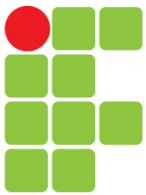
Pré-processador de C

Programação de computadores II

Prof. Renan Augusto Starke

Instituto Federal de Santa Catarina – IFSC
Campus Florianópolis
renan.starke@ifsc.edu.br

13 de fevereiro de 2020



INSTITUTO FEDERAL
SANTA CATARINA

Ministério da Educação
Secretaria de Educação Profissional e Tecnológica
INSTITUTO FEDERAL DE SANTA CATARINA

Tópicos da aula

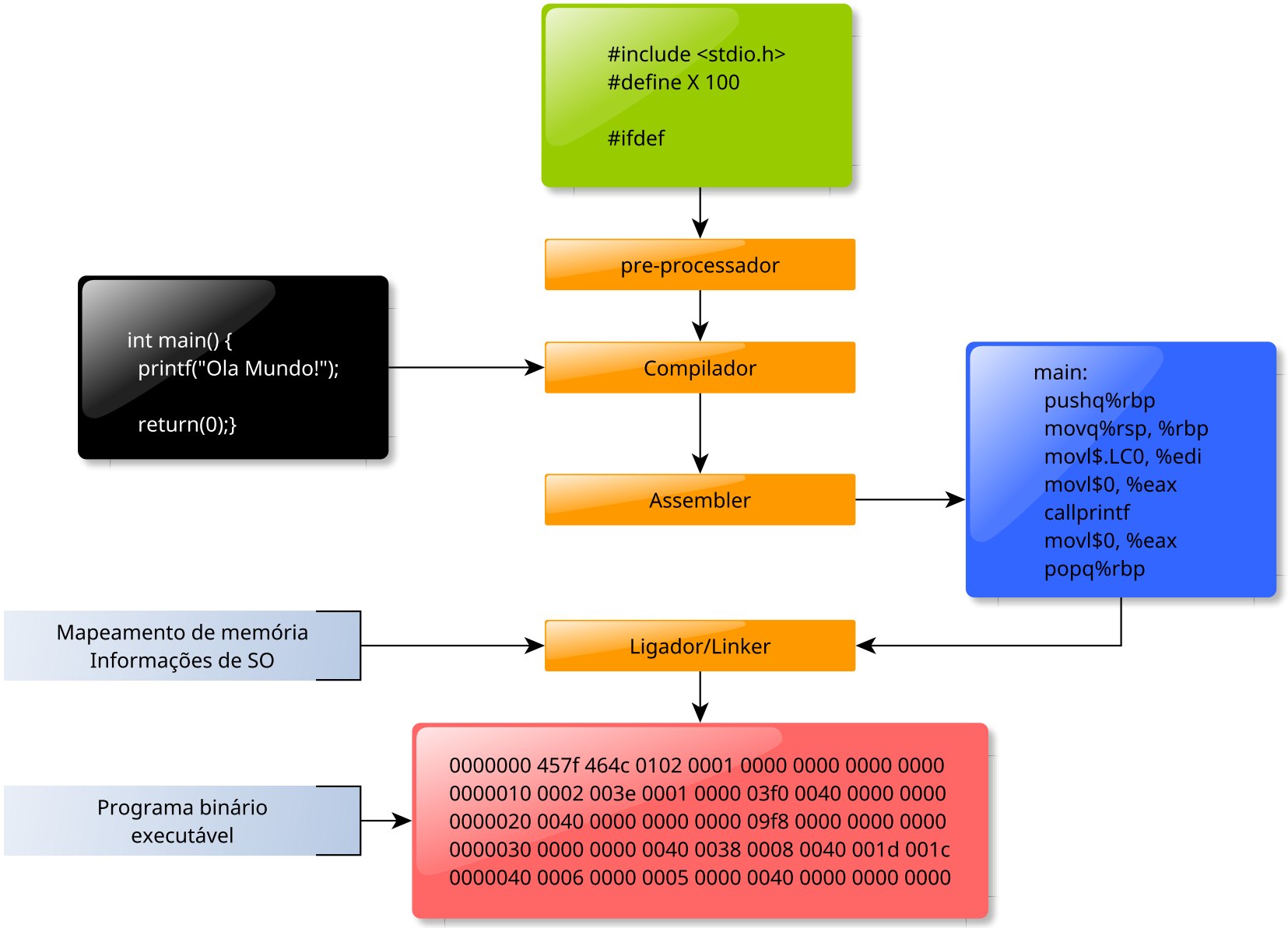
Tópico

Pré-processador

Diretivas do pré-processador:

- Processa o código-fonte antes do compilador, trocando expressões
  - Melhora leitura código
  - Inclusão de cabeçalho de bibliotecas
  - Depuração
  - Portabilidade entre plataformas
- Sintaxe é diferente de C
- Todas diretivas do pré-processador iniciam com #
- Diferente da linguagem, não é de formato livre
- Principais comandos
- **#define, #include, #ifdef, #ifndef**

Pré-processador



Pré-processador

- Para analisar o código após o pré-processador, sem compilá-lo:

  gcc -E arquivo.c

- Opção -E do gcc:

  -E: *Stop after the preprocessing stage; do not run the compiler proper. The output is in the form of preprocessed source code, which is sent to the standard output.*

Saída do pré-processador

```
#define TAMANHO 20     /* agora, em 20 elementos */

int dado[TAMANHO];     /* vetor de dados */
int dobro[TAMANHO];    /* vetor com dados vezes 2 */

int main()
{
    int index;    /* indice do vetor */

    for (index = 0; index < TAMANHO; ++index) {
            dado[index] = index;
            dobro[index] = index * 2;
    }
    return (0);
}
```

Saída do pré-processador

```
# 1 "exemplo_define.c"
# 1 "<built-in>"
# 1 "<command-line>"
# 1 "/usr/include/stdc-predef.h" 1 3 4
# 1 "<command-line>" 2
# 1 "exemplo_define.c"

int dado[20];
int dobro[20];

int main()
{
    int index;

    for (index = 0; index < 20; ++index) {
      dado[index] = index;
     dobro[index] = index * 2;
    }
    return (0);
}
```

Tópico

#include

- Incluir conteúdo (texto) de arquivos externos
- Pode ser qualquer arquivo, porém utiliza-se para:
  - Cabeçalhos – header files – .h
  - Cabeçalhos de bibliotecas: *#include <stdio.h>*
  - Cabeçalhos e definições locais
- Se #include apontar para uma biblioteca padrão do compilador:
  - Utiliza-se < >: #include <stdlib.h>
  - ATENÇÃO: #include não resolve problemas de bibliotecas binárias: #include <math.h> não é suficiente.
- Inclusão de arquivos locais:
  - Utiliza-se " ": #include "meu_include.h"
- Diretório de *include* pode ser modificado por opção -I do compilador: **gcc -I <meu diretorio >**

Tópico

#define

- #define permite definir "definições" para serem substituídas pelo o pré-processador
- Melhora leitura e portabilidade do código
- Utilizado macros e "constantes"

  Forma geral:

  #define NOME texto_substituto
  - NOME: qualquer identificador válido
  - texto_substituto: qualquer texto
- Por padronização, **NOME** SEMPRE EM MAIÚSCULO

Exemplos #define

```
int dado[10];     /* vetor de dados */
int dobro[10];    /* vetor com dados vezes 2 */

int main()
{
    int index;    /* indice do vetor */

    for (index = 0; index < 10; ++index) {
        dado[index] = index;
        dobro[index] = index * 2;
    }
    return (0);
}
```

Exemplos #define

```c
#define TAMANHO 20    /* agora, em 20 elementos */

int dado[TAMANHO];    /* vetor de dados */
int dobro[TAMANHO];   /* vetor com dados vezes 2 */

int main()
{
    int index;   /* indice do vetor */

    for (index = 0; index < TAMANHO; ++index) {
      dado[index] = index;
      dobro[index] = index * 2;
    }
    return (0);
}
```

Exemplos #define – não recomendado

```c
#define FOR_10  for (i=0; i<10; i++)

/*codigo principal*/
int i;
FOR_10
{
  printf("%d\n", i);
}
```

Exemplos #define – erro compilação

```
// definicao de 10 elevado a 10
#define BIG_NUMBER 10 ** 10

int main()
{
    int index = 0;

    while (index < BIG_NUMBER) {
        index = index * 8;
    }
    return (0);
}
```

Exemplos #define – erro compilação

```
// definicao de 10 elevado a 10
#define BIG_NUMBER 10 ** 10

int main()
{
    int index = 0;

    while (index < BIG_NUMBER) {
        index = index * 8;
    }
    return (0);
}
```

- Pré-processador não conhece sintaxe, erro na compilação: **index < 10 ** 10**

Exemplos #define – erro semântico

```c
#include <stdio.h>

#define FIRST_PART      7
#define LAST_PART       5
#define ALL_PARTS       FIRST_PART + LAST_PART

int main() {
    printf("O quadrado de todas as partes vale %d\n",
  ALL_PARTS * ALL_PARTS);
    return (0);
}
```

Exemplos #define – erro semântico

```c
#include <stdio.h>

#define FIRST_PART      7
#define LAST_PART       5
#define ALL_PARTS       FIRST_PART + LAST_PART

int main() {
    printf("O quadrado de todas as partes vale %d\n",
  ALL_PARTS * ALL_PARTS);
    return (0);
}
```

Saída pre-processador:

```c
# 7 "define_composto.c"
int main() {
    printf("O quadrado de todas as partes vale %d\n",  7 + 5 * 7 + 5);
    return (0);
}
```

Exemplos #define – erro semântico

```c
#include <stdio.h>

#define FIRST_PART      7
#define LAST_PART       5
#define ALL_PARTS       FIRST_PART + LAST_PART

int main() {
    printf("O quadrado de todas as partes vale %d\n",
  ALL_PARTS * ALL_PARTS);
    return (0);
}
```

Saída pre-processador:

```c
# 7 "define_composto.c"
int main() {
    printf("O quadrado de todas as partes vale %d\n",  7 + 5 * 7 + 5);
    return (0);
}
```

Correto: **#define ALL_PARTS (FIRST_PART + LAST_PART)**

Exemplos #define – erro semântico

```c
#include <stdio.h>

#define SIZE    10;
#define SIZE_2  (SIZE -2);

int main()
{
    int tamanho;

    tamanho = SIZE_2;
    printf("O tamanho vale %d\n", tamanho);
    return (0);}
```

Exemplos #define – erro semântico

```c
#include <stdio.h>

#define SIZE    10;
#define SIZE_2  (SIZE -2);

int main()
{
    int tamanho;

    tamanho = SIZE_2;
    printf("O tamanho vale %d\n", tamanho);
    return (0);}
```

Saída pre-processador:

```c
int main()
{
    int tamanho;

    tamanho = (10; -2);;
    printf("O tamanho vale %d\n", tamanho);
    return (0);
}
```

Exemplos #define – erro semântico

```c
#include <stdio.h>

#define SIZE    10;
#define SIZE_2  (SIZE -2);

int main()
{
    int tamanho;

    tamanho = SIZE_2;
    printf("O tamanho vale %d\n", tamanho);
    return (0);}
```

Saída pre-processador:

```c
int main()
{
    int tamanho;

    tamanho = (10; -2);;
    printf("O tamanho vale %d\n", tamanho);
    return (0);
}
```

Correto: **#define SIZE 10** e **#define (SIZE -2)**

Tópico

#ifdef / #endif

- Permite mudar o código conforme uma definição
- Exemplo para depuração:

```
#ifdef DEBUG
  printf("##Indice de x: %d\n", indice);
#endif
```

- Para ativar/desativar, basta definir #DEBUG

```
/*ativa informacoes de debug*/
#define DEBUG

/* para desativar */
#undef DEBUG
```

- Pode ser defino através de linha de comando: **gcc -D DEBUG < arquivo.c >**

#ifdef / #endif

```c
#include <stdio.h>

#define FIRST_PART      7
#define LAST_PART       5
#define ALL_PARTS       (FIRST_PART + LAST_PART)

int main() {
    printf("O quadrado de todas as partes vale %d\n",
  ALL_PARTS * ALL_PARTS);

#ifdef DEBUG
  printf("##FIRST_PART: %d\n", FIRST_PART);
#endif


    return (0);
}
```

gcc -D DEBUG ifdef.c -o ifdef

#ifndef / #endif

- Inclui trecho se **não** houver definição

```
#ifndef DEBUG
  printf("Codigo normal, sem debug\n");
#endif
```

- #else

```
#ifdef DEBUG
  printf("Versão de teste. Debug ativado\n");
#else
  printf("Código normal, sem debug\n");
#endif
```

- Pode ser defino através de linha de comando: **gcc -D DEBUG < arquivo.c >**

Tópico

Personalização com macros

- Macros podem receber parâmetros
- #define QUADR(x) ((x) * (x))
- QUADR(3) é expandido para ((3) * (3))
- IMPORTANTE: parênteses

Personalização com macros

```c
#include <stdio.h>
#define SQR(x) (x * x)

int main()
{
    int counter;

    for (counter = 0; counter < 5; ++counter)
    {
        printf("x: %d, x ao quadrado: %d\n",
            counter+1, SQR(counter+1));
    }
    return (0);
}
```

macros.c

Personalização com macros

#define SQR(x) (x * x)

```
int main()
{
    int counter;

    for (counter = 0; counter < 5; ++counter)
    {
        printf("x: %d, x ao quadrado: %d\n",
            counter+1, (counter+1 * counter+1));
    }
    return (0);
}
```

Personalização com macros

#define SQR(x) (x * x)

```
int main()
{
    int counter;

    for (counter = 0; counter < 5; ++counter)
    {
        printf("x: %d, x ao quadrado: %d\n",
            counter+1, (counter+1 * counter+1));
    }
    return (0);
}
```

#define SQR(x) ((x) * (x))

```
int main()
{
    int counter;

    for (counter = 0; counter < 5; ++counter)
    {
        printf("x: %d, x ao quadrado: %d\n",
            counter+1, ((counter+1) * (counter+1)));
    }
    return (0);
}
```

Inclusão de arquivos de cabeçalho (.h)

- Macros também são importantes para evitar redefinição de tipos e funções em arquivos de cabeçalho quando incluídos vários módulos:

```c
#include "fila.h"

int main()
{
   fila_t *fila;

   fila = cria_fila(10);

   (...)
}
```

```c
#ifndef FILA_H_INCLUDED
#define FILA_H_INCLUDED

typedef struct filas fila_t;

int dequeue(fila_t* fila);
void enqueue(int x, fila_t* fila);

fila_t* cria_fila(int tamanho);
void libera_fila(fila_t* fila);

#endif // FILA_H_INCLUDED
```

Tópico

Exercício

Implementar e testar seguintes macros:

- Atribuir nível alto em um bit específico em uma palavra/variável:

```
#define SET_BIT(Y, BIT) Y = Y & (1 << BIT)
```

- Atribuir nível baixo em um bit específico em uma palavra/variável:

```
#define CLR_BIT(Y, BIT)
```

- Complementar um bit específico em uma palavra/variável:

```
#define CPL_BIT(Y, BIT)
```

- Testar um bit específico em uma palavra/variável:

```
#define #TST_BIT(Y, BIT)
```