LINGUAGEM C

1 - ESTRUTURA DE UM PROGRAMA EM C

Um programa em linguagem C é composto dos seguintes elementos:

- Diretivas de compilação
- Definições de tipos
- Protótipos de funções
- Funções
- Comentários

Eis um exemplo de um programa bem simples em linguagem C:

```
#include<stdio.h>

int a;

main()
{
a=10;
printf("O valor de a é: %d", a );
}
```

onde: `#include<stdio.h>` é uma diretiva de compilação

`int a` indica que o usuário criou uma variável numérica inteira (int) de nome **a**

`main()` é o corpo principal (ou função principal) do programa. Quando um programa inicia, o compilador reconhece as variáveis criadas e parte imediatamente para a execução do programa principal, que por sua vez, pode chamar outras funções. Uma correlação pode ser feita com os programas em Assembly ou basic, onde existe um corpo do programa principal, e de repente, há a chamada de uma subrotina que desvia o programa para outro lugar momentaneamente a fim de executar alguma função específica.

1.1 - DIRETIVAS DE COMPILAÇÃO

São os comandos que instruem o compilador a realizar determinadas tarefas antes de iniciar a compilação de todos ou parte do programa. Também são chamadas de diretivas de pré-processamento.

Todas as diretivas iniciam com `#include` , e os arquivos tem uma extensão `.h`

1.2 - SINTAXE

A sintaxe de um programa em linguagem C é a maneira de escrever o programa de forma que o compilador entenda.

Esse conjunto de regras é específico de cada linguagem. Quando há a necessidade de utilizar uma outra linguagem, como: Pascal, Basic, Cobol ou Assembly por exemplo, a sintaxe dos comandos é modificada. Por exemplo:

Em C existe uma necessidade de um programa principal `main()`, e cada comando é finalizado com um ponto e vírgula (;). Existem também comentários, que são dados, palavras ou frases que auxiliam o programador enquanto está escrevendo o software. Os comentários são escritos da seguinte maneira:

```
/* comentário de um programa em C para um compilador C Ansi */

// comentário de um programa em C para o compilador C++
```

Tudo que existir entre o intervalo /* .... */ não será interpretado pelo compilador C, então pode ser escrito qualquer caractere ou frase a fim de auxiliar o programador ou quem está lendo o programa fonte. A diferença entre os 2 tipos de comentário descritos acima, é que as duas barras (//) definem como comentário o que estiver à direita das barras, não necessitando um fechamento (como o */), porém só é reconhecido com sintaxe de comentário em compiladores C++.

2 - VARIÁVEIS E CONSTANTES SIMPLES

Para poder fazer uso das variáveis e constantes dentro de um programa, devemos utilizar identificadores. Existem algumas regras para a utilização de identificadores

- Todos os identificadores devem começar por a ... z ou A ... Z ou sublinhado (_). Por exemplo :

```
int a;          int pedra;
int A;          int _pedra;
```

Obs: a linguagem C é caractere sensível e interpreta letras minúsculas diferente de letras maiúsculas. Todo cuidado deve ser tomado quando definir uma variável deve ser utilizada de forma idêntica no programa.

- O resto do identificador pode ser composto de letras, números ou o próprio identificador sublinhado. Nenhum outro caractere é permitido.
  Exemplo: `int calcula_numero_primo;`

- Os primeiros 32 caracteres de um identificador são significativos

2.1 - DECLARAÇÃO DE VARIÁVEIS

Todas as variáveis devem ser declaradas antes de poderem ser utilizadas. A forma de declarar é:

Tipo nome_variavel = valor;

Exemplo:

```
int a;     // cria uma variável de nome a com inteira
int dolar=1000; // cria uma variável de nome dolar e inicializa em 1000
unsigned inicio=0,fim=83;
char opcao_menu;
```

2.2 - TIPOS DE VARIÁVEIS E CONSTANTES

2.2.1 - INTEIROS

- int

Representa um número inteiro (negativo ou positivo). O tamanho máximo varia dependendo do compilador C utilizado.

Por exemplo, nos compiladores para DOS (BorlandC^{++} 3.1 , Turbo C 2.0) são compiladores para máquinas de 16 bits, então seu tamanho máximo do número inteiro será 2^{16} ou por utilização do sinal negativo $\pm 2^{15}$ (-32768 à 32767).
Já o BorlandC 5.0 é um compilador construído para máquinas de 32 bits, então o valor máximo da variável inteira será de $\pm 2^{31}$ (-2147483648 à 2147483647).

- unsigned int

Define uma variável inteira (int) com um valor sem o sinal negativo. Essa conversão aumenta o range de aplicação em 2^{16} (0 à 65535).

- void

É um dado que não retorna valor algum à variável ou função. Será visto mais adiante quando for apresentada a idéia de funções.

- char

É uma variável ou constante do tipo caractere (o compilador utiliza a tabela ASCII). Também pode ser tratado com int porque a tabela ASCII retorna um valor numérico inteiro de 8 bits quando pressiona-se uma tecla. Seu valor vai de -128 à +128 ($\pm 2^7$). Se a função for definida como unsigned char fica seu valor fica 2^8 (0 à 255). A definição de um char para o compilador C deve ser colocada entre apóstrofes (‘A’).

Exemplos:

```
Char codigo,inicial; //cria 2 var.de nome codigo e outra de nome inicial
Unsigned char tipo,tab;    //idem ao anterior

/* em seguida atribui-se assim */

codigo=65;  // codigo ASCII de A
inicial='A'; // idem ao anterior
tab='\t';   // caractere tab
tipo='\x41'; //caractere 41 em hexadecimal = 65 em decimal = caractere A
tipo='\o101';  // caractere 101 em octal = 65 em decimal = caractere A
tipo=inicial+1;  // resulta no caractere B
tipo='A'+1;  // idem ao anterior
```

2.2.2 – PONTO FLUTUANTE

- float

É um dado em ponto flutuante (número fracionário). São os números reais da teoria de conjuntos na matemática. Quando o número tiver uma magnitude considerável, representa-se pela notação científica (3.45 e^{-10}).

Uma variável definida como float ocupa 4 bytes (32 bits)o que significa ter um range de 3,4 10^{-38} `a 3,4 10^{38}. A precisão de uma variável ou constante definida como float é o número de casas decimais que podem ser armazenadas como dígitos válidos. Para o compilador BorlandC 3.1 (16 bits) há pelo menos 6 dígitos significativos de precisão. Exemplos de variável float:

```
float a=-1.3546e-12;
float PI=3.1415;
```

- double

É também um dado em ponto flutuante, mas com precisão dupla. Possui 8 bytes (64 bits), valores entre 1.7 10^{-308} à 1.7 10^{308} com precisão de, pelo menos, 10 dígitos. Exemplo:

```
double x = -253.778978e274  //número com 9 dígitos significativos
```

Existe ainda o long double que tem sua precisão aumentada em relação ao double. A tabela abaixo vale para os tipos e varáveis em ponto flutuante para compiladores em 32 bits:

TIPO	BYTES	VARIAÇÃO	DÍGITOS SIGNIF.
float	4	3.4 10^{-38} à 3.4 10^{38}	7
double	8	1.7 10^{-308} à 1.7 10^{308}	15
long double	10	3.4 10^{-4932} à 3.4 10^{4932}	18

Programa exemplo

Esse programa permite a observação do tamanho máximo que o compilador aloca para cada tipo de dado

```
#include<stdio.h>

main()
{
char c;
unsigned char uc;      //definição das variáveis
int i;
unsigned int ui;
float f;
double d;

printf("Char %d" , sizeof(c));
printf("Unsigned Char %d" , sizeof(uc));
printf("Int %d" , sizeof(i));
printf("Unsigned Int %d" , sizeof(ui));
printf("Float %d" , sizeof(f));
printf("Double %d" , sizeof(d));
}
```

3 - OPERADORES

3.1 - ATRIBUIÇÃO

Quando quiser atribuir um valor numérico ou um caractere à uma variável, utiliza-se o sinal de igual (=).

```
int a;   // cria-se a variável de nome a
a=0;    // atribui o valor zero à variável a
```

Essa sintaxe pode ser resumida em:

```
int a=0;
```

obs: a idéia de atribuir um valor inicial à variável, ou seja , inicializar a variável com um valor pré-determinado é imprescindível em alguns casos, porque no momento que se cria uma variável, o compilador atribui à ela um valor aleatório qualquer (contido na memória do computador).

3.2 - ARITMÉTICOS

São idênticos à maioria dos operadores comumente utilizados em outras linguagens.

```
+ - * /
```

Há sempre algum operador um pouco diferente, por exemplo:

% (indica o resto de uma divisão inteira) exemplo: 4 % 2 é zero.

As operações podem ter seus operadores de forma combinada:

```
x=x+1;    é equivalente a escrever    x+=1;
x=x–5;      ⇒  x-=5;
x=x*(y+1); ⇒  x*=y+1;
```

Exemplo:

```
// programa para mostrar o resto de uma divisão entre 2 numeros

#include<stdio.h>
#include<conio.h>  // diretiva de compilação consoleIO.h inserida

      main()
     {
     int x,y;

     x=10;
     y=3;
     clrscr();  // limpa a tela do DOS
     printf("%d\n" ,x/y); /* quociente da divisão de x por y */
     printf("%d\n" ,x%y); /* mostra o resto da divisão de x/y */
     }
```

3.3 - OPERADORES RELACIONAIS

Traz a idéia de comparação entre valores. São eles:

>	maior que
> =	maior ou igual
<	menor
< =	menor ou igual
= =	igual (note a diferença entre a atribuição (=)
! =	diferente

O resultado obtido na utilização desses operadores é dado em verdadeiro (true) ou falso (false). Verdadeiro é qualquer valor diferente de zero. O resultado falso é o próprio zero. Por exemplo:

```
# include<stdio.h>
# include<conio.h>

     main()
     {
```

```
int i,j;
clrscr();
printf(“Digite dois números:”);
scanf(“%d%d”, &i , &j );
printf(“%d == % d  é  %d\n” , i , j , i == j);
printf(“%d != % d  é  %d\n” , i , j , i != j);
}
```

3.4 - INCREMENTO E DECREMENTO

O incremento de valores nas variáveis pode ser feito assim:

```
int x;
x=10;
x=x+1;  //incrementa em 1 o valor anterior de x
```

porém há uma maneira diferente de incrementar as variáveis

++ x ⇒ incrementa x antes de utilizar o seu valor
x ++ ⇒ incrementa x depois de ter utilizado seu valor

Exemplo:

```
#include<stdio.h>
#include<conio.h>

main()
{
int x=10;

clrscr( );
printf(“x= %d\n” , x++);
printf(“x= %d\n” , x);
printf(“x= %d\n”, ++x);
printf(“x= %d\n” , x);
}
```

3.5 - OPERADORES LÓGICOS

São operadores a nível de bit e permitem a operação de baixo nível alterando individualmente o valor dos mesmos.

& and
| or
^ xor
~ not
<< shift left
>> shift right

Exemplo:

a = 11010101 = 0xD5 = 213
b = 01001011 = 0x4B = 75

a & b = 01000001 = 0x41 = 65
a | b = 11011111 = 0xDF = 223
~a = 00101010 = 0x2A = 42
a << 1 = 10101010 = 0XAA = 170

Exemplo: Fazendo um programa para testar os operadores lógicos

```
#include<stdio.h>
#include<conio.h>

main()
{
int a=0xD5, b=0X4B;

clrscr();
printf("a & b = %X\n" , a & b );
printf("a | b = %X\n" , a | b );
}
```

obs. Não há formatadores direto no compilador C para trabalhar com números em binário, então o mais fácil para testar as operações lógicas é utilizar notação hexadecimal.

Ainda existem operadores a nível de byte, que da mesma forma dos operadores relacionais, retornam apenas 2 valores: verdadeiro(True=1) ou falso(false=0).

&&	and
||	or
!	not

Esse tipo de operador tem o objetivo de testar as combinações nas expressões relacionais.
Exemplo:

```
if (x>=3)&&(x<7) //executa o comando se x estiver entre 3 e 6
```

3.6 - PRECEDÊNCIA ENTRE OS OPERADORES

Da mesma forma que na matemática algumas operações tem precedência sobre as outras, o compilador também estabelece uma ordem :

maior

( ) [ ]
! ~ ++ -- - * sizeof
* / %

```
+ -
<< >>
< <= > >=
== !=
&
^
|
= += -= *= %= >>= <<= &= ^= !=
```

menor ,

4 - FUNÇÕES BÁSICAS DA LINGUAGEM C

4.1 - FUNÇÃO `printf`

Sintaxe: `printf("expressão de controle" , argumentos);`

É uma função padrão para exibir uma mensagem , número, caractere ou qualquer tipo de símbolo no dispositivo padrão de saída de dados, no caso a tela do microcomputador.

A expressão de controle pode ser uma mensagem que o programador deseja imprimir na tela, podendo conter também formatadores padrão que indicam o tipo de variável a ser vinculada nessa mensagem. Os argumentos são as próprias variáveis, que podem ser inclusive impressas em forma de operações lógicas ou aritméticas. Cada argumento deve ser separado um do outro por vírgulas (,).

Expressões de controle

\n	new line (pula linha)
\t	tab (tabulação horizontal)
\b	backspace (volta um caractere)
\f	form feed (avanço de página)
\\	imprimir a barra invertida
\‘	imprimir o apóstrofe
\”	imprimir aspas

Formatadores

%d ou %i	inteiro
%f	float
%o	octal
%x	hexadecimal (%X hexadecimal maiúsculo)
%u	inteiro sem sinal (unsigned)
%e	notação científica
%s	string (seqüência de caracteres)
%c	char
%p	ponteiro

%ld ou %li long int
%lf double

Exemplos:

```
main()
{
printf("Cinco é igual a  %d ", 5);
}
```

```
main()
{
int a=5 , b=10;
printf("O valor de  %d  +  %d  é = %d ", a, b, a+b);
}
```

```
main()
{
printf("%s está a  %d está a milhares de milhas\n do sol","venus", 67);
}
```

```
main()
{
printf("%d  %c  %x  %o ", 'A', 'A' , 'A', 'A');
printf("%c  %c  %c  %c ", 'A', 65 , 0x41, o101);
}
```

obs. Cuidar que a tabela ASCII possui 256 valores. Se por acaso passar desse número e tentarmos imprimir em formato caractere (%c) um número maior que 255, será impresso o resto da divisão do número por 256. Se o número for 3393 será impresso A, porque 3393%256 é 65.

Existe ainda uma maneira de limitar o número máximo de algarismos significativos nos formatadores.

- Para arredondamento

```
main()
{
float a=3456,78;
printf("%4.2f\n ",a);
printf("%3.2f\n ",a);
printf("%3.1f\n ",a);
printf("%10.3f\n ",a);
}
```

- Complementando com zeros à esquerda

```
main()
{
printf("%04d\n ",21);
printf("%06d\n ",21);
```

```
printf(“%6.4d\n ”,21);
printf(“%6.0d\n ”,21);
}
```

4.2 - FUNÇÃO `scanf`

Da mesma forma que é importante imprimir dados na tela do microcomputador, há necessidade de entrar dados pelo teclado.

Sintaxe: `scanf(“expressão de controle”, argumentos);`

Na expressão de controle valem os mesmos formatadores da função printf.

Os argumentos são precedidos do & indicando uma operação utilizando endereço de memória para armazenar o valor da variável.

Exemplo:

```
main()
{
int num;

clrscr();
printf(“Digite um número inteiro: ”);
scanf(“%d”,&num);
printf(“O número digitado foi %d ”,num);
printf(“O endereço de memória onde foi armazenado %d é %u”,num,&num);
}
```

Note que o endereço onde foi armazenado a variável num não é familiar, porque o compilador C aloca uma posição de memória livre no micro.

Exemplo de scanf para caracteres

```
main()
{
char nome[50]; //define um vetor de caracteres com tamanho máximo 50 posições

clrscr();
printf(“Digite seu nome completo: ”);
scanf(“%s”,&nome);   // captura uma string do teclado
printf(“O nome digitado foi  %s ”,nome);
}
```

Obs: Note que o scanf necessita do <enter> como confirmação do dado (int, float, char, string, etc.) digitado para continuar.

Se o usuário digitar o nome completo:

Antonio Carlos da Silva

O scanf trunca a string assim que encontra o primeiro espaço em branco, porque o compilador entende que é o fim da string, imprimindo apenas **Antonio** na tela.

Para contornar esse problema utiliza-se outra função, por exemplo:

```
main()
{
char nome[50];

clrscr();
printf(“Digite seu nome completo: ”);
gets(nome); //lê uma string do teclado inclusive espaços em branco
printf(“O nome digitado foi  %s ”,nome);
puts(nome);     // coloca a string digitada no teclado
}
```

A função gets lê tudo o que foi digitado no teclado até pressionar <enter>. A função não armazena o <enter>, mas adiciona um caractere nulo(\o) para indicar o final da string.

4.3 - ENTRADA E SAÍDA DE DADOS BÁSICA

Na linguagem C, a entrada e saída de dados é feita através de funções como printf, puts, scanf, gets, etc. As funções printf e scanf são as mais genéricas, mas exigem a especificação dos formatos de entrada e saída e, por outro lado, as funções que não exigem a especificação do formato operam sobre um único tipo de dado.

Em C++ pode ser utilizado funções (nesse caso chamadas de métodos) da classe istream ou ostream. Para ter acesso a estes métodos é preciso declarar (instanciar) objetos destas classes. O compilador C++ cria objetos cin (istream) e cout (ostream) previamente, de modo que não precisamos fazê-lo explicitamente.

A classe istream redefine o operador ">>" para atuar sobre diferentes tipos de dados permitindo que a formatação correta seja obtida sem a necessidade de especificar explicitamente os tipos de dados de entrada envolvidos. Da mesma forma, a classe ostream redefine o operador "<<" para formatar corretamente a saída de dados. Exemplo:

```
# include <iostream.h>

void main()
{
char nome[50];;

clrscr();
cout << “Digite seu nome completo: ”;
cin >> nome;
cout << "Olá Sr. " << nome << endl << "Tenha um bom dia";
}
```

Note que o problema do caractere SPACE (20h) idêntico ao scanf continua, indicando final da string. Então, se houver necessidade de obter o nome completo deve-se usar o gets. O endl é idêntico ao \n utilizado para pular uma linha.

4.3 - FUNÇÕES `getchar` e `putchar`

É uma função que captura um caractere do teclado (getchar), e o imprime na tela do micro (putchar).

Exemplo:

```
main()
{
char ch;

clrscr();
printf("Pressione uma tecla qualquer ");
ch=getchar();
printf("A tecla pressionada foi: %c\n ",ch);
}
```

obs: Existe uma variação getch() , o qual o caractere não ecoa na tela quando pressionada uma tecla.

Exemplo:

```
main()
{
char ch;

clrscr();
printf("Digite um caractere minúsculo ");
ch=getch();
putchar(toupper(ch));
putchar('\n');
}
```

obs: Há várias funções para manipulação de caracteres como: isalpha( ), isupper( ), islower( ), isdigit( ), isespace( ), tolower( )

5 - ESTRUTURAS CONDICIONAIS

5.1 - INSTRUÇÕES COMPOSTAS

Como foi visto até agora em exemplos, os comandos foram executados de forma única em uma única instrução, mas para que possa executar várias instruções em um único comando é necessária a utilização de uma instrução composta. As instruções compostas consiste de:

- Uma chave aberta ({)
- Uma seqüência de instruções (cada uma terminada por ponto e vírgula)
- Uma chave fechada (})

5.2 - CONDICIONAL `if`

sintaxe:

```
if (condição)
    instrução 1;
  else
   instrução 2;
```

onde a condição pode ser uma operação relacional entre 2 ou mais variáveis. Se a condição for verdadeira (diferente de zero), a instrução 1 é executada, caso contrário a instrução 2 será executada. O else é opcional quando se tratar de uma condição simples de teste.

```
if (condição)
     instrução 1;
```

Exemplos:

```
// proteção para divisões por zero

main()
{
int a, b;

clrscr();
printf("Digite 2 números: ");
scanf("%d%d",&a,&b);
if (b!=0)
      printf("Divisão= %d\n ",a/b);
else
      printf("Divisão por zero");
}
```

5.3 - TYPE CASTING

Note no programa anterior um problema que pode acontecer quanto a utilização dos formatadores. O usuário entra com 2 números, por exemplo: 10 e 2

$$10/2 = 5$$

é um número inteiro. O formatador é %d não há problema no formato de saída. Supondo, porém que o usuário entre com os números 10 e 3.

$$10/3 = 3.3333...$$

o formatador de saída é %d (inteiro), e o resultado é um número em ponto flutuante (float). Vai haver um erro na impressão do resultado na tela porque o compilador não interpreta automaticamente o que é float, ou int e assim por diante.

Para contornar esse problema necessita ser feita uma mudança de tipo (type casting). É uma situação na qual"engana-se" o compilador forçando um determinado tipo de dado a ser interpretado como se fosse outro. A linha correta no exemplo seria:

```
printf("Divisão= %f\n ",float(a/b));
```

Atribuindo no momento da formatação de saída para o vídeo um valor float para a divisão de a por b.

5.4 - CONDICIONAL `if–else–if`

É uma maneira de fazer testes sucessivos quando se tem uma lista de combinações para a variável.
Exemplo:

```
      main()
      {
      float a,b,c;

      clrscr();
      printf(“Digite 3 números: ”);
      scanf(“%f%f%f”,&a,&b,&c);
      if (a<b)
            if(a<c)
            {
            min=a;
            if(b<c)
                {
                 max=b;     // condição teste a < c < b
            else
                 max=c;    // condição teste a < b < c
                }
         else
         {
          min=c;
          max=b;

      else       // condição a > b
           if(a>c)
           {
            max=a;
            if(b>c)
                min=c;
            else
                min=b;
            }
            else
            {
                max=c;
                min=b;
            }
printf  (“maior valor =  %f\n  Menor valor= %f\n”,max,min);
}
```

5.5 - OPERADOR TERNÁRIO

É uma maneira compacta da expressão `if–else` .

Sintaxe: `condição ?  expressão 1 : expressão 2`

Exemplo:

```
main()
{
int x,y,max;

clrscr();
printf("Entre com 2 números inteiros:");
scanf("%d%d",&x,&y);
max=(x>y)? x:y;
printf ("O valor máximo é :  %d\n", max);
}
```

5.6 - OPERADOR `switch`

É uma instrução que permite a seleção de várias opções que dependam do resultado de uma condição ou entrada pelo usuário. Essa instrução pode substituir uma seqüência de condicionais if encadeados.

Sintaxe:

```
Switch(expressão)
{
case  constante 1:
      instrução;
      break;

case  constante 2:
      instrução;
      break;

case  constante 3:
      instrução;
      break;
      .
      .
      .
case  constante N:
      instrução;
      break;

default :
      instruções
}
```

Obs. O teste é feito para n condições. Se não foi encontrada nenhuma das condições executa-se o default.

Exemplo:

```
#include<stdio.h>
#include<conio.h>
```

```
# define   DOM    1
# define   SEG    2
# define   TER    3
# define   QUA    4
# define   QUI    5
# define   SEX    6
# define   SAB    7

       main()
      {
      int dia;

      clrscr();
      printf("Dia da semana (DOM=1, SEG=2, TER=3, ..., SAB=7:" );
      scanf("%d",&dia);
      switch(dia)
            {
            case  SEG:
            case  TER:
            case  QUA:
            case  QUI:
            case  SEX:  puts ("vai trabalhar"); break;
            case  SEG:  printf ("Limpe o jardim e:");
            case  DOM: printf ("Relaxe !"); break;

            default : puts ("Essa dia não existe!");
            }
      }
```

6 - LAÇOS

Laços (ou em inglês - loops) são estruturas utilizadas quando houver a necessidade de sucessivas repetições durante, por exemplo, uma condição de testes. Pode ser utilizado também para implementar cálculos sucessivos (séries matemáticas), incrementando ou decrementando variáveis ou funções.

6.1 - LAÇO `for`

Sintaxe:

```
for (inicialização; condição; incremento/decremento)
      comando;
```

onde:

inicialização : normalmente é uma atribuição a uma variável índice

condição: é a expressão de teste sobre a condição da variável índice, em relação a um parâmetro de comparação ou igualdade.

Incremento/decremento: alguma modificação (aumentando ou diminuindo) a variável índice a fim/ou não de estabelecer a condição

Exemplo de laço for

```
 main()
{
int x;

clrscr( );
printf("Imprime a seqüência de 1 a 1000 na tela \n" );
for(x=1; x<=1000; x++)
      printf("%d\n",x);    // imprime o  nº e pula uma linha
}
```

Obs: note que o ponto e vírgula vem somente no final do comando (nesse exemplo o printf).

O resultado da execução do programa é uma seqüência impressa na tela de 1 até o número 1000, porque a condição incrementa o x (x++). Se quisesse fazer uma contagem regressiva de 100 para 1, a linha de comando muda para:

```
      for(x=1000; x>=1; x--)
```

Se houver a necessidade de um incremento maior que 1 à cada passagem. É só fazer:

```
      for(x=1; x>=1000; x+=2)  //incrementa de 2 em 2 à cada passagem
```

Exemplo:

```
// programa para calcular a média

   main()
   {
   unsigned n,i;
   float valor, soma = 0.0;

   clrscr();
   printf("Qual o número de valores para o cálculo da média? \n" );
   scanf("%u",&n);
   for(i=1; i<=n; i++)
         {
         printf("Valor %3u: ",i);
         scanf("%f",&valor);
         soma+=valor; // a cada novo valor incremento a variável soma
         }
   if(n!=0)  printf("Média= %10.4f\n ",soma/n);
   }
```

6.2 - LAÇO `while`

É o laço mais simples de todos. Normalmente um laço substitui o outro, porém alguns trazem vantagens ou facilidades em relação aos outros dependendo da aplicação.

Sintaxe:
```
while(condição)
      instrução;
```

O comando while testa a condição e executa a instrução, ou conjunto de instruções, no caso da instrução composta. Terminando a execução das instruções, retorna à condição, executa instrução novamente, e assim sucessivamente. Dependendo da condição imposta pode se tornar um loop infinito:

```
#include<stdio.h>
#include<conio.h>
#include<dos.h>    // mais uma diretiva de compilação

      main()
      {
      int x=100;

      clrscr();
      while (x!=0)  // sai do loop quando x for igual a zero
            {
            printf("loop infinito");
            delay(10); //instrução de atraso
                        //Gasta 10 m segundos nessa linha
            }
      }
```

Obs: o programa é chamado de loop infinito porque nunca a variável x será igual a zero

Para que o programa tenha sentido é necessário estabelecer um incremento ou decremento na variável x.

```
      while(x!=0)  // sai do loop quando x for igual a zero
            {
            printf("loop finito");
            delay(10); //instrução de atraso.
                        //Gasta 10 m segundos nessa linha
            x --;
            }
```

6.3 - LAÇO `do-while`

A principal diferença do do-while em relação ao while é que os comandos existentes no laço do-while sempre são executados ao menos uma vez, porque a condição de teste está no final do loop.

Sintaxe:
```
do
{
instruções;
}
while(condição);
```

Exemplo:

```
#include<stdio.h>
#include<conio.h>
# define  ENTER  '\r' //caractere de controle ASCII da tecla <enter>

       main()
      {
      char ch;
      int comp;

      clrscr();
      do
        {
         comp = 0;
         puts ("digite uma frase e pressione <enter>");
         while((ch=getch()) != ENTER)
            {
            putch(ch);
            comp ++;
            }
         printf("\n Sua frase tem  %d  letras \n",comp);
         }
        while(comp);  // essa sintaxe eqüivale a while(comp == 0)
      }
```

Esse programa sofre de algumas deficiências. Por exemplo se o usuário erra a digitação e pressiona o backspace a fim de corrigir, a tecla do backspace é contada como uma tecla pressionada e não corrige o erro anterior.

Para corrigir essa deficiência do programa basta imprimir um espaço em branco(\b) seguido do backspace. A contagem, porém, continua alterada.

6.4 - INSTRUÇÕES `break e continue`

Quando um comando break é encontrado dentro de um laço qualquer, automaticamente é interrompida a execução do laço. Por exemplo:

```
      main()
      {
      char ch;

      clrscr();
      for (;;)   //é uma sintaxe alternativa de loop infinito
         {
         ch=getche(); //getche ecoa o caractere na tela
         if(ch=='a')  break; // ao pressionar a tecla a o laço for termina
          }                        //imediatamente
        }
```

O comando continue faz com que o laço seja interrompido e volte ao início do laço novamente.

Exemplo:

```
main()
{
int num_notas, num_alunos, turma, aluno;
float nota, total;

clrscr();
printf(“Número de turmas:”);
scanf(“%d”,&num_turmas);
for(turma=1; turma<=num_turmas; turma++)
      {
      printf(“Turma %d\n ”, turma);
      printf(“Número de alunos”);
      scanf(“%d”, &num_alunos);
      aluno=1;
      total=0.0;
      while(aluno<=num_alunos)
            {
            printf(“Nota de aluno %4d: ”,aluno);
            scanf(“%f”, &nota);
            if(nota<0.0 || nota>10.0)
                  {
                  puts (“Nota inválida”);
                  continue;
                  }
            total+=nota;
            aluno++;
            }
      printf(“Média da turma %d =“);
      if( num_alunos && aluno > num_alunos)
            printf(“%4.1f\n”, total/num_alunos);
      else
            printf(“Não calculada \n”);
      }
 }
```

Obs. Quando há uma nota inválida digitada pelo usuário, ao invés de finalizar o laço while, simplesmente o compilador assume o erro ocorrido e retorna ao início do laço. Como não foi incrementado esse aluno no contador (aluno++), vai ser perguntada a nota novamente referente ao mesmo aluno.

7 - PONTEIROS

É uma variável que contém o endereço de outra variável. Em alguma situações deseja-se saber onde determinados dados estão armazenados, ao invés de, simplesmente, seu valor. Os ponteiros são utilizados para alocação dinâmica (requisição de determinada quantidade de memória em bytes), podendo substituir matrizes com mais eficiência. A declaração de ponteiros é feita da seguinte forma:

```
int x, *px;
```

px=&x; // a variável px aponta para x

Se houver necessidade de utilizar o conteúdo da variável para qual o ponteiro aponta:

y= *px;

que significa o mesmo que:

y=x;

Exemplo:

```
# include<iostream.h>
void main()
{
int ivar, *iptr;
iptr = &ivar;
ivar = 421;
cout << "endereço de ivar: " << &ivar << endl;
cout << "conteúdo de ivar: " << ivar << endl;
cout << "conteúdo de iptr: " << iptr << endl;
cout << "valor apontado: "   << *iptr << endl;
}
```

Ao executar esse programa obteremos um resultado semelhante a esse:

endereço de ivar: 0x0064fe00
conteúdo de ivar: 421
conteúdo de iptr: 0x0064fe00
valor apontado: 421

8 - VETORES E MATRIZES

8.1 - VETORES

Vetores também são chamados de arranjos unidimensionais. São listas ordenadas de determinados tipos de dados, iniciam com índice 0 (primeiro elemento do vetor) e vai até o último elemento declarado na variável, por exemplo:

```
main()
{
int vetor[10];
int x;
for(x=0; x<10; x++)
      {
      vetor[t]=x;
      printf("%d\n", vetor[t]);
      }
}
```

Foi criada uma variável de nome vetor que será uma seqüência de números inteiros com um tamanho máximo de 10 elementos. O laço for ordena a variável com seus valores, imprimindo na tela linha a linha.

Um exemplo similar para cálculo de média pode ser feito utilizando um vetor para armazenar todas as notas em apenas uma variável.

```
main()
{
float nota[20], soma;
int  x;

for( x=0; x<20; x++) // instrução composta; foram abertas novas {}
      {
      printf("Digite a nota do aluno");
      scanf("%f",&nota[x]);
      }
soma=0.0;
for(x=0; x<20; x++) //instrução simples;não necessita abrir novas{}
      soma+=nota[x];
printf("Média das notas %3.2f ",soma/20);
}
```

8.2 - MATRIZES

São vetores (ou arranjos) multidimensionais. A idéia é criar 2 vetores e montar dois laços encadeados. Funciona como um vetor, porém possui mais de um índice.

Exemplo:

```
// programa para cálculo de matrizes N x N

main()
{
unsigned  m, n, i, j;
float matriz, x[10][10];

clrscr();
printf("Cálculo de matrizes \n\n");
printf("Digite o número de linhas");
scanf("%u",&m);
printf("Digite o número de colunas");
scanf("%u",&n);
for(i=1; i<=m; i++)
      for(j=1; j<=n; j++)
      {
      x[i][j]=0;
      printf("X[%d][%d]=",i,j);
      scanf("%f",&x[i][j]);
      }          // até aqui houve a leitura dos dados
                 // do teclado e armazenado na matriz mxn

for(i=1; i<=m; i++)      // executo o mesmo laço anterior para imprimir
      for(j=1; j<=n; j++)  //os dados inseridos pelo usuário no teclado
      printf("X[%d][%d] = %3.2f\n",i,j,x[i][j]);
}
```

Obs: Cuidado deve se tomar ao definir o tamanho máximo do vetor ou matriz (x[10] por exemplo). Se o laço for estiver acima do limite (i=0; i<20, i++) o compilador vai alocar um espaço em memória para armazenar esse dado do vetor. Como esse espaço não foi reservado quando foi definida a variável, o programa não vai funcionar, podendo, inclusive travar o compilador.

9 - FUNÇÕES

Por enquanto só houveram programas utilizando o corpo da função principal(main), porém pode-se criar blocos isolados contendo comandos ou instruções que executem etapas do programa, denominadas de funções.

A sintaxe da função é a seguinte:

```
<tipo de variável><nome da função>(parâmetro1,parâmetro2,...,parâmetron)
   {    //início da função

   ...instruções

   }      //final da função
```

Exemplo:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>

 void area_quadrado()
{
float area,a;

printf("Entre com o lado do quadrado:\n");
scanf("%f",&a);
area=pow(a,2); //area quadrado é l²
printf("A área do quadrado é: %f",area);
}
void main()
{
area_quadrado(); // chamada da função
}
```

Obs: o nome da função é area_quadrado, e dentro dessa função foi criada duas variáveis locais (area, a). São denominadas locais porque seus valores de entrada são válidos apenas dentro da função area_quadrado. No momento que o compilador deixa a função, em direção ao programa principal, ou mesmo a uma outra função, o valor dados às variáveis (area, a) dentro da função area_quadrado, não existem mais. Inclusive se em outra função estiverem definidas variáveis com o mesmo nome (area, a) serão consideradas como novas variáveis existentes no programa em um outro local.

A função pode ser escrita também da seguinte forma:

```
#include<stdio.h>
#include<conio.h>
```

```
#include<math.h>

    float area_quadrado(float a)
   {
   float area;

   area=pow(a,2);
   return area;
   }

   void main()
   {
   float b;
   printf("Entre com o lado do quadrado:\n");
   scanf("%f",&a);
   b=area_quadrado(a); // chamada da função
   printf("%f",b);
   }
```

Quando há a chamada da função, a mesma transporta uma variável que será utilizada dentro da função. A variável `area` é calculada e retorna o valor ao programa principal quando houver a chamada da função `area_quadrado`. Para acompanhar a execução do programa passo-a-passo utilize a tecla F7, que entra dentro das funções.

10 - MODO GRÁFICO EM LINGUAGEM C

O modo usual da execução de um programa em linguagem C é o modo texto, ou seja, 80 colunas por 25 linhas. Para formar o caractere no modo texto é utilizada a composição de *pixels*, ou elementos de imagem.

Para utilizar o compilador C/C++ no modo gráfico há de utilizar a biblioteca *GRAPHICS.LIB*, através da diretiva *GRAPHICS.H*. O cabeçalho de inicialização do modo gráfico é o seguinte:

```
# include <graphics.h>

void main
{

// requisita autodetecção do modo gráfico
int gdriver = DETECT, gmode, errorcode;

//inicialização do modo gáfico
initgraph(&gdriver,&gmode,"c:\bc\bgi\egavga.bgi");

}
```

Obs. o diretório relacionado entre aspas deve ser o local que contenha os *drivers* de vídeo específicos do compilador.

A partir da execução desses comandos, o compilador trabalha no modo gráfico de 640x480 pixels. As coordenadas X e Y da posição inicial (0,0) estão no canto superior esquerdo do vídeo. Se desejar mostrar um pixel na tela do micro, rode o seguinte exemplo:

```
# include <graphics.h>

void main
{

// requisita autodetecção do modo gráfico
int gdriver = DETECT, gmode, errorcode;

//inicializa o modo gráfico e suas variáveis
initgraph( &gdriver, &gmode, "c:\bc\bgi\egavga.bgi");

putpixel (320, 240, 15); // x=320, y=240 e 15 é a cor do pixel = branca

getch();

//comando para fechar o modo gráfico
//e retornar o compilador ao modo texto
closegraph();
}
```

A tabela de cores para utilização do modo gráfico pode ser conseguida no *help* COLORS.

Através do modo gráfico é permitido a confecção de janelas em diversas cores e formas, círculos, arcos, linhas, *strings* de diversos tamanhos e modelos de fontes, etc.

11 - FUNÇÕES PARA ACESSO AO HARDWARE DO PC

Os comandos na linguagem C para acessar as portas de hardware no IBM PC são o INPORT (para entrada de dados) e OUTPORT (para saída de dados). Esse comando lê ou envia, respectivamente, uma WORD, ou seja 16 bits.

Existe uma variação em relação aos comandos anteriores, muito útil a processamento com largura de dados de 8 bits, que são as funções INPORTB e OUTPORTB, que trabalham com largura de um Byte.

Sintaxe:

unsigned inport (unsigned Endereço da Porta);

unsigned char inportb (unsigned char Endereço da Porta);

void outport (unsigned Endereço da Porta, unsigned Valor Desejado);

void outportb (unsigned char Endereço da Porta, unsigned char Valor Desejado);

Exemplo:

```
# include <stdio.h>
# include <dos.h> //diretiva necessária para as funções de acesso ao hardware

void main()
{
//coloca em "1" todos os 8 bits de dados da porta 378h
outportb(0x378, 0xFF);

delay(1000); //demora 1000 milissegundos para executar essa linha

// coloca em "0" todos os 8 bits de dados da porta 378h
outportb(0x378, 0x00);
}
```

BIBLIOGRAFIA

- Apostila Linguagem C – Wilson H. Bogaddo, CEFET,PR, 2000
- Schildt, Herbert: *C Avançado*, McGraw-Hill, SP, 1989
- Introdução a Linguagem C –Versão 2.0 – Centro Computação UNICAMP
- www.beyondlogic.org – acesso em 15/05/2000
- Data-sheet LCD ALFACON