Módulos de Programas em C

Os Módulos em C são chamados de funções. O programador pode escrever funções para definir tarefas específicas e que podem ser utilizadas em muitos locais dos programas. Um dos benefícios mais óbvios de usar funções é que podemos evitar repetição de código.

Funções também são chamadas de procedimento ou sub-rotinas em outras linguagens de programação).

Funções

- A melhor maneira de desenvolver e manter um programa grande é construí-lo a partir de partes menores, ou de módulos, cada um mais facilmente administrável que o programa original.
- Essa técnica é chamada de dividir e conquistar.
- Facilitam o projeto, a implementação, a operação e a manutenção de programas de grande porte.

Funções

As funções são ativadas (chamadas ou invocadas) por uma chamada de função. A chamada da função especifica o nome da função e fornece informações (como argumentos) de que a referida função necessita para realizar a tarefa designada.

Criando funções, um programa C pode ser estruturado em partes relativamente independentes que correspondem as subdivisões do problema.

Funções

Um programa C consiste de uma ou mais definições de funções (e variáveis). Há sempre uma função chamada ***main***. Outras funções também podem ser definidas. Quando nos referirmos a uma função neste texto usaremos a maneira frequentemente utilizada que é o nome da função seguido de ().
O formato geral da definição de uma função é:

tipo-do-resultado nome-da função (lista-de-argumentos)
{
declarações e sentenças
}

Funções simples

Abaixo, mostramos um exemplo simples de um programa que consiste de duas funções: main() e hello().

```
#include <stdio.h>
/* declaracao (protótipo) da funcao hello() */
void hello(void);
/* definicao da funcao main() */
main()
{
hello();
hello();
hello();
}
/* definicao da funcao hello() */
void hello(void)
{
printf("Hello Word! \n");
}
```

Funções simples

Abaixo, mostramos um exemplo simples de um programa que consiste de duas funções: main() e hello().

```
#include <stdio.h>
/* declaracao (protótipo) da funcao hello() */
void hello(void);
/* definicao da funcao main() */
main()
{
hello();
hello();
hello();
}
/* definicao da funcao hello() */
void hello(void)
{
printf("Hello Word! \n");
}
```

Funções com argumentos

Abaixo, mostramos um exemplo simples de um programa que consiste de duas funções: main() e hello().

```
#include <stdio.h>
/* declaracao (protótipo) da funcao alo() */
void hello(char nome[50]);
/* definicao da funcao main() */
main()
{
 char nome[50];
 scanf("%s",&nome);
 hello(nome);
}
/* definicao da funcao hello() */
void hello(char arg[50)
{
 printf("Hello %s", arg);
}
```

Funções com retorno

Abaixo, mostramos um exemplo simples de um programa que consiste de duas funções: main() e calculaidade().

```
#include <stdio.h>
/* declaracao (protótipo) da funcao */

int calculaidade(int nasc, int ano);
/* definicao da funcao main() */
main()
{
 int nasc, ano;
 scanf("%d",&nasc);
 scanf("%d",&ano);
 printf("Idade:%d", calculaidade(nasc,ano));
}
/* definicao da funcao calculaidade() */

int calculaidade(int nasc, int ano){
 int idade;
 idade = ano - nasc;
return idade;
}
```

Escopo de variáveis

Chamamos de escopo de variável ao conjunto de regras que determinam a utilização de uma variável em um programa. Podemos dividir as variáveis quanto ao escopo em três tipos: variáveis locais, parâmetros formais e variáveis globais.

Variáveis locais:

- São aquelas declaradas dentro do bloco de uma função.
- Não podem ser usadas ou modificadas por outras funções.
- Somente existem enquanto a função onde foi declarada estiver sendo executada.

Escopo de variáveis

Parâmetros formais:

- Os parâmetros formais de uma função também são variáveis locais da função.

Variáveis Globais:

- São declaradas fora de todos os blocos de funções.
- São acessíveis em qualquer parte do programa, ou seja, podem ser usadas e modificadas por todas as- outras funções.
- Existem durante toda a execução do programa.

Escopo de variáveis

```
#include<stdio.h>

float media, nota1, nota2;

void entrada(void);

main() {
char resposta;
do {
    entrada();
    media = (nota1 + nota2) / 2;

    printf("\nMedia do aluno: %.2f\n", media);

    printf("\nDeseja calcular outra media?
(s/n)");
    fflush(stdin);
    scanf("%c",&resposta);
  }
  while(resposta == 's');
  return(0);
}
```

```
void entrada(void)
{
  printf("\nDigite a primeira nota: ");
  scanf("%f", &nota1);

  printf("Digite a segunda nota: ");
  scanf("%f", &nota2);

  return;
}
```

Classes de armazenamento

As mais importantes classes de armazenamento em C são representadas pelas palavras reservadas ***auto, static, extern e register***.

Uma variável auto é local para a função onde está definida. Como esta é a classe de armazenamento normal para uma variável definida dentro de uma função, a palavra reservada auto é sempre opcional, e raramente utilizada na prática.

Classes de armazenamento

Uma variável ***static*** definida dentro de uma função é também local em utilização, mas retém seu valor de uma chamada de função para a próxima.

Utilização

– Quando for necessário recuperar o valor de uma variável na execução passada da função

Classes de armazenamento

Uma variável ***extern*** está definida fora de qualquer função e está disponível globalmente - para todas as funções localizadas abaixo da declaração da variável. Geralmente se omite a declaração de classe externa para as variáveis globais, por considerá-la redundante.

Classes de armazenamento

O computador tem a memória principal e os registradores da CPU. As variáveis são espaços reservados na memória. O modificador ***register*** diz ao compilador que a variável em questão deve ser, se possível, usada em um registrador da CPU.

Ponteiros

Permitem manipulação direta de endereços de memória.
Variáveis do tipo ponteiro
– Armazenam endereços de memória
– É possível definir um ponteiro para cada tipo do C que seja capaz de armazenar endereços de memória em que existem valores do tipo correspondente
– int a;
– int* p; // *p* armazena endereço de memória em que há valor inteiro
– p = &a; // armazena o endereço de memória da variável *a* no ponteiro *p*

Operadores de ponteiros

- Operador & (“endereço de”)

– Aplicado a variáveis, retorna o endereço da posição de
memória reservada para variável

- Operador * (“conteúdo de”)

– Aplicado a ponteiros, acessa o conteúdo de memória do
endereço armazenado pela variável ponteiro

Exemplos

```
int main (void)
{
int a;
int *p;
p = &a;
*p = 2;
printf (“ %d “, a);
return;
}
```

Imprime o valor 2

Passando ponteiros para função

- Ponteiros permitem modificar o valor das variáveis Indiretamente.
- Passagem por referência em C.

```
void somaprod (int a, int b, int *p, int *q)
{ *p = a + b;
*q = a * b;
}
int main (void)
{
int s, p;
somaprod (3, 5, &s, &p);
printf (“soma = %d produto =%d\n”, s, p);
return 0;
}
```