Linguagem C - Sub-Programação

Prof. Antonio L. Apolinário Jr.

UFBA/IM/DCC/BCC 2015.2

Roteiro

- Sub-Programação
- Exercícios

Sub-progamação em C

Sub-programação em C

- Sub-programas em C são funções:
  - sempre um tipo de retorno
    - tipo básico
    - tipo abstrato
    - nenhum tipo
      - `void`
  - Retorno de dentro do corpo da função
    - `return <valor>`

Sub-programação em C

- Funções de bibliotecas do sistema:

Function	Description	Example
`sqrt( x )`	square root of x	`sqrt( 900.0 ) is 30.0` `sqrt( 9.0 ) is 3.0`
`exp( x )`	exponential function e^x	`exp( 1.0 ) is 2.718282` `exp( 2.0 ) is 7.389056`
`log( x )`	natural logarithm of x (base e)	`log( 2.718282 ) is 1.0` `log( 7.389056 ) is 2.0`

`log10( x )`	logarithm of x (base 10)	`log10( 1.0 ) is 0.0`
		`log10( 10.0 ) is 1.0`
		`log10( 100.0 ) is 2.0`

`fabs( x )`	absolute value of x	`fabs( 5.0 ) is 5.0`
		`fabs( 0.0 ) is 0.0`
		`fabs( -5.0 ) is 5.0`

`ceil( x )`	rounds x to the smallest integer not less than x	`ceil( 9.2 ) is 10.0` `ceil( -9.8 ) is -9.0`
`floor( x )`	rounds x to the largest integer not greater than x	`floor( 9.2 ) is 9.0` `floor( -9.8 ) is -10.0`
`pow( x, y )`	x raised to power y (x^y)	`pow( 2, 7 ) is 128.0` `pow( 9, .5 ) is 3.0`
`fmod( x, y )`	remainder of x/y as a floating point number	`fmod( 13.657, 2.333 ) is 1.992`
`sin( x )`	trigonometric sine of x (x in radians)	`sin( 0.0 ) is 0.0`
`cos( x )`	trigonometric cosine of x (x in radians)	`cos( 0.0 ) is 1.0`
`tan( x )`	trigonometric tangent of x (x in radians)	`tan( 0.0 ) is 0.0`

Fig. 5.2 Commonly used math library functions.

Sub-programação em C

- Funções de bibliotecas do sistema:

Standard library header	Explanation
`<assert.h>`	Contains macros and information for adding diagnostics that aid program debugging.
`<ctype.h>`	Contains function prototypes for functions that test characters for certain properties, and function prototypes for functions that can be used to convert lowercase letters to uppercase letters and vice versa.
`<errno.h>`	Defines macros that are useful for reporting error conditions.
`<float.h>`	Contains the floating point size limits of the system.
`<limits.h>`	Contains the integral size limits of the system.
`<locale.h>`	Contains function prototypes and other information that enables a program to be modified for the current locale on which it is running. The notion of locale enables the computer system to handle different conventions for expressing data like dates, times, dollar amounts and large numbers throughout the world.
`<math.h>`	Contains function prototypes for math library functions.
`<setjmp.h>`	Contains function prototypes for functions that allow bypassing of the usual function call and return sequence.
`<signal.h>`	Contains function prototypes and macros to handle various conditions that may arise during program execution.
`<stdarg.h>`	Defines macros for dealing with a list of arguments to a function whose number and types are unknown.
`<stddef.h>`	Contains common definitions of types used by C for performing certain calculations.
`<stdio.h>`	Contains function prototypes for the standard input/output library functions, and information used by them.
`<stdlib.h>`	Contains function prototypes for conversions of numbers to text and text to numbers, memory allocation, random numbers, and other utility functions.
`<string.h>`	Contains function prototypes for string processing functions.
`<time.h>`	Contains function prototypes and types for manipulating the time and date.

Fig. 5.6 Some of the standard library header.

Sub-programação em C

- Exemplo:

```
#include<stdio.h>

void fun(int a);

int main() {
  fun(10);
}
void fun(int x) {
  printf("%d",x);
}
```

Headers

Cabeçalho da Função

Definição da Função

Chamada da Função

```
#include<stdio.h>

void fun(int x) {
  printf("%d",x);
}

int main() {
  fun(10);
}
```

Passagem de Parâmetros

- Passagem de parâmetros em linguagem C pode ser feita
  - Por Valor
    - Cópia do conteúdo do parâmetro é enviada para a função
  - Por referencia
    - Um ponteiro para o parâmetro é enviado a função

Passagem de Parâmetros

- Exemplos: Passagem por valor

```
void FuncParValor(int k) {
      k = 20;
   }
(...)
int main() {
   int v=10;
   FuncParValor(v);
   printf("%d\n", v);
}
```

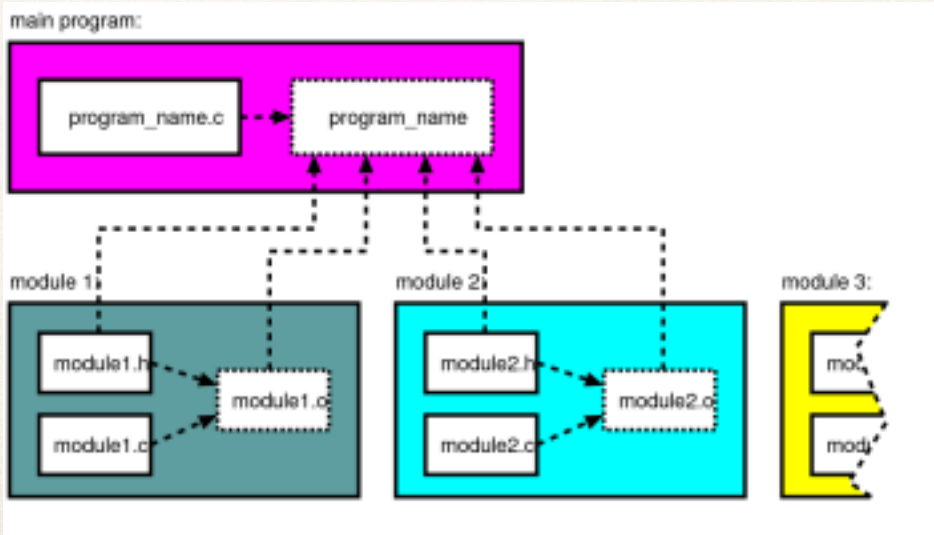
Passagem de Parâmetros

- Exemplos: Passagem por referencia

```
void FuncParRefer(int* k) {
      *k = 20;
   }
(...)
int main() {
   int v=10;
   FuncParValor(&v);
   printf("%d\n", v);
}
```

Modularização

- A estruturação do código em funções permite que um programa seja divido em componentes funcionais denominados módulos



Modularização

- Como fazer:
  - Cada módulo é composto de 2 arquivos
    - xxxx.c => contem as funções do módulo (algoritmos)
      - Não há função **`main()`** em um módulo!
    - xxxx.h => contem a descrição dos dados e funções do módulo (estrutura de dados)

Modularização

- Exemplos:
- módulo pontos: header

```
/* Modulo Ponto */
typedef struct cor      {   int r;
                            int g;
                            int b;
                        } tCor;
typedef struct ponto    {   int ID;
                            float x;
                            float y;
                            float z;
                            tCor   c;
                        } tPonto;

bool preencheVetorAleatorio(tPonto P[], int n);
void imprimePontos(tPonto P[], int n);
void imprimePonto(tPonto P);
```

Modularização

- Exemplos:
- módulo pontos: funções

```
/* Modulo Ponto */

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include "Ponto.h"

bool preencheVetorAleatorio(tPonto* P, int
n) {

int i;
    if (P == NULL)
        return false;
    srand ( time(NULL) );
    for ( i = 0 ; i < n ; i++) {
    P[i].ID = i;
    P[i].x = (float)rand() /
(float)RAND_MAX;
    P[i].y = (float)rand() /
(float)RAND_MAX;
    P[i].z = (float)rand() /
(float)RAND_MAX;

    P[i].c.r =
    P[i].c.g =
    P[i].c.b = 0.0f;
    }
    return true;
}

void imprimePontos(tPonto P[], int n) {
int i = 0;
    if (P == NULL)
        return;
    while (i < n) {
        imprimePonto(P[i]);
        i++;
        }
}
```

Modularização

- Exemplos:
- Usa do módulo ponto

```
/* Funcoes e Passagem de Parametros */

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include "Ponto.h"
#define MAX_VET 10

int main() {
tPonto P[MAX_VET];
    printf("Alocacao Estatica de Memoria\n");
    if (!preencheVetorAleatorio(P, MAX_VET)) {
        printf("Problemas no preenchimento do vetor!!\ncausa provavel => vetor nao
alocado\n");
        exit(-1);
        }
    imprimePontos(P, MAX_VET);
    return 0;
}
```

Exercício

Exercício

1. Desenvolver um Tipo Abstrato de Dados para representação de vetores de inteiros. Esse TAD deve ser criado como um módulo em linguagem C. Como operações básicas devem ser implementadas, no mínimo:
   a) Criação de um vetor vazio;
   b) Inclusão de um novo elemento em um vetor;
   c) Busca de um valor em um vetor;
   d) Remoção de um valor em um vetor;
   e) Concatenação de 2 vetores dados em um terceiro vetor, com repetição de valores;
   f) Concatenação de 2 vetores dados em um terceiro vetor, sem repetição de valores.

Codifique um programa em linguagem C para teste do seu módulo Vetor.