LINGUAGEM “C”

Estrutura básica de um programa

Um programa em linguagem “C” é constituído por uma sequência de funções (módulos) que em conjunto irão permitir resolver o problema proposto. Estas funções contêm instruções que especificam as tarefas que o processador deve executar.

A execução de um programa começa por uma função de nome ***main*** que deverá existir sempre num programa escrito em linguagem “C”. Todas as instruções terminam com “**;**”.

Directivas de Pré-Compilação

```
#include <stdio.h>
#include ”prog1.h”
#define ALUNOS 15
```

Declaração de novos tipos e de variáveis globais

```
typedef float numero_real;
int a;
```

Definição das funções

```
int ler() {
      int  x;
      scanf(“%d”,&x); return(x);
}

void escreve(x) {
      printf(“%d”,x);
}
```

Função Principal - main

```
main()
{
      int valor;
      valor=ler();
      escreve(valor);
}
```

Tipos de dados

A declaração de uma variável em linguagem “C” implica a sua atribuição a um dado tipo de dados. Essa especificação para além de caracterizar o conteúdo da variável perante a informação que ela irá conter permite também ao compilador reservar um espaço de memória condizente com o tamanho necessário para o seu armazenamento.

Tipo de dados	Informação	Tamanho reservado
char	Caracter	1 byte
int	Inteiro	2 bytes
float	Virgula flutuante	4 bytes
double	Virgula flutuante	8 bytes
short int	Inteiro	1 byte
long int	Inteiro	4 bytes
signed char/int	Com sinal	1/2 bytes
unsigned char/int	Sem sinal	1/2 bytes

Desta forma numa variável do tipo **short int** poderá ser armazenado um qualquer valor entre 0 e 255, já que com um byte o maior valor é $11111111_{(2)}$ = 255.

Um **signed char** armazenam-se toma valores entre –127 e 127, e um **int** tem por valor máximo de armazenamento 32767.

Exemplos:

```
int a,b;
char v;
unsigned int valor;
```

Operações de Entrada/Saída (E/S)

A linguagem “C” por si só não possui comandos para efectuar a entrada e a saída de informação num programa. É portanto necessário fazer uso de funções externas que efectuem estas tarefas.

Tais funções estão definidas numa biblioteca da linguagem de nome **stdio.h** (*Standard Input/Output header file*) pelo que qualquer programa que faça uso delas terá de incluir a directiva ao pré-compilador

#include <stdio.h>

Funções de E/S

- *getchar* - Permite ler o próximo caracter do teclado

```
int c;
c=getchar();
```

- *putchar* – Escreve um caracter no ecran

```
putchar(c)
```

- *scanf* – Entrada formatada

```
int a;
char b;
scanf(“%d %c”,&a,&b);
```

- *printf* – Saída formatada

```
printf(“A nota do aluno %ld é %d\n”,numero,nota);
```

Especificadores de Conversão

Especificador	Descritivo
%d	Inteiro
%f	Vírgula Flutuante
%c	Caracter
%s	String
%ld	Inteiro longo

Caracteres especiais

Caracter	Descritivo
\n	*newline*
\t	*tab*
\b	*backspace*

Atribuição

O operador atribuição representa-se pelo "**=**" e a sua sintaxe é:

variável = expressão

Numa perspectiva mais técnica a representação acima poderia ser lida da seguinte forma:

"*O conteúdo do endereço da variável será preenchido com o valor da expressão*"

Exemplos:

```
a = 5;
b = a;
c = sin(x)+ln(23+abs(y)%5);
a = b = c = 0;
```

Operadores

Aritméticos:

+	Adição
-	Subtracção
*	Multiplicação
/	Divisão
%	Módulo

Exemplos:

- 2+4.5 = 6.5
- 4 – 3 = 1
- 2 * 3 = 6
- 15 / 2 = 7
- 15.0 / 2 = 7.5

Incremento e Decremento:

++	Incremento
--	Decremento

Estes operadores podem ser usados de forma prefixa e de forma sufixa.

Forma prefixa: a actualização do valor acontece antes da utilização da variável;

Forma sufixa: a actualização do valor é feito após a utilização da variável.

Exemplos:

- y = x++ ⇔ y=x; x=x+1;
- y = --x ⇔ x=x-1; y=x;

Relacionais:

x < y	Menor que
x <= y	Menor ou igual
x > y	Maior que

x >= y	Maior ou igual
x == y	Igual
x != y	Diferente

Lógicos:

&&	e (and)
||	ou (or)
!	negação (not)

Manipulação de bits:

~	Complemento
<<	Shift para esquerda
>>	Shift para direita
&	Conjunção
|	Dijunção
^	Ou exclusivo

Estruturas de Decisão

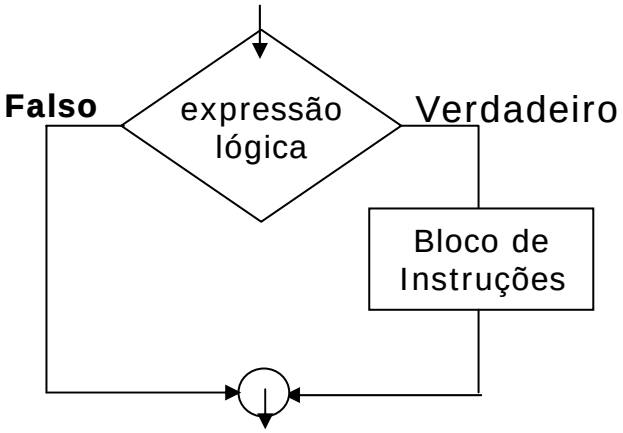
A implementação algorítmica da resolução de um problema implica eventuais tomadas de decisão sobre o modo como este deverá ser solucionado. As instruções da linguagem que permitem implementar estas construções são chamadas **Estruturas de Decisão**.

Instrução IF

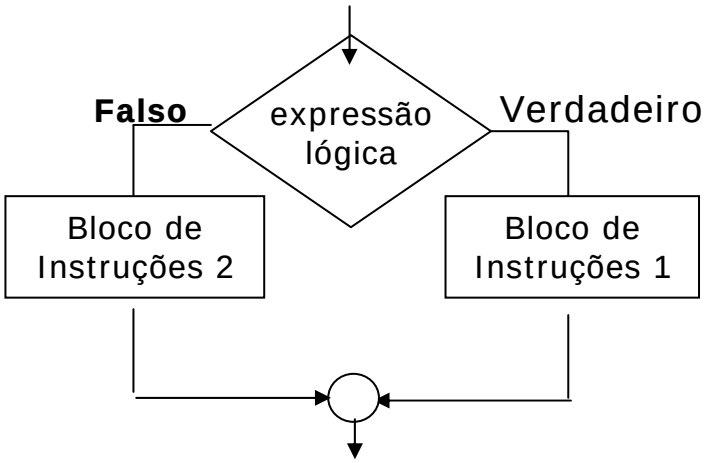
Esta instrução permite executar um conjunto de tarefas em função da avaliação do valor lógico de uma condição.

As duas construções possíveis são:

if(*expressão* **)**
*Bloco de Instruções***;**



if(*expressão* **)**
*Bloco de Instruções 1***;**
else
*Bloco de Instruções 2***;**



Exemplo 1:

Elaborar em Linguagem C um programa que permita calcular a raiz quadrada de um qualquer número real.

```
#include<stdio.h>

main()
{
float x;
        int complexo=0;

        scanf("%f",&x);

        if( x<0 )
        {
                x =-x;
                complexo=1;
        }

        printf("a raiz quadrada é %f",sqrt(x));

        if(complexo==1)
                printf("i");
}
```

Exemplo 2:

Elaborar em Linguagem C um programa que permita verificar se um dado número inteiro é par ou ímpar.

```
#include<stdio.h>

main()
{
        int numero;

        scanf("%d",&numero);

        if( numero%2 == 0)
                printf("%d é par\n",numero);
        else
                printf("%d é impar\n",numero);
}
```

Instrução SWITCH

Esta estrutura de decisão múltipla deverá ser utilizada quando a tarefa a realizar pelo programa depende do valor de uma variável (ou expressão.

A sintaxe geral é:

```
switch (variável) {
    case hipótese 1:        bloco de instruções 1;
                            break;

    case hipotese 2:        bloco de instruções 2;
                            break;

    (…)

    case hipotese n:        bloco de instruções n;
                            break;

    default:                bloco de instruções;
}
```

Esta instrução avalia uma expressão do tipo inteiro (ou caracter) e compara o valor obtido com as hipóteses afectas aos diversos *cases* se alguma das comparações suceder então executa o respectivo bloco de instruções. Se nenhuma das comparações for verdadeira a instrução faculta a existência de uma hipótese *default* com o respectivo bloco de instruções.

A instrução break é necessária para terminar a execução do bloco de instruções que se pretende executar. Caso esta fosse omitida todas as instruções de todos os blocos abaixo do pretendido seriam executadas.

Exemplo:

Elaborar em Linguagem C um programa que escreva o nome do mês introduzido na forma numérica.

```
#include<stdio.h>

main()
{
        int mes;

        scanf(“%d”,&mes);

        switch(mes){
                case 1 : printf("Janeiro");break;
                case 2 : printf("Fevereiro");break;
                case 3 : printf("Março");break;
                case 4 : printf("Abril");break;
                case 5 : printf("Maio");break;
                case 6 : printf("Junho");break;
                case 7 : printf("Julho");break;
                case 8 : printf("Agosto");break;
                case 9 : printf("Setembro");break;
                case 10 : printf("Outubro");break;
                case 11 : printf("Novembro");break;
                case 12 : printf("Dezembro");break;
                default : printf("Mês Invalido");
        }
}
```

Estruturas de Controlo do Fluxo
CICLOS

A implementação de um problema leva muitas vezes a uma “análise” mais ou menos exaustiva de um bloco de informação. A referida “análise” implica a repetição de um conjunto de tarefas para cada elemento da informação.

As instruções da linguagem que permitem implementar estes raciocínios são denominadas de **ciclos**.

Os ciclos subdividem-se em:

Ciclos Contados: São estruturas cíclicas onde o número de repetições das instruções que constituem o bloco de acção está a partida definido.
Em linguagem C este tipo de ciclo é implementado pela instrução **FOR**.

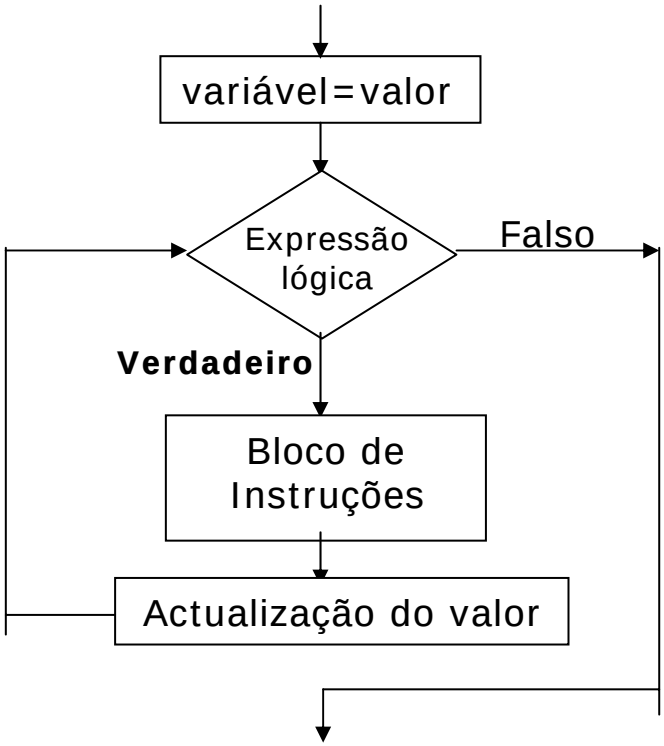
Ciclos Condicionais: Nestas estruturas o número de repetições das instruções do bloco de acção está dependente de factores externos, pelo que não é conhecido à partida.
Em linguagem C este tipo de ciclo é implementado pelas instruções **WHILE** e **DO ... WHILE**.

Ciclo FOR:

Este ciclo possui uma construção muito versátil.

A sintaxe mais usual é:

for(*variável=valor* **;** *expressão lógica* **;** *actualização do valor***)**
Bloco de Instruções



A instrução suporta três parâmetros embora nenhum deles seja obrigatório. No primeiro é usual fazer-se a inicialização da variável controladora do ciclo, no segundo estipula-se qual a condição de fim de ciclo, e no último actualiza-se a variável controladora.

Como futuramente veremos esta estrutura pudera transforma-se facilmente numa estrutura do tipo ciclo WHILE ...

Existem duas instruções que permitem alterar a execução normal do ciclo.

BREAK

Interrompe definitivamente a execução do ciclo.
Também é utilizado na instrução *Switch*.

CONTINUE

Interrompe a execução do bloco de acção do ciclo iniciando a iteração seguinte.

Exemplo 1:

Imprimir todos os números de 1 até 10000.

```
#include <stdio.h>

main()
{
 int a;
 for(a=1;a<=10000;a++)
        printf("%d\n",a);
}
```

Exemplo 2:

Verificar se um dado número inteiro é primo.

```
#include <stdio.h>

main()
{
 int a,b,primo=0;

 scanf("%d",&a);
 for(b=2;b<=a/2;b++)
        if( a%b == 0){
                primo=1;
                break;
        }

 if(primo==0)
        printf("%d é primo",a);
 else
        printf("%d não é primo",a);
}
```

Versatilidade da sintaxe do ciclo FOR

1.

```
(...)
int a;

printf(“Insira um valor:”);
scanf(“%d”,&a);

for(;a<1000;a++)
   printf(“%d\t%d",a,2*a);
(...)
```

2.

```
(...)
int a;

for(a=1;;a+=5){
  if(sin(a)==ln(a)*sin(a))
break;
printf(“%d",a);
(…)
```

3.

```
(...)
int a;
for(a=1;a<500;)
printf(“%d",a++);
(…)
```

4.

```
(...)
int a;

for(a=1;a<10;
          printf(“%d",a++));
(…)
```

5.

```
(...)
int a;

for(;;)
   printf(“%d",a++);
(…)
```

6.

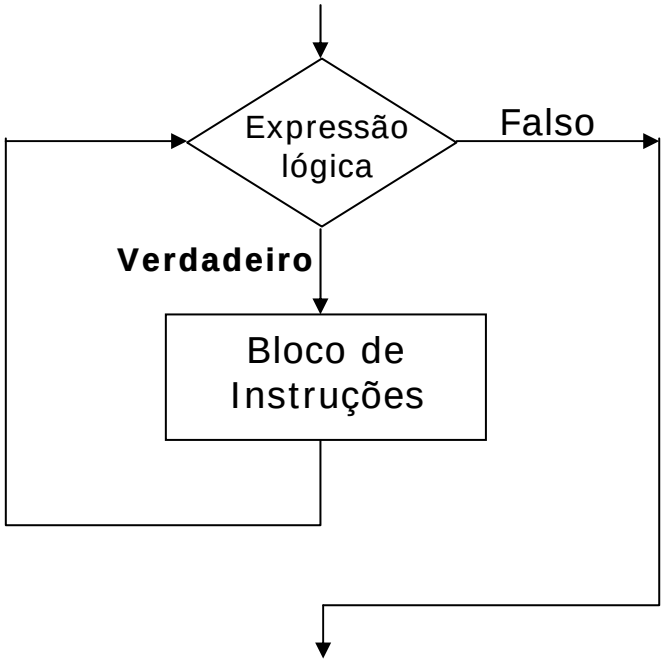
```
(...)
int a=1;

for(;a<10;)
   printf(“%d",a++);
(…)
```

Ciclo WHILE

Esta instrução da linguagem permite executar um conjunto de instruções enquanto uma dada condição lógica for verdadeira.

A sintaxe é:

while(*expressão lógica***)**
Bloco de Instruções



A avaliação do valor da condição lógica é feita no inicio do ciclo, pelo que, se a condição for á partida falsa o bloco de instruções nunca será executado.

EXEMPLO:
Elaborar um programa em linguagem C que permita calcular a média de uma sequência de valores inseridos pelo utilizador. A sequência termina obrigatoriamente com a inserção de um 0 que não deverá contar para a avaliação da média.

Ex: 2 45 6 –1 78 9 0 $\Rightarrow$ média=23.16667

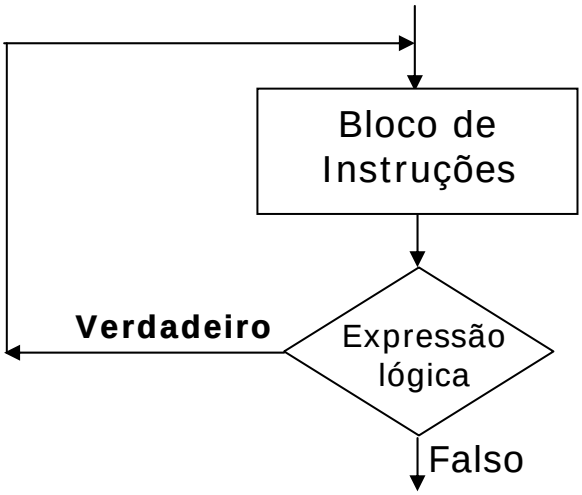
```
#include <stdio.h>

main()
{
int contador=0, soma=0, valor;
float média;

scanf("%d",&valor);

while(valor != 0){
        contador++;
        soma += valor;
        scanf("%d",&valor);
}

if(contador==0)
        média=0;
else
        média=(float)soma/contador;

printf("A média da sequência inserida é %f\n",media);
}
```

Ciclo DO ... WHILE

Esta instrução da linguagem permite executar um conjunto de instruções até que uma dada condição lógica fôr verdadeira.

A sintaxe é:

do
Bloco de Instruções
while(*expressão lógica***)**



A avaliação do valor da condição lógica é feita no fim do ciclo, pelo que, o bloco de instruções será sempre executado pelo menos uma vez.

EXEMPLO:
Elaborar um programa em linguagem C que permita calcular a média de uma sequência de valores inseridos pelo utilizador. A sequência termina obrigatoriamente com a inserção de um 0 que não deverá contar para a avaliação da média.

Ex: 2 45 6 –1 78 9 0 $\Rightarrow$ média=23.16667

```
#include <stdio.h>

main()
{
int contador=0, soma=0, valor;
float média;

do{
        scanf("%d",&valor);
        contador++;
        soma += valor;
} while(valor != 0)

if((--contador)==0)
        média=0;
else
        média=(float)soma/contador;

printf("A média da sequência inserida é %f\n",media);
}
```

CICLOS EMBUTIDOS

Um ciclo diz-se embutido se faz parte integrante do bloco de instruções de outro ciclo.

Por exemplo:

```
(...)
for(i=1;i<4;i++)
 for(j=1;j<4;j++)
        printf("%d\n",i-j);
(...)
```

i	j	saída
1	1	0
1	2	-1
1	3	-2
2	1	1
2	2	0
2	3	-1
3	1	2
3	2	1
3	3	0

As instruções **break** e **continue** da linguagem C actuam sobre o ciclo do qual fazem parte.

EXEMPLO:
Encontrar todos os números capicua entre 10 e 10000.
*Defina-se por **capicua** um número que é lido da mesma forma de trás para a frente e de frente para trás*.

```
#include<stdio.h>

main()
{
int numero, valor, digito, inverso;

for(numero=10; numero<=10000; numero++)
        {
        valor=numero;
        inverso=0;
        while(valor)
        {
                digito=valor%10;
                inverso=10*inverso+digito;
                valor=valor/10;
        }
        if(inverso==numero)
                printf("%d\n",numero);
      }
}
```