CAPÍTULO 6

Sub-rotinas: Funções

CAPÍTULO 6

SUB-ROTINAS

Sub-rotinas são blocos de instruções que realizam tarefas específicas. O código de uma sub-rotina é agregado ao programa principal uma única vez e pode ser executado quantas vezes for necessário no decorrer do programa.

Como os problemas complicados exigem programas extensos, a solução é dividir os programas grandes em pequenos módulos, sub-rotinas, menores e de solução mais simples. Na literatura de programação isto se chama **modularização.**

Os programas (no C++, a função principal ***int*** *main*) são executados linearmente até o fim. Entretanto, quando são utilizadas sub-rotinas, é possível a realização de desvios na execução natural desses programas. Estes desvios são realizados no C++ quando uma função secundária é chamada pela função principal (***int*** *main*). E quando na função secundária é encontrado o comando *return*, a execução do *main* continua no ponto onde foi interrompido e continua sua execução normal.

Vantagens em se utilizar sub-rotinas:

- subdivisão de programas complexos com o objetivo de um melhor entendimento;
- estruturação de programas com o objetivo de detecção de erros e melhores documentação e manutenção;
- reutilização do código.

DEFINIÇÃO DE UMA SUB-ROTINA

Uma sub-rotina pode receber dados do programa que a chamou e ao terminar sua tarefa, ela pode fornecer ao programa principal os resultados obtidos nela.
Assim como as variáveis, as sub-rotinas também precisam ser declaradas antes de serem utilizadas.

CABEÇALHO

- **tipo** (identificar o tipo da sub-rotina: *int, float, char, bool, void* .....)
- **nome** (que é chamado pelo programa principal)
- **parâmetros** (canais por onde os dados são transferidos do programa chamador para a sub-rotina)

variáveis locais (definidas dentro da sub-rotina e que só podem ser utilizadas pela mesma)

CORPO — instruções que são executadas cada vez que a sub-rotina é chamada

Os componentes necessários para adicionar uma sub-rotina a um programa são: o seu protótipo (se a definição/declaração da mesma for após o programa chamador), a sua chamada e a sua definição (declaração).

SINTAXES DE SUB-ROTINA

As sub-rotinas definidas antes de serem chamadas não necessitam de protótipos.

1) Definição de sub-rotina antes do programa principal:

```
<definição da sub-rotina>

int main ()
{
        <declaração das variáveis locais ao main>

        <corpo do programa principal>
        <chamada à sub-rotina>
}
```

Se uma sub-rotina é chamada antes de ser definida/declarada, é necessário que ela seja prototipada.

2) Definição de sub-rotina depois do programa principal:

```
<definição do protótipo da sub-rotina>

int main ()
{
        <declaração das variáveis locais ao main>

        <corpo do programa principal>
        <chamada à sub-rotina>
}

<definição da sub-rotina>
```

O protótipo de uma sub-rotina tem a mesma forma da primeira linha da definição/declaração da sub-rotina, exceto por terminar com um ponto e vírgula (;).

Um importante recurso apresentado nas linguagens de programação é a modularização, na qual um programa pode ser particionado em sub-rotinas bastante específicas. Em geral, as linguagens possibilitam a modularização por meio de procedimentos e funções. A linguagem C/C++ possibilita esta modularização por meio de funções.

FUNÇÃO

A função tem o objetivo de desviar a execução do programa para realizar uma tarefa específica e quando ela retorna valor, ela retorna um e somente um valor ao programa chamador: que pode ser uma constante, uma varíavel ou um resultado de uma expressão. A função é chamada quando seu nome aparece no corpo do programa.

Pode-se escrever quantas funções quiser e qualquer função pode chamar uma outra, mas uma função não pode ser declarada/definida dentro de outra função. As funções são módulos independentes. Elas não podem estar aninhadas.

Pode-se utilizar funções sem ou com passagem de parâmetros. Quando for com passagem de parâmetros, a função é chamada e na sua chamada lhe é atribuída alguns valores.

Pode-se utilizar funções com ou sem retorno. Quando for com retorno, a função quando chamada e executada, produz um resultado que é retornado ao programa ou função principal (*main*).

SINTAXE DA FUNÇÃO

```
<tipo_da_função> <nome> ( <tipo_de_dado_dos_parametros> <lista_de_parâmetros> )
{
        <declaração das variáveis locais>

        <bloco de comandos da função>
        return .....;
}
```

onde:

<tipo_da_função> é o tipo de dado retornado pela função ao programa ou função principal, que pode ser numérico, literal ou lógico (*int, float, double, char, bool*...). Precisa-se definir um determinado tipo para a função que está sendo declarada. Quando se declara uma função, na verdade, está se declarando uma "variável". O tipo da função é determinado pelo valor que ela retorna via comando **return**, e não pelo tipo de parâmetros que ela recebe. Uma função pode não ter retorno, seu tipo é *void.*

<nome> para se nomear uma função deve-se seguir as mesmas regras de nomes para variáveis, ou seja, o nome não pode conter caracteres especiais diferentes de _ (*underline*) e não pode começar com números.

<tipo_de_dado_dos_parametros> é o tipo de dado de cada parâmetro a ser passado pelo programa ou função principal (main)

<lista_de_parâmetros> as variáveis que passarão informações à função são chamadas de parâmetros e a função deve declarar esses parâmetros entre parênteses, no cabeçalho da definição da função. Esses parâmetros podem ser utilizados livremente no corpo da função.

return termina a execução da função e retorna o controle para a instrução seguinte no programa ou função principal (main), após a chamada da função. O comando **return** pode retornar somente um único valor para o programa chamador. A sintaxe do return tem 4 formas:

```
return constante;       // retorne o valor -1, por exemplo;

return variável;        // retorne o conteúdo da variável soma;

return expressão;       // retorne o resultado da expressão aritmética (F-32)*5/9; ou

return;                 // apenas termina a execução da função. Função que não
                        // retorna nada, e, portanto, seu tipo também não é definido.
                        // Exemplo: função void
```

DIFERENTES TIPOS DE FUNÇÕES

Pode-se dividir as funções em diferentes categorias com base no retorno e nos parâmetros:

Com base no retorno:
- com retorno
- sem retorno

Com base nos parâmetros:
- com passagem de parâmetros
  - por cópia (ou por valor)
  - por referência
- sem passagem de parâmetros

VARIÁVEIS GLOBAIS E LOCAIS

Ao se declarar uma variável, dependendo do local onde a mesma foi declarada, ela pode ser uma variável global ou uma variável local. Ao se declarar uma variável dentro de qualquer função (inclusive do *main*), ela assume o papel de variável local, e, portanto, só pode ser acessada na própria função onde foi declarada. Mas se a declaração for fora de qualquer função, tem-se uma variável global, que pode ser acessada em qualquer função do código. Lembre-se, toda variável (global ou local) precisa ser declarada antes de qualquer linha de código que a utiliza.

Exemplo:

```
int a, b;          // variáveis globais

int funcao1 ()
{
        int x, y;          // variáveis locais
        return 1;
}

int main ()
{
        int c, d;          // variáveis locais
        return 0;
}
```

FUNÇÃO COM RETORNO E COM PASSAGEM DE PARÂMETRO POR CÓPIA

Neste tipo de função os parâmetros recebem uma cópia dos valores passados a ela. Qualquer alteração feita nos parâmetros dentro da função, isto é feito apenas localmente, portanto esta alteração não se aplica às demais funções.

Se vários parâmetros são passados à função, eles são passados entre parênteses e separados por vírgula (não com ponto e vírgula). E pode-se passar quantos parâmetros e de tipos distintos forem necessários para a função.

A função é com retorno, portanto ao ser chamada, fornece um valor ao código, tal valor pode corresponder a qualquer tipo de dado (**int, float, double, char, bool**....).

Como ela tem retorno, é importante verificar como este retorno está sendo recebido no código que a chamou: atribuição a uma variável, dentro de uma expressão ou condição, dentro de um comando de saída, etc. O valor retornado não pode fica "solto".

Exemplo 1: Escreva um programa que leia um número e retorne 1 se o número digitado for positivo ou 0 se o número for negativo, utilizando uma função com retorno e com parâmetro para efetuar esta consulta.

SINTAXE 1 (declaração da função antes da *main*; não é necessário definição de protótipo):

```
int verifica (float posneg)
        /* posneg é o parâmetro e a função retorna um valor numérico inteiro */
{
        if (posneg >= 0)
                return 1;
        else return 0;
}
```

```
int main ()             // programa positivo/negativo
{
        float num;      // num é o número lido
        int x;          // x recebe o retorno da função verifica

        cout << "Entre com o número a ser consultado ";
        cin >> num;
        x = verifica(num);      // chamada à função
        if  (x == 0)
                cout <<  num << " é um número negativo" << endl;
        else cout << num << " é um número positivo" << endl;

        return 0;
}
```

SINTAXE 2 (declaração da função depois da *main*; necessidade de definição de protótipo):

```
int verifica (float posneg);            // definição do protótipo da função
```

```
int main ()             // programa positivo/negativo
{
        float num;      // num é o número lido
        int x;          // x recebe o retorno da função verifica

        cout << "Entre com o número a ser consultado ";
        cin >> num;
        x = verifica(num);      // chamada à função
        if  (x == 0)
                cout <<  num << " é um número negativo" << endl;
        else cout << num << " é um número positivo" << endl;

        return 0;
}
```

```
int verifica (float posneg)
        /* posneg é o parâmetro e a função retorna um valor numérico */
{
        if (posneg >= 0)
                return 1;
        else return 0;
}
```

Exercício 6.1) Faça um programa que leia um número (N) natural e escreva a soma dos N números naturais existentes do número 1 até esse número lido, utilizando uma função com parâmetro e com retorno para efetuar o cálculo. Fazer a crítica de N natural e maior que 0.

FUNÇÃO SEM RETORNO E COM PASSAGEM DE PARÂMETRO POR CÓPIA

Neste tipo de função os parâmetros recebem uma cópia dos valores passados a ela. Qualquer alteração feita nos parâmetros dentro da função, isto é feito apenas localmente, portanto esta alteração não se aplica às demais funções.

Como a função é sem retorno, a saída do resultado é mostrada dentro dela (com o **cout**).

Exemplo 2: Faça um programa que leia um valor numérico positivo e escreva a raiz quadrada deste número através de uma função sem retorno e com parâmetro.

Usando a Sintaxe 1 (declaração da função antes da *main*):

```
void raiz (float n)          // função void sem retorno; n é o parâmetro
{
        cout << "Raiz quadrada de " << n << " é << sqrt(n);
        // o resultado foi mostrado dentro da própria função

        return;      // sem retorno, nenhum valor foi devolvido ao programa ou função
                     //  principal (main)
}
```

```
int main ()     // programa raiz quadrada
{
        float num;      // num é lido

        do
        {
            cout << "Entre com o número a ser processado ";
            cin >> num;
        } while (! (num >=0));    // crítica de número positivo

        raiz (num);                  // chamada à função sem retorno

        return 0;
}
```

Exercício 6.2) Faça um programa que leia um valor numérico inteiro e verifique se o mesmo é divisível por 2 e por 3. A verificação deve ser feita por uma função sem retorno e com parâmetro.

Função secundária:

Função principal:

FUNÇÃO SEM RETORNO E SEM PARÂMETRO

Neste tipo de função não é passado a ela nenhum parâmetro para o seu funcionamento. É comum que este tipo de função utilize variáveis globais em sua definição/desenvolvimento.

Como a função é sem retorno, a saída deve estar nela. Mas pode também não estar, pois se usa variável global, a saída pode estar em outra função.

Uma função que não retorna nenhum valor ao código é por definição uma função do tipo **void** (vazio). Neste tipo de função o uso do **return** é opcional.

Exemplo 3: Faça um programa que leia um valor numérico e apresente o seu quadrado, utilizando uma função sem retorno e sem parâmetro para efetuar esse cálculo.

Usando a Sintaxe 2 (declaração da função depois da *main* e prototipada antes)

```
float x, y;       // x e y são variáveis globais declaradas fora das funções
```

```
void quad();   // definição do protótipo da função sem retorno e sem parâmetro
```

```
int main ()     // programa quadrado
{
        cout << "Digite um número ";
        cin >> x;                  // usando a variável global
        quad();          // chamada da função
        return 0;
}
```

```
void quad()    // função sem retorno e sem parâmetro
{
        y = x * x;                 // usando as variáveis globais
        cout << "O quadrado de " << x << " é "<< y;

        return;          // sem retorno
}
```

Pode-se utilizar também a chamada a uma função como parâmetro para outra função.

RESUMO DAS FUNÇÕES:

CATEGORIA	**OBJETIVO**	**CHAMADA no *main***
com parâmetro e com retorno	processamento	NOMEFUNCAO(PARAMETRO) a) atribuída a uma variável b) dentro de uma expressão c) em uma condição d) dentro de um escreva e) como parâmetro de outra função
com parâmetro e sem retorno	processamento com saída	NOMEFUNCAO(PARAMETRO);
sem parâmetro e sem retorno	processamento com saída (ou não)	NOMEFUNCAO();

MECANISMO DE PASSAGEM DE PARÂMETROS

Para invocar uma sub-rotina (função), às vezes se utiliza parâmetros. Esta invocação pode se dar segundo 2 mecanismos: passagem por cópia (ou valor) e passagem por referência.

PASSAGEM POR CÓPIA (ou por VALOR)

O parâmetro é fornecido à sub-rotina no ato da invocação e as modificações feitas na sub-rotina com esse parâmetro não afetam o conteúdo da variável que foi passada, pois trabalha-se com a sua cópia, ou seja, passagem de parâmetros por cópia significa que, para a execução da função, será gerada uma cópia do valor de cada uma das variáveis que foram passadas como parâmetros.

SINTAXE:

```
void funcao(int p);

float função(int p);
```

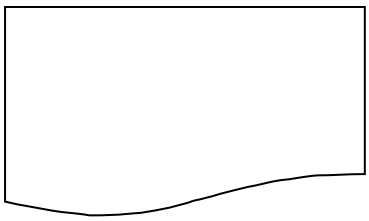
Exercício 6.3) Qual será o resultado da execução deste programa?

```
void proc (int y);       // protótipo da função

int main ()     //  programa passa valor
{
        int x;

        x = 1;
        cout << "Antes x = " << x << endl;
        proc(x);
        cout << "Depois x = " << x << endl;
        return 0;
}

void proc (int y)
{
        y = y + 1;
        cout << "Durante y = " << y << endl;
        return;
}
```



A função **proc** não alterou o valor da variável **x** passada como parâmetro durante sua execução. Foi feita uma reserva de espaço na memória para armazenar uma cópia dessa variável **x**. No momento em que a função **proc** chega ao fim, o parâmetro **y** é destruído e, portanto, a alteração realizada pelo incremento unitário é perdida.

PASSAGEM POR REFERÊNCIA

A variável sofre eventuais modificações quando é executada a sub-rotina. Passagem de parâmetros por referência significa que os parâmetros passados para uma função correspondem a endereços de memória ocupados pelas variáveis passadas. Dessa maneira, toda vez que for necessário acessar um determinado parâmetro na função, isso será feito por meio de referência ao endereço da variável passada como parâmetro.

Funções que recebem parâmetros por referência utilizam o operador **&** somente na definição/declaração do tipo do parâmetro.

SINTAXE:

Na declaração:

```
void nomefuncao(int &p);
```

Na chamada:

```
nomefuncao(var);
```

```
void proc (int &y);              // protótipo da função

int main ()     //  programa passa valor
{
        int x;
        x = 1;
        cout << "Antes x = " << x << endl;
        proc(x);
        cout << "Depois x = " << x << endl;
        return 0;
}
void proc (int &y)
{
        y = y + 1;
        cout << "Durante y = " << y << endl;
        return;
}
```

Um ponteiro é uma variável que aponta para o endereço de memória de outra variável. Para criar um ponteiro e utilizá-lo, é necessário acrescentar um ***** antes do nome da variável, não apenas na sua declaração, mas também quando for utilizar o valor para o qual ele aponta. Para acessar o endereço de memória de uma variável basta adicionar o símbolo **&** antes da variável a qual se deseja obter o endereço.

Permite-se utilizar ponteiros para implementar parâmetros por referência. No exemplo abaixo a função proc (**int** *y) utiliza *y para acessar ao inteiro referido pelo endereço **int** da variável x na sua chamada.

SINTAXE:

```
Na declaração:
        void nomefuncao(int *p);

Na chamada:
        nomefuncao(&var);
```

Exercício 6.4) Qual será o resultado da execução deste programa?

```
void proc (int *y);              // protótipo da função

int main ()     //  programa passa valor
{
        int x;
        x = 1;
        cout << "Antes x = " << x << endl;
        proc(&x);
        cout << "Depois x = " << x << endl;
        return 0;
}
void proc (int *y)
{
        *y = *y + 1;
        cout << "Durante y = " << *y << endl;
        return;
}
```

Quando a função **proc** chega ao fim, o parâmetro **y** é destruído. Entretanto, a alteração decorrente do incremento unitário feito, é mantida, pois não foi gerada duplicata de valor, mas sim, referência ao endereço de memória da variável **x** que foi passada como parâmetro.

Utiliza-se este mecanismo de passagem por referência quando se deseja que uma função modifique o valor do parâmetro passado e devolva esse valor modificado para o programa ou função principal.

A principal vantagem de passagem por referência é a de que a função pode acessar as variáveis do programa chamador (*int main*, por exemplo). Este mecanismo além de economia de espaço de memória, possibilita que uma função retorne mais de um valor para o programa chamador. Ou se a função é sem retorno, o retorno pode-se dar através deste parâmetro passado por referência. Os valores (resultados da função) a serem retornados são colocados em referências de variáveis do programa chamador.

RECURSIVIDADE

Uma função é dita recursiva quando existe dentro dela uma chamada para ela mesma por diversas vezes, ou seja, ela chama a si própria.

O código gerado por uma função recursiva exige a utilização de mais memória, o que torna a sua execução mais lenta.

Uma função recursiva deve ter uma parte BASE e uma parte RECURSIVA. A parte BASE é a parte que pára a recursão, ou seja, nesta parte básica deve ter uma condição de término da recursividade. A parte RECURSIVA é onde a função chama ela própria.

A função recursiva utiliza uma estrutura de dados chamada PILHA (abordagem *LIFO*). Várias chamadas da função estarão ativas ao mesmo tempo. Enquanto a última chamada não terminar, a penúltima não termina e assim por diante. Isso faz as variáveis de cada chamada serem todas mantidas na memória, o que requer mais memória.

A recursividade produz repetição sem usar as estruturas repetitivas conhecidas (*while, do-while e for*).

Exemplo 4:

$$n! = \begin{cases} n \times (n-1) \times (n-2) \times \ldots\ldots \times 1 & \text{para } n \geq 1 \\ 1 & \text{para } n = 0 \end{cases}$$

$$n! = \begin{cases} n \times (n-1)! & \text{para } n \geq 1 \\ 1 & \text{para } n = 0 \end{cases}$$

Função não recursiva para cálculo do fatorial:

```
int fat (int n)
{
        int i, f;          //  i é variável de controle e f é o fatorial
        f = 1;
        for (i = n ;  i  >= 2 ; i = i–  1)
                f = f * i;

        return f;
}
```

Função recursiva para cálculo do fatorial:

```
int fat (int n)
{
        int f;
        if (n == 0)                  } PARTE BASE
                f = 1;               }

        else f = n*fat(n-1);         } PARTE RECURSIVA
        return f;
}
```

Exemplo 5: Dados dois números inteiros, calcule o Máximo Divisor Comum entre eles através de uma função recursiva.

```
int mdc (int p1, int p2)
{
        int m;
        if (p2 == 0)
                m = p1;              } PARTE BASE

        else m = mdc(p2, p1%p2);     } PARTE RECURSIVA
        return m;
}
```

Note que nos exemplos apresentados de funções recursivas, um processo é repetido várias vezes, mesmo sem estar escrito fisicamente uma certa estrutura de repetição.

EXERCÍCIOS PROPOSTOS DO CAPÍTULO 6

P6.1) Faça um programa que leia dois números naturais, diferentes de zero e diferentes entre si; e escreva a soma dos números naturais existentes entre eles (exclusive os extremos). Utilize uma função (sem parâmetro e sem retorno) para efetuar o cálculo.

Se A < B, a soma vai de A até B (desconsidere os extremos)
Se A > B, a soma vai de B até A (desconsidere os extremos)

P6.2) Uma região possui 45.500 habitantes. De cada habitante foram coletados os seguintes dados: SEXO (M ou F) e IDADE (>0). Faça um programa para ler e criticar estes dados e determinar, por meio de uma função, o número de habitantes que são do sexo masculino e maiores ou iguais a 18 anos. Mostre o percentual deste número no programa.

P6.3) Faça um programa que leia três notas e uma letra para cada aluno de uma turma de 100 alunos. Faça uma função que calcule a média (ponderada) das notas do aluno com pesos 5, 3 e 2, se a letra digitada for A. Caso contrário, calcule a média aritmética das notas. A média calculada deve ser devolvida ao programa para ser mostrada. Consista a entrada das notas ($0 \leq$ notas ≤ 100).

P6.4) Faça um programa que leia um número inteiro positivo e verifique se o mesmo é primo. A verificação deve ser feita através de uma função (sem retorno e sem parâmetro).

P6.5) Faça um programa que leia 4 números e apresente os valores máximo, mínimo e a média dos valores, através de uma função. Uma função para cada valor.

P6.6) Faça um programa que leia um número inteiro positivo e calcule o seu fatorial. O cálculo do fatorial deve ser feito por uma função com parâmetro e com retorno.

P6.7) Faça um programa que leia as coordenadas (X_1,Y_1) e (X_2,Y_2) de dois pontos localizados no plano e calcule a distância euclidiana entre os dois. O cálculo da distância deve ser feito através de uma função com retorno e com parâmetros.

$$D^2 = (X_1 - X_2)^2 + (Y_1 - Y_2)^2$$

P6.8) Escreva um programa que leia um valor numérico (inteiro e diferente de zero) e escreva:

a) se ele é par ou ímpar (função com parâmetro e com retorno);
b) seu valor absoluto (função com parâmetro e sem retorno); não utilize a função abs(), crie sua própria função que faça/escreva o valor |em módulo| do parâmetro.
c) sua tabuada (função sem parâmetro e sem retorno).

Para cada um destes itens, escreva uma função.

P6.9) Faça um programa que leia 3 números reais (>0) e verifique se os mesmos podem ser os lados de um triângulo. Se forem, calcular o seu perímetro. Crie uma função para verificar se os números podem ser os lados de um triângulo e outra função para o cálculo do perímetro. Ambas as funções com parâmetros e com retorno.

P6.10) Escreva um programa para gerar e escrever o valor do cosseno de um ângulo (X) fornecido em radianos pelo usuário, empregando os primeiros 15 termos da série:

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!} + ...$$

O cálculo do fatorial do denominador deve ser feito por uma função com retorno.

P6.11) As funçoes hiberbólicas seno, cosseno e tangente são definidas como:

$$\text{senh}(x) = \frac{e^x - e^{-x}}{2} \qquad \cosh(x) = \frac{e^x + e^{-x}}{2} \qquad \tanh(x) = \frac{\text{senh}(x)}{\cosh(x)}$$

Construa:

a) duas funções com retorno que calcule, uma o seno e a outra o cosseno, tendo x como parâmetro de entrada, passado por cópia/valor;
b) uma função sem retorno que calcule o valor da tangente, tendo como parâmetros, passados por cópia/valor, os valores calculados do seno e cosseno e o valor da tangente (parâmetro de saída), passado por referência;
c) um programa que lê o valor de x e apresente os valores calculados do seno, cosseno e tangente.

P6.12) Faça um programa para ler um vetor de N números naturais, sendo 0<N≤100, construir e imprimir um novo vetor de mesmo tamanho, sendo que cada posição desse novo vetor contém a informação ('S' ou 'N') se o elemento correspondente à mesma posição do vetor lido é primo (S=Sim) ou não (N=Não).

A verificação, se o número é primo ou não, deve ser feita por uma função com retorno e com parâmetros

P6.13) Execute o programa a seguir e indique os valores que serão impressos para as variáveis **VOLT** e **AMPERE** ao final da execução do programa e justifique estes valores.

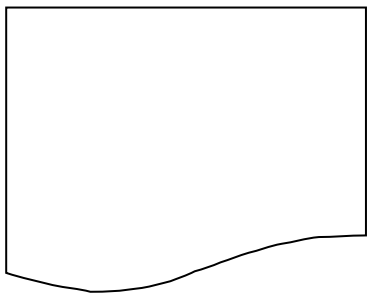
```
void circuito (int henry, int *farad);

int main ()       // programa elétrico
{
        int volt, ampere;

        volt = 8;
        ampere = 2;
        circuito (volt, &ampere);  // chamada à função

        cout << "Volt = " << volt << " - Ampère = " << ampere << endl;
        return 0;
}

void circuito (int henry, int *farad)
{
        bool maxwell;
        henry =  3;
        *farad = 7;
        maxwell = false;
        do
        {
                henry = henry – 1;
                *farad = *farad - henry;
                if (henry == 0)
                        maxwell = true;
        } while (!maxwell);
        return;
}
```



P6.14) Execute o programa a seguir e indique os valores que serão impressos para as variáveis VT e AP ao final da execução do programa e justifique estes valores.

```
void tempo (int descd, int *ascd)
{
        bool existe;
        descd = 12;
        *ascd = 15;
        existe = false;
        while (!(existe))
        {
                descd = descd – 5;
                if (descd < 0)
                    existe = true;
                *ascd = *ascd – descd;
        }
        return;
} // fim função
int main ()       // programa contagem
{
        int vt, ap;

        vt = 15;
        ap = 12;
        tempo (vt, &ap);  // chamada à função

        cout << "VT: " << vt << " - AP: " << ap << endl;
        return 0;
}
```

VT: **AP:**

Justificativa para os valores impressos de VT e AP:

P6.15) Escreva uma função recursiva que calcule a potência N (inteira) de X.

P6.16) A soma S dos N primeiros números naturais (N>0) pode ser escrita da seguinte forma:

$$S = 1 + 2 + 3 + ... + (N-2) + (N-1) + N$$

Como o valor de S depende do valor de N, tem-se:

$$S(N) = 1 + 2 + 3 + ... + (N-2) + (N-1) + N$$

ou seja,

$$S(N) = \begin{cases} 1 \ \ se \ N = 1 \\ N + S(N-1) \ \ se \ \ N > 1 \end{cases}$$

Faça uma função recursiva que calcule o valor de S.