Linguagem C:
diretivas, compilação separada

Prof. Críston

Algoritmos e Programação

Diretivas do pré-processador

- Permite que o programador modifique a compilação
- O pré-processador é um programa que examina e modifica o código-fonte antes da compilação
- As diretivas são os comandos utilizados pelo pré-processador
  - Estes comandos estarão disponíveis no código-fonte, mas não no código compilado
- As diretivas iniciam com #
  - Ex.: #include <stdio.h>

Diretiva #include

- Permite inserir um arquivo qualquer no código-fonte
- A diretiva include é substituída pelo conteúdo do arquivo
- Quando usamos <> para indicar o arquivo, este arquivo é procurado somente na pasta include (no linux, /usr/include)
- Quando utilizamos “” para indicar o arquivo, este arquivo é procurado na pasta atual, e se não for encontrado é procurado na pasta include

Diretiva #include

```
int soma(int a, int b)
{
    return a+b;
}
```

arquivo.c

```
#include "arquivo.c"

int main()
{
    soma(1, 2);
    return 0;
}
```

principal.c

prompt$ gcc principal.c

Diretiva #define

- Permite definir constantes sem consumir memória durante a execução
- Não use o sinal de atribuição (=) !

```
#define PI 3.14
int main()
{
    double raio = 1.0;
    double area = PI * raio * raio;
    ...
    return 0;
}
```

Diretiva #define

- Permite definir trechos fixos de código

```
#define ERRO printf("Ocorreu um erro\n"); exit(1);

int main()
{
    ERRO;
    ...
    return 0;
}
```

Diretiva #define

- Permite definir trechos de código com parâmetros (macros)
- Não pode ter espaços no identificador. Ex.: SOMA (x,y)

```
#define SOMA(x,y) x + y
int main()
{
    int a = SOMA(1, 2);
    double b = SOMA(1.0, 2.0);
    ...
    return 0;
}
```

Diretiva #define

- Recomenda-se usar parênteses em macros..

```
#define SOMA(x,y) x + y
int main()
{
    printf("%d\n", 10 * SOMA(1,2));
    ...
    return 0;
}
```

Diretiva #define

- Recomenda-se usar parênteses em macros..

```
#define SOMA(x,y) x + y
int main()
{
    printf("%d\n", 10 * SOMA(1,2));
    ...
    return 0;
}

// solução:  #define SOMA(x,y) (x + y)
```

Diretiva #define

- Recomenda-se usar parênteses em macros..

```
#define PRODUTO(x,y) (x * y)
int main()
{
    printf("%d\n", PRODUTO(2+3, 4));
    ...
    return 0;
}
```

Diretiva #define

- Recomenda-se usar parênteses em macros..

```
#define PRODUTO(x,y) (x * y)
int main()
{
    printf("%d\n", PRODUTO(2+3, 4));
    ...
    return 0;
}

// solução:  #define PRODUTO(x,y) ((x) * (y))
```

Diretiva #undef

- Remove a definição criada com #define

```
#define TAM_STRING 20

...

#undef TAM_STRING
#define TAM_STRING 100
```

Compilação condicional

```
#define DEBUG

int main()
{
    ...
#ifdef DEBUG
    printf("Descricao detalhada: ...\n");
#else
    printf("Nenhuma descricao\n");
#endif
    ...
}
```

Compilação condicional

```
#define DEBUG 1

int main()
{
    ...
#if DEBUG == 1
    printf("Descricao detalhada: ...\n");
#elif DEBUG == 2
    printf("Descricao resumida: ...\n");
#else
    printf("Nenhuma descricao\n");
#endif
    ...
}
```

Compilação condicional

- Podemos fazer a definição na linha de comando no momento da compilação

```
prompt$  gcc -D DEBUG programa.c
```

```
prompt$  gcc -D DEBUG=2 programa.c
```

Compilação separada

- Podemos seperar nosso programa em vários arquivos utilizando a diretiva #include
- Porém, seria interessante compilar apenas os arquivos que foram modificados desde a última compilação
  - Sistemas grandes podem levar muitos minutos para compilar
- Podemos compilar cada arquivo .c separadamente, produzindo um arquivo .o para arquivo .c
  - Ex.: prompt$ gcc -c arquivo.c
- Para gerar o executável, devemos então passar os arquivos .o
  - Ex.: prompt$ gcc principal.c arquivo.o

Compilação separada

```
int soma(int a, int b)
{
    return a+b;
}
```

arquivo.c

```
int soma(int a, int b);

int main()
{
    printf("%d\n", soma(1, 2));
    return 0;
}
```

principal.c

É necessário inserir o protótipo da função soma em principal.c.

Arquivo de cabeçalho (.h)

- Toda vez que um programa utilizar o código contido em arquivo.c, deverá inserir os protótipos das funções e declarar as variáveis com extern..
- Para simplificar, para cada arquivo compilado separadamente criamos um arquivo de cabeçalho contendo
  - os protótipos das funções,
  - declaração dos novos tipos (struct, enum..)
  - constantes (#define)

Compilação separada

```
#include “arquivo.h”

int soma(int a, int b)
{
    return a+b;
}
```

arquivo.c

```
int soma(int a, int b);
```

arquivo.h

```
#include “arquivo.h”

int main()
{
    x = 2.0;
    printf(“%d\n”, soma(1, 2));
    return 0;
}
```

principal.c