pointers;
- Bibliotecas padrão com algoritmos e estruturas de dados.

Exemplo de Ola Mundo Simples

```
1 #include <iostream>
2 int main () {
3     std::cout << "Ola Mundo\n";
4     return 0;
5 }
```

Explicando o << I

- Por padrão o operador << serve como shift left quando aplicado em 2 numeros.
- O operador foi sobrecarregado para quando operar com ostream formatar o dado e imprimir.

Explicando o << II

```
template<typename _CharT, typename _Traits>
  basic_ostream<_CharT, _Traits>&
  basic_ostream<_CharT, _Traits>::
  operator<<(int __n)
  {
    // _GLIBCXX_RESOLVE_LIB_DEFECTS
    // 117. basic_ostream uses nonexistent num_put member functions.
    const ios_base::fmtflags __fmt = this->flags() & ios_base::basefield;
    if (__fmt == ios_base::oct || __fmt == ios_base::hex)
return _M_insert(static_cast<long>(static_cast<unsigned int>(__n)));
    else
return _M_insert(static_cast<long>(__n));
  }
```

Figura 2: Exemplo de sobrecarga de operador arquivo ostram

Exemplo de Ola Mundo Maratona

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 int main () {
4     ios_base::sync_with_stdio(false);
5     cin.tie(0);
6     cout << "Ola Mundo\n";
7     return 0;
8 }
```

Explicando o Ola Mundo

- **#include <bits/stdc++.h>**: Inclui todas as bibliotecas padrão do c++;
- **using namespace std**: Coloca todo o arquivo em um namespace std, sendo assim não precisa usar o std:: antes de utilizar funções e objetos;
- **ios˙base::sync˙with˙stdio(false)**: Desabilita a sincronização entre as streams do c com do c++, retire se for usar printf e scanf;
- **cin.tie(0)**: Desabilita o flush automatico do buffer, retire caso use um cout sem fim de linha e logo em seguida cin;

Dicas

- Resolver um problema da maratona é diferente de resolver uma lista de C1 ou de uma entrevista de emprego:
  - Os bons padrões que aprendemos não importam muito;
  - Pre-alocar a memória (evitar malloc e new);
  - Escrever código direto no main;
- Escrever a menor quantidade de código possível;
- Menor quantidade de estruturas de dados;
- Menor quantidade de funções;
- Testar a complexidade do algoritmo;

Sequenciais I

Vetores: Tem uma "api" idêntica a vetores do C com acesso de um elemento específico, "vetor[i] = valor;"

- **array<T, size>**: Vetor do tipo T de tamanho size definido em tempo de compilação;
- **vector<T>**: Vetor do tipo T de tamanho variável;
- **deque<T>**: Fila duplamente encadeada do tipo T, funciona como o vector porém as ações de inserir e remover são um pouco mais eficientes;

Sequenciais II

Listas encadeadas: Não possui acesso a um elemento específico

- **forward_list<T>**: Lista encadeada do tipo T, so pode ser percorrida do começo para o fim;
- **list<T>**: Lista duplamente encadeada do tipo T, pode ser percorrida dos dois lados;

Interfaces FIFO e LIFO

Interfaces que por padrão utilizam estruturas como deque e list internamente:

- **stack**<**T**>: Pilha do tipo T, funciona como uma pilha de pratos, o último a entrar é o primeiro a sair (*LIFO, Last In First Out*);
- **queue**<**T**>: Fila do tipo T, funciona como uma fila de banco, o primeiro a entrar é o primeiro a sair (*FIFO, First In First Out*);

Associativas I

Implementam estruturas que permitem buscar um elemento de forma eficiente $O(\log(N))$ normalmente utilizando árvores como a rubro-negra ou tabelas hash.

- **set<T>**: Conjunto do tipo T, não permite elementos repetidos, é uma árvore rubro-negra;
- **map<K, V>**: Mapeia a chave do tipo K para um valor do tipo V, não permite chaves repetidas, é uma árvore rubro-negra;

Associativas II

Modificadores de estruturas associativas (multi e unordered):

- **multi**: Permite elementos repetidos, no caso da map chaves repetidas;
- **unordered**: Elementos não são alocados de maneira sequencial na estrutura;

Associativas III

- **set**: Árvore rubro-negra;
- **multiset**: Árvore rubro-negra;
- **unordered_set**: Tabela hash;
- **unordered_multiset**: Tabela hash;
- **map**: Árvore rubro-negra;
- **multimap**: Árvore rubro-negra;
- **unordered_map**: Tabela hash;
- **unordered_multimap**: Tabela hash;

Algoritmos

- **sort(inicio, fim, comp)**: Ordena uma estrutura com acesso a elementos específicos utilizando a função comp;
- **binary_search(inicio, fim, valor, comp)**: Busca um elemento na estrutura de forma eficiente;
- **lower_bound(inicio, fim, valor, comp)**: Busca o menor elemento maior ou igual ao valor;
- **upper_bound(inicio, fim, valor, comp)**: Busca o maior elemento menor ou igual ao valor;
- **min_element(inicio, fim, comp)**: Busca o menor elemento da estrutura;
- **max_element(inicio, fim, comp)**: Busca o maior elemento da estrutura;

Enunciado

Link para o problema: ⟨https://cses.fi/problemset/task/1621⟩

You are given a list of n integers, and your task is to calculate the number of distinct values in the list.

Input: The first input line has an integer n: the number of values. The second line has n integers $x_1\ x_2 \ldots x_n$
Output: Output the number of distinct values.

Exemplos de entradas e saídas

Entrada:
5
2 3 2 2 3
Saída:
2

Entrada:
10
1 1 1 1 1 1 1 1 1 1
Saída:
1

Referências I

1 KHOURI, J. H. E. *Conceitos de Linguagens de Programação Evolução das principais linguagens de programação*. 2020. Acesso em: 21 de maio de 2022.