SSC0101 - ICC1 – Teórica

Introdução à Ciência da Computação I

Modularização de Programas

Parte I

Prof. Vanderlei Bonato: vbonato@icmc.usp.br

Prof. Claudio Fabiano Motta Toledo: claudio@icmc.usp.br

Sumário

- Modularização de programas
- Sub-rotinas em algoritmos
- Funções em linguagem C
  - Return
  - Void
- Declarações de Funções

Modularização de programas

- Divisão do programa em sub-rotinas (procedimentos) e funções buscando melhorar seu desempenho e facilitar sua manutenção.
- Essas sub-rotinas e funções contêm instruções específicas que podem ser executadas várias vezes pelo programa principal.
- Sub-rotina ou procedimento não retorna valor para o programa principal ao final da execução.
- Funções retornam um valor para o programa principal ao final da execução.

Sub-Rotinas em Algoritmos

- SUB-ROTINA <nome_rotina>(<lista_parametros>)

  .....

  FIM_SUB_ROTINA<nome_rotina>

- SUB-ROTINA <nome_rotina>(<lista_parametros>)

  .....

  RETORNE <valor>

  FIM_SUB_ROTINA<nome_rotina>

Sub-Rotinas em Algoritmos

- Exemplo:

```
ALGORITMO
DECLARE sal, aum NUMÉRICO
LEIA sal
aum ← calculo(sal)
sal ← sal + aum
FIM_ALGORITMO

SUB-ROTINA calculo(sal NUMÉRICO)
   DECLARE perc, valor NUMÉRICO
   LEIA perc
   valor ← sal*per/100
   RETORNE valor
FIM_SUB_ROTINA calculo
```

Funções em linguagem C

```
<tipo_retornado>  <nome_função>(<lista_dos_parametos>)
        {
                <declarações>
                <instruções>
        }
```

Exemplo:

```
int factorial (int n)       /* cabeçalho da função*/
{                           /* início do corpo da função*/
   int i, product = 1;
   for (i=2; i<=n; ++i)
           product *= i;
   return product;
 }
```

Return

- As funções retornam um resultado que deve ser do mesmo tipo para o qual a função foi declarada.

<**tipo_retornado**> <nome_função>(<lista_dos_parametos>)

int factorial (int n);

- O comando **return** é responsável por encerrar a execução da função e retornar o valor daquele tipo.

return product;

Return

- Se um tipo não é especificado para uma função, o tipo **int** será o *default*.

```
int all_add( int a, int b)
{
      int c;
      ....
      return  (a+b+c);
}
```

⇔

```
 all_add( int a, int b)
{
      int c;
      ....
      return  (a+b+c);
}
```

Return

- O valor retornado é convertido, se necessário, para o tipo retornando pela função

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

Return

- Recomenda-se limitar a função para que tenha um único **return** visando facilitar a compreensão da função.

- Todavia, o uso de mais que um **return** também pode tornar o código mais legível.

- Desta forma, a quantidade de **return** em uma função deve facilitar o entendimento e a manutenção do código.

Return

- Exemplo:

```
double absolute_value(double x)
{
  if(x>-0.0)
    return x;
  else
    return –x;
}
```

Void

- As sub-rotinas na linguagem C podem ser encaradas todas como funções.

- A palavra reservada **void** na declaração de uma sub-rotina indica que se trata de uma função que não retorna valor.

- O uso de **void** no lugar de uma lista de parâmetros indica que a função não utiliza argumentos (lista de parâmetros).

Void

- Exemplos:

```
void nadafaz(void) { }

void wrt_endereço(void){
 printf("%s\n%s\n%s\n%s\n%s\n\n",
    "                    **************************",
    "                     ***   SANTA CLAUS  *",
    "                      ***   NORTH POLE    *",
    "                      ***   EARTH         *",
    "                    **************************");
}
```

Void

- Os trechos abaixo são equivalentes:

  void func() ⇔ void func(void)

- A declaração abaixo, considerando a linguagem C tradicional, significa que o número de argumentos da função não é conhecido.

  int func();

- Isso ocorre pelo fato de **void** não ser uma palavra reservada na linguagem C tradicional.

Declarações de funções

- Exemplos:

```
float func(x,  y) /*  C  tradicional */
int x, float y;
{...
     <corpo_da_função>
...}
```

```
float func(int x,  float y) /* ANSI C  tradicional */

{...
     <corpo_da_função>
...}
```

Declarações de funções

- Exemplo: a chamada teste(1) será executada da mesma forma nas funções declaradas abaixo?

```
int teste(x) double x;
{...}
```

```
int teste(double x)
{...}
```

Declarações de funções

- As declarações de funções são geradas de várias formas para o compilador:
  - Quando a função é invocada
  - Quando a função é definida
  - Quando a função é explicitamente declarada
  - Quando o protótipo da função é declarado

Declarações de funções

- Um protótipo da função indica ao compilador o número e o tipo de argumentos que devem ser passados para a função e o tipo de valor que deve ser retornado pela função.

<**tipo_retornado**> <nome_função>(<lista_dos_parametos>);

- A lista dos parâmetros apresenta os tipos separados por vírgula, onde os identificadores são opcionais.

float func(int, float); ⇔ float func(int x, float y);

Declarações de funções

- Se uma função, por exemplo, func(x) é chamada antes de sua declaração, definição ou protótipo, o compilador assume a declaração abaixo como default

```
int func();
```

- A maioria dos compiladores precisa conhecer os tipos de retorno e os parâmetros, antes que o programa principal faça uma chamada à sub-rotina.

Declarações de funções

- Programas maiores teriam grande quantidade de linhas inicialmente dedicadas às funções e procedimentos até se chegar à função main().

- Além disso, ficaria cada vez mais difícil manter as funções na ordem correta de acordo com sua chamada pelo programa principal.

- A linguagem C permite que protótipos de funções sejam declarados antes do programa principal.

Declaração de funções

- Exemplo1:

```
#include <stdio.h>
#define N 7
long   power(int, int);
void   prn_heading(void);
void   prn_tbl_of_powers(int);

int main(void)
{
 prn_heading();
Prn_tbl_of_powers(N);
return 0;
}
```

Declarações de funções

```
void prn_heading(void)
{
   printf("\n::: A TABLE OF POWERS :::\n\n");
}

void prn_tbl_of_powers(int n)
{
       int i,j;
      for(i=1; i<=n; ++i){
       for(j=1; j<=n; ++j)
           if(i==j)
              printf("%ld", power(i,j));
           else
               printf("%9ld",power(i,j));
       putchar('\n');
      }
}
```

```
void power(int m, int n)
{
       int i;
       long product = 1;

      for(i=1; i<=n; ++i){
         product *=m;

      return product;
}
```

Declarações de funções

- Exemplo2:

```
#include <stdio.h>
#define N 7
void prn_heading(void)
{....}
long power(int m, int n)
{....}
long prn_tbl_of_powers(int n)
{...printf("%ld", power(i,j));...}

int main(void)
{
      prn_heading();
      prn_tbl_of_powers(N);
      return 0;
}
```

Declarações de funções

- Exemplos 1 e 2: Saída

::: A TABLE OF POWERS ::::

1	1	1	1	1	1	1
2	4	8	16	32	64	128
3	9	27	81	243	729	2187
.....						

Exercício I

- Usando algoritmo, crie uma sub-rotina que receba três números inteiros **a**, **b** e **c** sendo **a** maior que 1. A função deverá somar todos os inteiros entre **b** e **c** que sejam divisíveis por a (inclusive **b** e **c**) e retornar o resultado para a função principal.

Exercício II

- Crie um código usando linguagem C para imprimir os valores da função f(x) no intervalo [0,3] . O valor de x deve percorrer o intervalo com passos de tamanho 0.1. Uma função deverá ser implementada para retornar os valores de f(x). Os valores de a, b e c são reais e devem ser passados à função via parâmetros.

$f(x) = ax^2+bx+c.$

Referências

Ascencio AFG, Campos EAV. Fundamentos de programação de computadores. São Paulo : Pearson Prentice Hall, 2006. 385 p.

Kelley, A.; Pohl, I*., A Book on C*: programming in C. 4ª Edição. Massachusetts: Pearson, 2010, 726p.

Kernighan, B.W.; Ritchie, D.M. C, *A Linguagem de Programação*: padrão ANSI. 2ª Edição. Rio de Janeiro: Campus, 1989, 290p.

FIM Aula 11