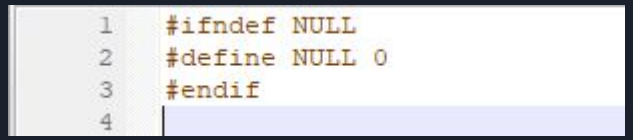
Diretivas de Pré-Processamento e Uso do Namespace em C++

Grupo 14: Arthur Alexandre, João Mateus, José Olívio, Marcos Eduardo e Marcus Vinicius.

Índice

Diretivas de pré-processamento o que é?

As tarefas de pré-processamento são definidas por diretivas de pré-processamento no C++. Todas essas diretivas de pré-processamento se iniciam com o símbolo “#” e não possuem nenhum caractere antes, exceto espaços em branco.



Colocar o símbolo “;” (ponto e vírgula) no final de uma diretiva de pré-processamento é, na verdade, um erro de programação comum.

#define

#ifndef

#include

Como utilizar as Diretivas de pré-processamento?

Como visto anteriormente temos #include , #define e #ifndef .

#include

Todo código em C++ inicia com a diretiva #include<>. Esta diretiva instrui o pré-processador a incluir a biblioteca padrão do C++, #include faz com que uma cópia de um arquivo específico seja incluído no código, É empregada geralmente para incluir arquivos de cabeçalho padrão como por exemplo o <iostream>.

Dois Exemplos de escrita

#include <arquivo>

#include "arquivo"

A primeira forma, com <>, é empregada para a inclusão de arquivos de cabeçalho da biblioteca padrão C++. O pré-processador procura o arquivo especificado em pastas pré-definidas no computador.

Já com as “ “ o pré-processador procura o arquivo primeiramente na mesma pasta (diretório) onde se encontra o arquivo a ser compilado, e então nos diretórios pré-definidos pela implementação do sistema. (Ou seja criado pelo Usuário.

exemplo #Include

```
#include <iostream>
#include <string>
#include <iomanip>
#include <locale>
#include <cmath>
using namespace std;
//incluindo os programas cabeçarios//
#include "prototipo.cpp"

int main(){
    setlocale(LC_ALL, "Portuguese");
    Estacionamento Cliente1;
    string nome,placa;
    //aqui atribuido a horas de entrada//
    int horas,minutos, segundos;
    //aqui atribuido a horas de saida//
    int time, minutes, seconds;
    cout<<"Informe o nome do Dono do Carro: ";
    cin >> nome;
    cout <<"Informe a Placa do Veículo ";
    cin >> placa;
    cout << "Qual Horário de entrada do veículo em Horas: ";
    cin >> horas;
    cout << "Qual Horário de entrada do veículo em Minutos: ";
    cin >> minutos;
    cout << "Qual Horário de entrada do veículo em Segundos: ";
    cin >> segundos;
    cout << "Qual Horário de saida do veículo em Horas: ";
    cin >> time;
    cout << "Qual Horário de entrada do veículo em Minutos: ";
    cin >> minutes;
    cout << "Qual Horário de entrada do veículo em Segundos: ";
    cin >> seconds;
```

Como utilizar as Diretivas de pré-processamento?

#define

Com uma diretiva #define criamos Constantes Simbólicas, que são constantes representadas como símbolos, e Macros, que são pequenas operações funcionais.

Exemplo:

Constante

#define distancia 150 (Com esta declaração, todas as ocorrências da distancia no código são substituídas pelo 150 informado.) **Note que NÃO usamos o operador de atribuição = aqui, pois não se trata de uma atribuição de valores, e sim de uma substituição de nomes.**

Macro

#define area_triangulo(b, h) ((b * h)/2) (toda vez que area_triangulo(b, h) aparecer no código, os valores de b e h serão substituídos pelos valores fornecidos como argumentos, e será retornado o valor calculado da área do triângulo retângulo)

Como utilizar as Diretivas de pré-processamento?

#ifndef

Com uma diretiva de pré-processamento condicional, as expressões são avaliadas e será determinado se o código associado será compilado ou não. O exemplo mais comum de diretiva condicional é a diretiva #ifndef (“IF NOT DEFINED”).

```
#ifndef NULL

 #define NULL 0

#endif
```

Esta diretiva verifica se uma constante simbólica NULL não está definida. Se estiver definida, nada ocorre, mas se ainda não estiver definida, a constante simbólica será criada com o valor 0.

Note que a diretiva #ifndef deve ser fechada (terminada) com #endif. Também existe uma diretiva que verifica se uma constante já está definida, que é a constante #ifdef (“IF DEFINED”), a qual, como se espera, funciona de forma inversa à #ifndef.

exemplo #ifndef e #define

```
1
2   # ifndef PROTOTIPO_H
3   # define PROTOTIPO_H
4
5   //essa parte aqui estou definindo como uma arquivo "cabecario"//
6
7   //aqui estou criando um struct para armazenar o horario de entrada e saida//
8   struct tempo{
9       int hora;
10      int minutos;
11      int segundos;
12
13  };
14  //abrindo uma classe que ira armazenar nome do cliente e a placa//
15  class Estacionamento{
16      string nome;
17      string placa;
18      tempo entrada;
19      tempo saida;
20      float calculo_valor;
21      float cal;
22      float calexit;
23      float horas_ocupadas;
24
25    public:
26       void  setplaca(string);
27       void  setnome(string);
28       void  setentrada(int,int,int);
29       void  setsaida(int,int,int);
30       float getcalculo_valor();
31       void  convert();
32       void  convertexit();
33       Estacionamento();
34      ~Estacionamento();
35       string getplaca();
36       string getnome();
37       float gethoras_ocupadas();
38
39  };
40  #endif
```

Namespaces e Diretivas Using

Namespace

Um namespace (espaço de nomes), é uma forma de organização de funções e outros elementos de um programa. São úteis para remover conflitos de nomes entre recursos distintos, como arquivos de cabeçalho que possuam funções ou métodos de mesmo nome, mas com funcionalidades distintas.

Namespace

Podemos manter os namespaces em arquivos de cabeçalho separados e incluir esses cabeçalhos no código para ter acesso às suas funções quando necessário, a partir de outra parte do programa.

std: Biblioteca Padrão (standard). Indica o namespace padrão da linguagem, que trata de uma série de comandos, funções e declarações distintas.

Pode ser declarado no início do código, logo após as diretivas #include, com o uso da declaração using:

Diretivas Using

As diretivas using habilitam um programa a usar todos os nomes em qualquer cabeçalho C++ padrão, como por exemplo o cabeçalho <iostream>.

Assim, podemos incluir em um programa o namespace std da seguinte forma:

```
using namespace std;
```

E desta forma iremos poupar digitação de código posteriormente quando precisarmos utilizar as funções e métodos deste namespace, pois não será necessário referenciá-lo a todo instante.

Por conta disso, em vez de escrevermos:

```
std::cout<<"Mensagem"
```

podemos escrever simplesmente:

```
cout<<"Mensagem"
```

Pois o compilador já saberá que a função cout pertence ao namespace std, declarado por meio da diretiva using.

Namespaces e Diretivas Using

```cpp
#include <iostream>

using namespace std;

int main() {

  string first_name;

    cout << "Digite seu primeiro nome: ";
    cin >> first_name;
    cout << "Oi, " << first_name << "."<<endl;
    cout << "Bem vindo!";

  return 0;
}
```

Namespaces e Diretivas Using

```cpp
#include <iostream>

int main() {

  std::string first_name;

  std::cout << "Digite seu Primeiro Nome: ";
  std::cin >> first_name;
  std::cout << "Olá" << first_name << "!" << std::endl;
  std::cout << "Bem Vindo!";

  return 0;
}
```

Obrigado, pela atenção !!