Rotina

Um importante recurso das linguagens de programação é a modularização, na qual um programa pode ser particionado em sub-rotinas específicas. A linguagem C/C++ possibilita a modularização por meio de funções.

Um programa em linguagem C/C++ tem, no mínimo, uma função principal – a função main() – por onde a execução começa. Em C/C++ não existe o conceito de procedimento. Os procedimentos em C/C++ são também funções, com a particularidade de não retornarem nada.

Definição

A sintaxe geral para a definição de uma função é:

```
tipo_de_retorno nome_da_função (parâmetros) {
    variáveis_locais
    instruções
}
```

Onde:

tipo_de_retorno: tipo da informação que a função vai retornar (default é int).

parâmetros (ou argumentos – args): lista de parâmetros com o formato:

tipo nome1, tipo nome2, ..., tipo nomeN

Os parênteses ao lado do nome da função são obrigatórios mesmo que a lista de parâmetros esteja vazia.

Por exemplo, uma função para calcular o valor médio de dois números, poderia ser definida da seguinte forma:

```
float media(float a, float b) {
    float m;
```

```
        m = (a + b) / 2;
        return  m;
}
```

Esta função poderia depois ser chamada na função main() como é mostrado no exemplo seguinte:

```
#include <iostream>
using namespace std;
float media(float a, float b);
main() {
        float a=5, b=15, resultado;
        resultado = media(a, b);
        cout << "Media = " << resultado << endl;
}
```

Neste exemplo, foi calculada a média entre dois valores do tipo real (a e b) que foram passados à função media como parâmetros. O resultado é armazenado na variável m e será devolvido ao ponto em que a função foi chamada na função principal main(), para ser utilizado conforme a necessidade do programa. Isso só é possível porque a função media está preparada para tratar esses valores, conforme especificado na primeira linha da função (cabeçalho).

Repare na instrução de return na função, que além de terminá-la também é responsável pela definição do valor de retorno da mesma. Pode haver mais de um comando return em uma função. Sua forma é:

```
return valor_retornado;
ou
return;
```

Exemplo)

```
#include <iostream>

using namespace std;

int quadrado(int x) {

        return (x*x);

}

main() {

        int num;

        cin >> num;

        cout << quadrado(num) << endl;

}
```

Protótipo de Função

Toda função deve ser declarada antes de ser utilizada. Na definição da função está implícita a sua declaração, mas a linguagem C permite que se declare uma função antes de defini-la.

Isto é feito através do protótipo da função, que nada mais é do que o trecho de código que especifica o nome e os parâmetros da função (cabeçalho seguido de ;).

A forma geral de uma definição de protótipos de função é:

```
tipo_retorno nome_da_função(tipo param1, tipo param2, ..., tipo paramN);
```

O uso dos nomes dos parâmetros é opcional, porém eles habilitam o compilador a identificar qualquer incompatibilidade de tipos por meio do nome quando ocorre um erro.

Os protótipos permitem que a linguagem C forneça uma verificação mais forte de tipos, pois quando são utilizados o C pode encontrar e apresentar quaisquer conversões de tipos ilegais entre o argumento usado para chamar uma função e a definição de seus parâmetros. A linguagem C também encontra diferença entre o número de argumentos usados para chamar a função e o número de parâmetros da função.

Em C os protótipos são opcionais, porém são obrigatórios em C++.

Exemplo:

```
#include <iostream>

using namespace std;

int quadrado(int x);

main() {

        int num;

        cin >> num;

        cout << quadrado(num) << endl;

}

int quadrado(int x) {

        return (x*x);

}
```

Funções do tipo void

Um dos usos de void (vazio) é declarar explicitamente funções que não devolvem valores. Isso evita seu uso em expressões e ajuda a afastar um mau uso acidental.

Antes de poder usar qualquer função void, você deve declarar seu protótipo. Se isso não for feito, C assumirá que ela devolve um inteiro e, quando o compilador encontrar de fato a função, ele declarará um erro de incompatibilidade.

Resumidamente temos:

Protótipo de uma função que não retorna nada:

void nome_função(parâmetros);

Nesse caso return não é necessário.

Uma função sem parâmetros:

```
tipo_retorno nome_função(void);
```

Uma função sem parâmetros e sem retorno:

```
void nome_função(void);
```

Exemplo:

```
#include <iostream>

using namespace std;

void mensagem(void);

main() {

    mensagem();

}

void mensagem(void) {

    cout << "Mostrando a mensagem!";

}
```

Lembre-se que na chamada de funções sem parâmetros (argumentos) é sempre obrigatório utilizar parênteses, sem nada dentro, como acima.

Escopo de Variáveis

Escopos são regras que determinam o uso e a validade de variáveis nas diversas partes do programa.

A declaração de variáveis pode ser feita em três lugares básicos: dentro de funções (variáveis locais), na definição dos parâmetros das funções (parâmetros formais) e fora de todas as funções (variáveis globais).

Em C, todas as funções estão no mesmo nível de escopo. Isto é, não é possível definir uma função internamente a uma função.

Variáveis Locais

São as variáveis declaradas dentro de uma função e só têm validade dentro do bloco no qual foi declarada.

As variáveis locais existem apenas enquanto o bloco de código em que foram declaradas está sendo executado. Ou seja, uma variável local é criada na entrada de seu bloco e destruída na saída.

Geralmente as variáveis usadas por uma função são declaradas imediatamente após o abre-chaves da função e antes de qualquer outro comando. Porém as variáveis locais podem ser declaradas dentro de qualquer bloco de código. O bloco definido por uma função é simplesmente um caso especial.

Parâmetros Formais

Se uma função usa argumentos, ela deve declarar variáveis que receberão os valores dos argumentos. Essas variáveis são denominadas parâmetros formais da função. Elas se comportam como qualquer outra variável local dentro da função e suas declarações ocorrem dentro dos parênteses.

Uma vez declarada as variáveis como parâmetros formais, elas podem ser usadas dentro da função como variáveis locais normais, e assim, elas também são destruídas na saída da função.

Os parâmetros formais devem ser declarados do mesmo tipo dos argumentos utilizados para chamar a função, caso contrário resultados inesperados podem ocorrer.

Embora a linguagem C forneça os protótipos de funções, que podem ser usados para ajudar a verificar se os argumentos usados para chamar a função são compatíveis com os parâmetros, ainda podem ocorrer problemas (os protótipos de função não eliminam inteiramente incongruências de tipo de parâmetros). Além disso, os protótipos de funções devem ser incluídos explicitamente no programa para receber esse benefício extra.

Variáveis Globais

Essas variáveis são reconhecidas pelo programa inteiro e podem ser usadas por qualquer pedaço de código. Além disso, elas guardam seus valores durante toda a execução do programa e são declaradas fora de qualquer função. Elas podem ser acessadas por qualquer expressão independentemente de qual bloco de código contém a expressão.

```
#include <iostream>
using namespace std;
int cont;
void func1(void);
void func2(void);
main() {
        cont = 100;
        func1();
}
void func1(void) {
        int x;
        x = cont;
        func2();
        cout << "func1 cont = " << x << endl;
}
void func2(void) {
        int cont;
        for (cont=0; cont<10; cont++) {
                cout << ".";
        }
        cout << endl;
}
```

Saída)

```
.....
func1 cont = 100
```

Observe que, apesar de nem main() e nem func1() terem declarado a variável cont, ambas podem usá-la. A func2(), porém, declarou uma variável local chamada cont. Quando func2() referencia cont, ela referencia apenas sua variável local, não a variável global.

Se uma variável global e uma variável local possuem o mesmo nome, todas as referências ao nome da variável dentro do bloco onde a variável local foi declarada dizem respeito à variável local e não têm efeito algum sobre a variável global.

As variáveis globais são úteis quando o mesmo dado é usado em muitas funções no programa. No entanto, deve-se evitar usar variáveis globais desnecessárias. Elas ocupam memória durante todo o tempo em que o programa está sendo executado, não apenas quando são necessárias. Além disso, as variáveis globais tornam o programa mais difícil de ser entendido e a função menos geral (a função depende de alguma coisa que deve ser definida fora dela).

Finalmente, usar um grande número de variáveis globais pode levar a erros no programa devido à mudança acidental do valor de uma variável (em consequência de ela ter sido usada em algum outro lugar do programa).

Passagem de Parâmetros por Valor

Numa passagem de parâmetros por valor serão geradas cópias dos valores de cada um dos parâmetros e a função atua sobre essa cópia sem poder modificar a variável original que foi passada.

```
#include <iostream>

using namespace std;

int somadobro(int x, int y);

main() {

        int x, y, r;

        cout << "\nDigite dois numeros:" << endl;

        cin >> x;

        cin >> y;
```

```
        r = somadobro(x, y);
        cout << "\nA soma do dobro dos numeros " << x << " e " << y << " = "
             << r << endl;
}
int somadobro(int x, int y) {
        int sd;
        x = 2 * x;
        y = 2 * y;
        sd = x + y;
        return sd;
}
```

Passagem de Parâmetros por Referência

Neste método os parâmetros passados para uma função correspondem a endereços de memória ocupados pelas variáveis utilizadas na chamada da função. Esta referência ao endereço de uma variável é feita declarando os parâmetros formais como ponteiros e, consequentemente as alterações feitas no parâmetro afetam a variável usada para chamar a função.

Observação:

Operador *: faz acesso ao conteúdo de uma área de memória indicada por um ponteiro.

Operador &: retorna o endereço de uma variável.

Exemplo 01)

```
#include <iostream>

using namespace std;

int somadobro(int *a, int *b);

main() {

        int x, y, r;

        cout << "\nDigite dois numeros:" << endl;

        cin >> x;

        cin >> y;

        r = somadobro(&x, &y);

        cout << "\nA soma do dobro dos numeros " << x << " e " << y << " = "

                << r << endl;

}

int somadobro(int *a, int *b) {

        int sd;

        *a = 2 * *a;

        *b = 2 * *b;

        sd = *a + *b;

        return sd;

}
```

Exemplo 02)

```
#include <iostream>

using namespace std;

void troca(int *a, int *b);
```

```
main() {
        int x, y;
        cout << "\nDigite dois numeros:" << endl;
        cin >> x;
        cin >> y;
        troca(&x, &y);
        cout << x << " " << y << endl;
}
void troca(int *a, int *b) {
        int t;
        t = *a;
        *a = *b;
        *b = t;
}
```

Exercícios:

01) Faça uma função que retorne 1 se o número digitado for positivo ou 0 se o número for negativo.

02) Faça uma função que receba dois números positivos por parâmetro e retorne a soma dos N números inteiros existentes entre eles.

03) Fazer um programa que possua uma rotina que leia uma entrada do usuário e faça a validação da entrada de dados S/s ou N/n. Só permita a saída da rotina se o usuário digitou S/s ou N/n. Faça um programa para testar sua rotina. A rotina deverá ter o cabeçalho a seguir.

```
void validaEntrada(char *ent);
```

04) Faça um programa que possua as seguintes rotinas:

- Uma rotina para calcular a subtração de dois números reais (X e Y). A rotina deve receber como parâmetros os valores de X e Y e retornar o resultado (X – Y).

- Uma rotina para calcular a multiplicação de dois números reais (X e Y). A rotina deve receber como parâmetros os valores de X e Y e retornar o resultado (X * Y).
- Uma rotina para calcular a divisão de dois números reais (X e Y). A rotina deve receber como parâmetros os valores de X e Y e retornar o resultado (X / Y).
- Uma rotina para calcular o fatorial de um número inteiro (X). A rotina deve receber como parâmetro o valor de X e retornar o resultado (X!).

O programa deve ler um par de números inteiros positivos (M e P), calcular (USANDO SOMENTE AS ROTINAS ACIMA) e escrever o número de arranjos (fazer uma rotina) e combinações (fazer uma rotina) de M elementos P a P. Se M < P, então os valores para A e C são iguais a zero.

$$A_M^P = \frac{M!}{(M-P)!} \qquad C_M^P = \frac{M!}{P!(M-P)!}$$

05) Fazer um programa que possua duas rotinas. Uma para aumentar de 1 um número inteiro dado e outra rotinha para decrementar de 1 um número inteiro dado. Faça um programa para testar suas rotinas. (Mudar os valores das variáveis da função main).

06) Faça uma rotina para calcular as raízes de uma equação do segundo grau. A rotina deve retornar o tipo de raiz (se existe raiz real ou não) e o valor das raízes (se existirem raízes reais).

07) Faça uma rotina que retorno o valor absoluto de um número. Faça um programa para testar a sua rotina.