Módulo 11 - Pré-processadores

Pré-processadores são comandos pré-processados dentro do código. Serem pré-processados significa que eles são as primeiras coisas que o compilador se preocupa, são processados antes do processamento do código em si, ou seja, pré-processados. Isso permite criar “regras” para a execução do código.

Pré-processadores são chamados por meio de diretivas. Diretivas são palavras-chave da linguagem C que começam com o símbolo #. São elas:

`#if`

`#ifdef`

`#ifndef`

`#else`

`#elif`

`#endif`

`#include`

`#define`

`#undef`

`#line`

`#error`

`#pragma`

#define

Define um identificador e uma string que o substituirá toda vez que for encontrado no arquivo-fonte. É padrão da linguagem utilizar letras maiúsculas para definir um identificador. Veja um exemplo

```
#define CONDICAO 1

if (CONDICAO) {
    printf("Verdadeiro\n");
} else {
    printf("Falso\n");
}
```

Um identificador define também pode armazenar uma string de forma simples, basta declará-la utilizando aspas duplas

```
#define STRING "Isto é uma definicao\n"
printf(STRING);
```

Em relação à utilidade desta funcionalidade, imagine que você é responsável por criar um software que conta itens em uma esteira. Quando a contagem chegar a 10, será emitido um aviso de reposição. A máquina funciona perfeitamente, até que um dia decide-se que o aviso de reposição deverá ser emitido com 15 unidades, e não mais com 10. Uma diretiva `#define AVISO 10` ajudaria neste caso. Tente criar um programa que simule este algoritmo.

```
// Programa simulação de contagem em uma esteira
#include <stdio.h>
#include <unistd.h>

// Para alterar a quantidade de produtos, basta alterar esta definição
#define AVISO 5

int main(void){
    char conta_esteira = 0;

    while(1){
        conta_esteira++;

        printf("%d\n", conta_esteira);

        if (conta_esteira == AVISO) {
        printf("Reposicao\n");
            break;
        }

        sleep(1);
    }
    return 0;
}
```

As diretivas #define também podem se comportar como funções, permitindo a definição de argumentos. A vantagem de utilizar este sistema é a economia de processamento e memória, já que o processador não precisa executar uma chamada de função. Veja um exemplo de uma definição de uma função soma:

```
#include <stdio.h>

#define SOMA(a, b) a + b

int main(void){
    printf("%d", SOMA(1, 5));
```

```
    return 0;
}
```

#include

É uma das diretivas mais importantes. Ela instrui ao compilador incluir outro trecho de código ao projeto. Isso permite incluir bibliotecas de terceiros para aumentar a eficiência de seu código, mas também permite organizar seu próprio código em várias folhas. Ao fazer consumo de bibliotecas, devemos declarar o include assim:

```
#include <stdio.h>
```

Agora, para adicionar outras folhas, utilize aspas:

```
#include "utils.c"
```

A forma mais comum de se trabalhar com C é separando funções em arquivos específicos para elas, ficando o main() na folha principal.

Separando este este código em duas folhas

```
#include <stdio.h>

void swap(int a, int b){
    int temporario;

    temporario = a;
    a = b;
    b = temporario;
    printf("a=%d b=%d\n", a, b);
}

int main(void){
    int a = 10;
    int b = 20;

    printf("a=%d b=%d\n", a, b);
    swap(a, b); // Esta função vai fazer o swap de a e b;
    printf("a=%d b=%d\n", a, b);

    return 0;
}
```

Separe o código em mais folhas de forma que reste somente a função main na folha principal

```
#include <stdio.h>
#include <stdlib.h>

#define SIZE 16

/*
Programa que insere e remove elementos de um array
Ao digitar qualquer número ele insere no array
Se o número for 0 o último elemento do array é excluido
Se for -1 o programa cessa.
*/

// p é o ponteiro atual do array
int *p, array[SIZE];

void push(int i);
int pop(void);

int main(){
  int value;

  do{
    printf("Digite um numero: ");
    scanf("%d", &value);
    if(value!=0) push(value);
    else printf("\nO valor do topo e %d\n", pop());

  }while(value!=-1);
}

void push(int i){
  p++;
  if(p==(array+SIZE)){
    printf("Nao e possivel adicionar mais valores\n");
    exit(1);
  }
  *p = i;
}

int pop(void){
  if(p==array){
    printf("Nao existem valores no array\n");
    exit(1);
  }
  p--;
  return *(p+1);
}
```

#if, #else, #elif, #endif

São diretivas condicionais, funcionam como qualquer condição, mas agem mais profundamente no código, podendo excluir todo um trecho se necessário. Veja um exemplo de código abaixo onde é até possível incluir dois main() no código, pois um deles será completamente removido do código:

```
#include <stdio.h>

// Se exec for diferente de 1, um programa genérico é executado
#define EXEC 1

#if EXEC == 0
int main(void){
    printf("Executando programa 1");
    return 0;
}
#else
int main(void){
    printf("Executando programa generico");
    return 0;
}
#endif
```

Um caso importante é que as diretivas ifs só conseguem processar variáveis que também foram pré-processadas por outras diretivas, devido à ordem da compilação.

Veja outro exemplo de um código que define uma moeda em circulação

```
#define EUA 0
#define INGLATERRA 1
#define BRASIL 2

#define PAIS BRASIL

#if PAIS==EUA
  char moeda[] = "dollar";
#elif PAIS==INGLATERRA
  char moeda[] = "euro";
#else PAIS==BRASIL
  char moeda[] = "real";
#endif
```

As diretivas de compilação não devem ser associadas a variáveis criadas dentro do programa, já que elas são acessadas pelo compilador antes do código de fato. Logo, essas variáveis ainda não existirão quando as diretivas forem compiladas. O código abaixo não mostra o conteúdo dentro das diretivas if pois VALOR não existe no momento da compilação

```
#include <stdio.h>

int main(){
    char VALOR = 10;

#if VALOR == 10
printf("Valor\n");
#endif

printf("Terminou");

    return 0;
}
```

Agora quando adicionamos um `#define VALOR 10` este programa passa a funcionar como esperado

Outra forma que podemos trabalhar isso é com **ifdef e ifndef**. Podemos criar uma diretiva para testar a existência de um valor predefinido antes de começar nosso código de fato. Veja abaixo o funcionamento destas diretivas na prática:

```
#include <stdio.h>

#define VALOR 5

#ifndef VALOR
#define VALOR 10
#endif

int main(){

    if(VALOR == 5) printf("Valor = 5");

    if(VALOR == 10) printf("Valor = 10");

    return 0;
}
```

Também é possível criar ifs que trabalham com a existência de definições, não é necessário criar valores, mas apenas testar a existência deles

```
#define AUTOR "Victor"

#ifndef AUTOR
printf("Faltou definir um autor");
#endif
```

#error

Também é possível forçar erros no compilador. Isto é útil para definir padrões de funcionamento no código. Exemplo:

```
#define AUTOR "Victor"

#ifndef AUTOR
#error Defina um autor
#endif
```