Funções Parte II

Macro
Asserções
Escopo de Variáveis
Variáveis Locais x Variáveis Globais
Ponteiros - Conceitos Básicos
Passagem de Parâmetros por Referência em Funções
Exercício

Macro

- Macro é um fragmento de código com determinado nome.
- Macro é definido em uma diretiva #define para o pré-processador.
- Toda vez que o nome é utilizado, ele é substituído pelos conteúdos que formam a Macro.
- Macro sem argumentos é processada como constante simbólica:
  - #define BUFFER_SIZE 1024
- Exemplo:

```
a = X;
#define X 10  ⟹
b = X;
```

```
a = X
b = 10
```

Macro

- Macro com argumentos é **expandida**, onde o texto substituído entra no lugar do identificador da macro e da sua lista de argumentos.
- Exemplo:

```
#define AREA_RET(c,l) ( (c) * (l) )
…..
areaRet = AREA_RET(x+10, y+20);
```

```
#define AREA_RET(c,l) ( (c) * (l) )
…..
areaRet = ( (x+10) * (y+20) );
```

Macro

- **Cuidado!!!**

```
#define Fun_A() printf(“ERRO”)
…..
FUN_A();
```

⇕

```
#define Fun_A() printf(“ERRO”)
…..
printf(“ERRO);
```

```
#define Fun_A  ()  printf(“ERRO”)
…..
FUN_A();
```

⇕

```
#define Fun_A()  printf(“ERRO”)
…..
()  printf(“ERRO”)();
```

Asserções

- Um programa pode utilizar asserções para facilitar o processo de programação.
- A macro assert, definida em <assert.h>, é utilizada para determinar se o valor de uma expressão é falso (0).
- A macro chama a função **abort** para terminar a execução do programa, caso a expressão seja falsa.
- Exemplo:

```
int main(void){
    int a, b, c;
    scanf(%d%d, &a, &b);
    ....
    c = f(a,b)
    assert(c>0);
    ....
}
```

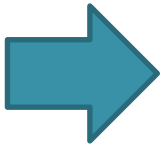
Escopo de Variáveis

- Estabelece onde uma variável poderá ser utilizada em um programa.
- A regra básica envolvendo escopo é que os identificadores são acessados apenas dentro do bloco em que foram declarados.
- Os identificadores não são conhecidos fora dos limites do bloco onde foram declarados.
- Programadores podem escolher utilizar um mesmo identificador em diferentes declarações.
- Neste caso, qual objeto está sendo utilizado?

Escopo de Variáveis

- **Exemplo: Blocos aninhados**

```
{
 int a=2;
 printf("%d\n",a); /* 2 é exibido*/
 {
    int a = 5;
    printf("%d\n",a); /* 5 é exibido */
 }
 printf("%d\n", ++a); /*3 é exibido*/
}
```



```
{
 int a_outer=2;
 printf("%d\n",a_outer);
 {
    int a_inner = 5;
    printf("%d\n",a_inner);
 }
 printf("%d\n", ++a_outer);
}
```

Exemplo: Blocos aninhados

```
{
  int a=1, b=2, c=3;
  printf("%3d%3d%3d\n",a,b,c);        /* 1  2  3  */
  {
    int b=4;
    float c=5.0;
    printf("%3d%3d%5.1f\n",a,b,c);     /* 1  4  5.0  */
    a=b;
    {
      int c;
      c=b;
      printf("%3d%3d%3d\n",a,b,c);     /* 4  4  4  */
    }
    printf("%3d%3d%5.1f\n",a,b,c);     /* 4  4  5.0  */
  }
 printf("%3d%3d%3d\n",a,b,c);          /* 4  2  3  */
}
```

Escopo de Variáveis

- **Exemplo: Blocos em paralelo**

```
{
  int a, b;
  ....
  {                    /* bloco interno 1*/
    float b;
    ....          /* int a é conhecido, mas int b não*/
  }
.....
{                      /* bloco interno 2*/
   float a;
   ....               /* int b é conhecido, mas int a não */
                      /* ninguém do bloco 1 é conhecido */
}
....
}
```

Variáveis locais x Variáveis globais

- **Variáveis Locais**
  - Tem como escopo a função onde foi declarada.
  - Os nomes e valores dessas variáveis tem uso restrito à função que declarou estas variáveis.

- **Variáveis Globais**
  - Nomes e valores dessas variáveis podem ser acessados em todo o programa principal.
  - Essas variáveis devem ser declaradas fora do corpo de todos os procedimentos ou funções do programa.
  - O programa pode alterar uma variável global em qualquer ponto, tornando difícil localizar a alteração que possa ter ocasionado erro.

- **Exemplo**:

```
#include <stdio.h>
int a=1, b=2, c=3;              /* variáveis globais */
int f(void);                /* protótipo da função*/
int main(void)
{
  printf("%3d\n", f( ));          / 12 é exibido */
  printf("%3d%3d%3d\n", a,b,c);     / 4 2 3 são exibidos*/
  return 0;
}


inf f(void)
{
  int b, c;             /* b e c são variáveis locais */
  a=b=c=4;
  return (a+b+c);
}
```

Ponteiros - Conceitos Básicos

- Uma **variável** é armazenada em um certo número de bytes em uma determinada posição de memória.

- Um **ponteiro** é uma variável que contém o endereço de outra variável.

- Os **ponteiros** são usados para acessar e manipular conteúdos em determinado endereço de memória.

Ponteiros - Conceitos Básicos

- Acesso ao endereço de memória da variável:
  &<nome_var>

- Declaração de um ponteiro:
  <tipo> *<nome_var_ponteiro>

- Atribuição de um ponteiro:
  <ponteiro_tipoX> = & <variável_tipoX>

Ponteiros - Conceitos Básicos

- Exemplos de declaração de ponteiros:

```
int *p;
 char *q;
 float *r,*s;
```

- Exemplos atribuição de ponteiros:

```
p = 0;
 p= NULL; /* equivalente a p = 0 */
 p = &i;
 p = (int *) 1776;  /* valor absoluto do endereço*/
```

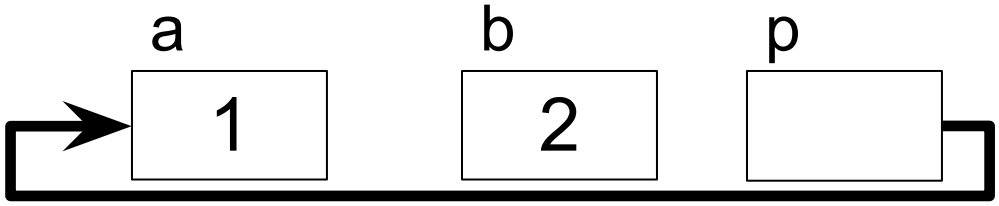
Ponteiros - Conceitos Básicos

- O operador unário * representa indireção.
- Se p é um ponteiro, então *p é o valor da variável da qual p é o endereço.
- O valor direto de p é o endereço de memória.
- *p é o valor indireto de p, pois representa o valor armazenado no endereço de memória.

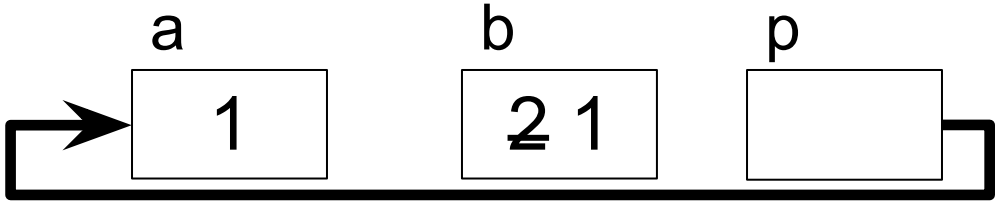
Ponteiros - Conceitos Básicos

int a=1, b=2, *p;

p=&a



b = *p; ⇔ b=a;



Ponteiros - Conceitos Básicos

Exemplo:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   int i=7, *p = &i;
   printf("%s%d\n%s%p\n", " Valor de i: ", *p,
        "Endereco de i:", p);
   return 0;
}
```

```
Valor de i: 7
Endereco de i: 0028FF44
```

Passagem de parâmetros por referência em funções

- O endereço de memória da variável é fornecido à função e não uma cópia do valor da variável.
- Qualquer alteração executada pela função ocorre na posição de memória fornecida.
- Portanto, as alterações permanecem quando a função é encerrada.
- Ponteiros, são usados nas passagens por referência.

Exemplo:

```
#include <stdio.h>
void swap(int *, int *);
int main(void)
{
      int i=3, j=5;
     swap(&i, &j);
     printf("%d   %d\n", i, j);  /* 5  e 3 são exibidos */
     return 0;
}
void swap (int *p, int *q)
{
       int tmp;
       tmp = *p;
       *p = *q;
        *q = tmp;
}
```

Passagem de parâmetros por referência em funções

- As variáveis i e j são passadas por referência, ou seja, o endereço de memória das variáveis é repassado à função.

swap (&i, &j)

- Os ponteiros *p e *q, declarados no argumento na função swap, passam a referenciar a posição de memória das variáveis i e j.

void swap (int *p, int *q)

Exercício

Escreva um programa que receba um número inteiro representando a quantidade total de segundos e, usando passagem de parâmetros por referência, converta a quantidade informada de segundos em Horas, Minutos e Segundos. Imprima o resultado da conversão no formato HH:MM:SS. Utilize o seguinte protótipo de função:

void converteHora(int total_segundos, int *hora, int *min, int *seg);