



Cap.4: Design de Algoritmos e Programação Estruturada

Algoritmos, fluxogramas e
pseudo-código



Sumário

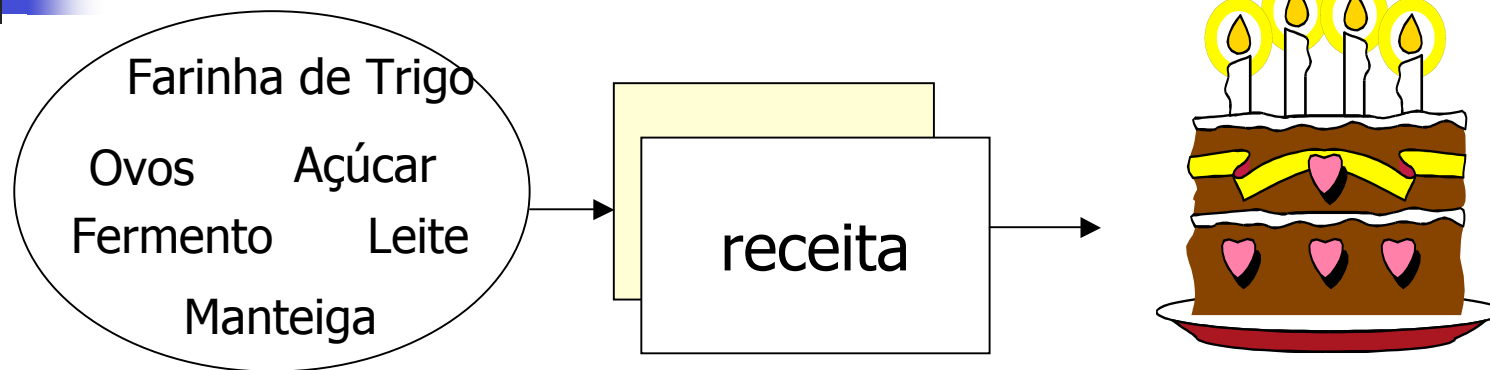
- Problemas e algoritmos
- Desenho de algoritmos/programas
- Passos na construção de algoritmos
- Método Cartesiano de Dividir-Para-Conquistar
- Características fundamentais dum algoritmo
- Representação de algoritmos
- Fluxogramas e programação visual
- Estruturas de controlo de fluxo: sequência, selecção e repetição
- Programação estruturada



Problemas & Algoritmos

- Para resolver um **problema** através dum computador é necessário encontrar em primeiro lugar uma maneira de descrevê-lo de uma forma clara e precisa.
- É também preciso que encontremos uma sequência de passos que conduzam à sua resolução. Esta sequência de passos é designada por **algoritmo**.
- A noção de algoritmo é central para toda a informática.
- A criação de algoritmos para resolver os problemas é uma das maiores dificuldades, mas também um dos desafios mais atractivos, dos iniciados em programação em computadores.

Problema: Fazer um bolo?

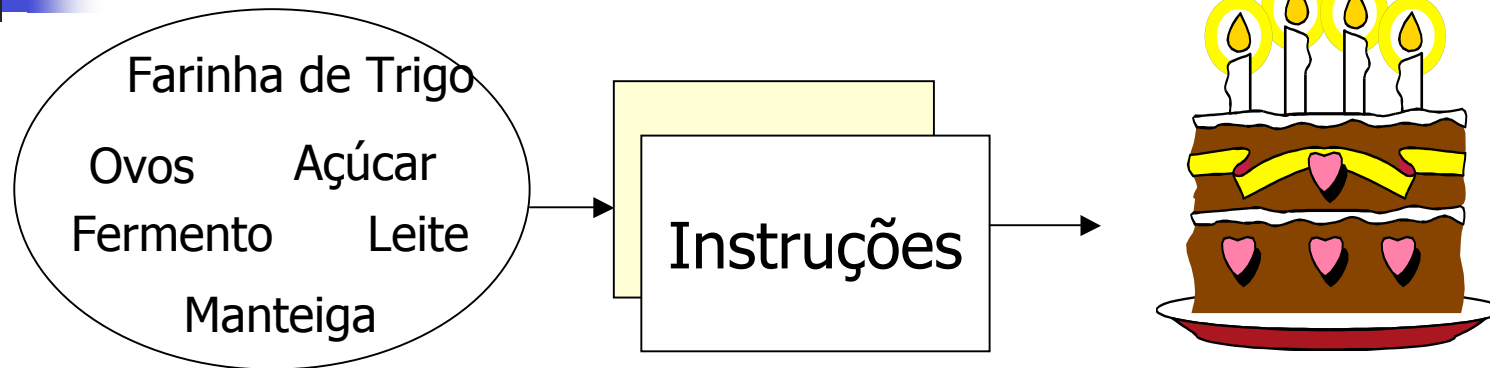


Uma receita é uma descrição dum conjunto de **passos** ou **acções** que fazem a combinação dum conjunto de ingredientes com vista a obter um produto gastronómico particular.

Algoritmo:

Como fazer um bolo?

Um algoritmo opera sobre um conjunto de entradas (farinha ovos, fermento, etc. no caso do bolo) de modo a gerar uma saída que seja útil (ou agradável) para o utilizador (o bolo pronto).



Algoritmo (receita de bolo):

- 1) Bater duas claras em castelo;
- 2) Adicionar duas gemas;
- 3) Adicionar um xícara de açúcar;
- 4) Adicionar duas colheres de manteiga;
- 5) Adicionar uma xícara de leite de coco;
- 6) Adicionar farinha e fermento;
- 7) Colocar numa forma e levar ao forno em lume brando.

Desenho de algoritmos/programas

- De um modo geral, considera-se que um algoritmo é uma descrição, passo-a-passo, de uma metodologia que conduz à resolução de um problema ou à execução de uma tarefa.
- A programação consiste na codificação precisa desse algoritmo, segundo uma linguagem de programação específica.
- Há, pois, que ter em consideração que existem três fases distintas na elaboração de programas:
 - a análise do problema (especificação do problema, análise de requisitos, pressupostos, etc.)
 - a concepção do algoritmo
 - a tradução desse algoritmo na linguagem de programação





Passos na construção de algoritmos

- Compreender o problema
- Identificar os dados de entrada
- Identificar os dados de saída
- Determinar o que é preciso para transformar dados de entrada em dados de saída:
 - usar a estratégia do **dividir-para-conquistar**
 - observar regras e limitações
 - identificar todas as acções a realizar
 - eliminar ambiguidades
- Construir o algoritmo
- Testar o algoritmo
- Executar o algoritmo



Método Cartesiano de Dividir-Para-Conquistar

- Também é o conhecido por **método descendente** (top-down method) ou método de **refinamento passo-a-passo**
- Este método consiste em dividir um problema em partes menores (ou sub-problemas) de modo a que seja mais fácil a sua resolução.
- **Exemplo:** Fazer sumo de laranja?
 - Lavar laranja;
 - Partir laranja ao meio;
 - Espremer laranja;
 - Filtrar o sumo;
 - Servir o sumo.
- Passo-a-passo, significa que cada passo é completado antes que o próximo comece.
- **Exemplo:** é impossível “ver telejornal” antes de executar por inteiro o passo anterior de “ligar a TV”



Características fundamentais dum algoritmo

Um algoritmo deve ter 5 características fundamentais:

- **Finitude:**um algoritmo deve sempre terminar após um número finito de passos.
- **Definição:** cada passo de um algoritmo deve ser precisamente definido. As acções devem ser definidas rigorosamente e sem ambiguidades.
- **Entradas:** um algoritmo deve ter zero ou mais entradas, isto é quantidades que lhe são fornecidas antes do algoritmo iniciar.
- **Saídas:** um algoritmo deve ter uma ou mais saídas, isto é quantidades que tem uma relação específica com as entradas.
- **Eficiência:**Um algoritmo deve ser eficiente. Isto significa que todas as operações devem ser suficientemente básicas de modo que possam ser em princípio executadas com precisão em um tempo finito por um ser humano usando papel e lápis.

NOTA: Pode haver mais do que um algoritmo para resolver um problema.

Por exemplo, para ir de casa até o trabalho, posso escolher diversos meios de transportes em função do preço, conforto, rapidez, etc..



Representações de algoritmos

- **Linguagem Natural**

Os algoritmos são expressos directamente em linguagem natural (e.g. o português como no exemplo do bolo).

- **Fluxograma (ou Diagrama de Fluxo)**

Esta é uma representação gráfica que emprega formas geométricas padronizadas para indicar as diversas acções e decisões que devem ser executadas para resolver o problema.

- **Pseudo-linguagem**

Emprega uma linguagem intermediária entre a linguagem natural e uma linguagem de programação para descrever os algoritmos.

Não existe consenso entre os especialistas sobre qual é a melhor maneira de representar um algoritmo. Actualmente a maneira mais comum de representar algoritmos é através de uma pseudo-linguagem ou pseudo-código. Esta forma de representação tem a vantagem de o algoritmo seja escrito de uma forma que está próxima de uma linguagem de programação de computadores.



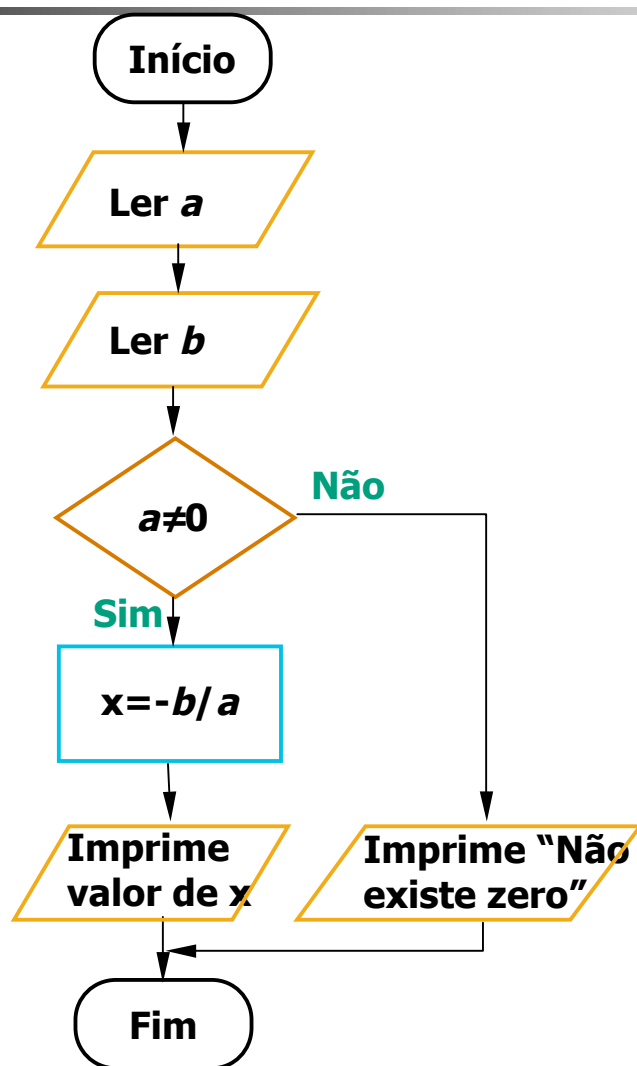
Código natural:

cálculo do zero da equação $ax+b=0$

1. *Início de programa*
2. *ler a, b*
3. *se a é diferente de 0 então*
 calcula o valor de x ($ax+b=0$)
 imprimir valor de x
senão
 imprimir "Não há zero"
4. *Fim de programa*

Fluxograma:

cálculo do zero da equação $ax+b=0$





Pseudo-código: cálculo do zero da equação $ax+b=0$

1. Início de programa
2. ler a, b
3. se $a \neq 0$ então
 - $x = -b/a$
 - imprimir valor do zero xsenão
 - imprimir "Não há zero"fim de se
4. Fim de programa



Código C:

cálculo do zero da equação $ax+b=0$

```
#include <stdio.h>

main()
{
    float a, b;
    printf("Entre com os coeficientes da equacao.\n");

    scanf("%f %f", &a, &b);

    if (a != 0)
    {
        x = -b/a;
        printf("O valor de x = %f\n", x);
    }
    else
        printf("Não existe zero");
}
```

Fluxogramas & programação visual

- Representação gráfica de um algoritmo.
- Programação visual: é a utilização de diagramas na programação.
- Descrevem o fluxo dum algoritmo através de um conjunto de figuras geométricas padronizadas ligadas por setas de fluxo.



início e fim de fluxograma



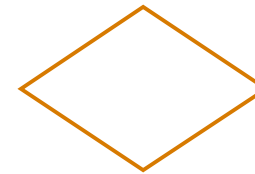
entrada e saída de dados



conector na mesma página



conector para outra página



teste e decisão



outras acções/sinistrações

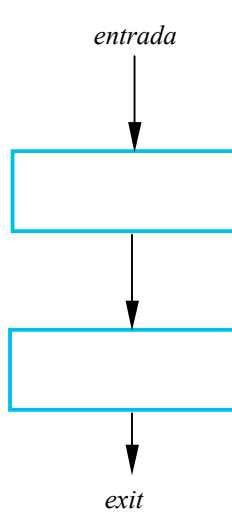


inicialização
teste e actualização

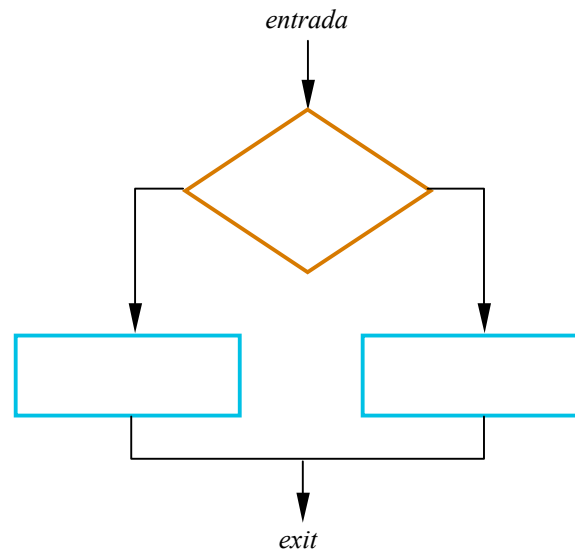
Estruturas lógicas de programação:

estruturas de controlo

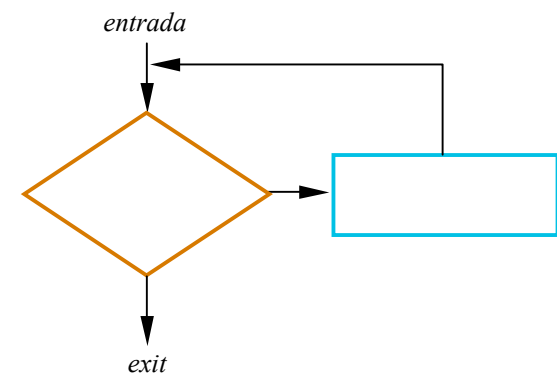
- Uma **estrutura** (de controlo) é a unidade básica da lógica de programação.
- Em meados da década de 60, alguns matemáticos provaram que qualquer programa podia ser construído através da combinação de 3 estruturas básicas: **sequência**, **selecção** e **repetição**.



SEQUÊNCIA



SELECÇÃO

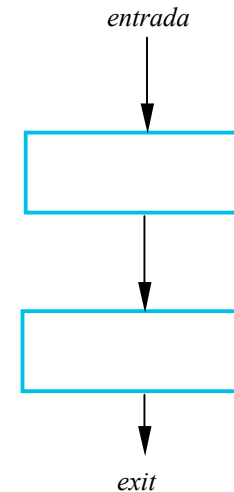


REPETIÇÃO

{...}

Sequência

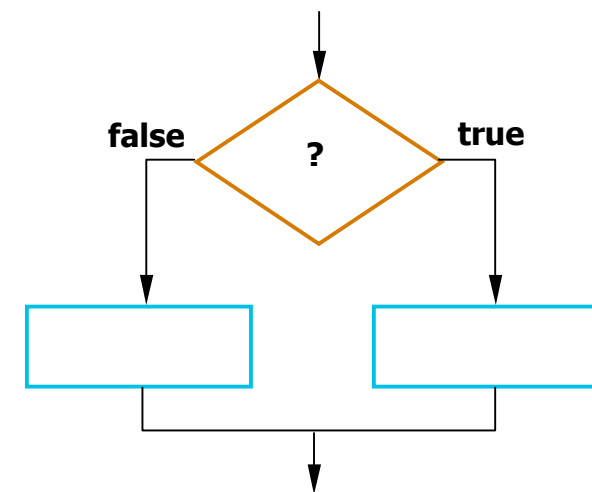
- Numa sequência é processado um conjunto de acções (ou instruções) em série.
- Não há qualquer possibilidade de alterar a ordem de processamento das acções, i.e. após processar a 1ª acção processa-se a 2ª, depois da 2ª processa-se a 3ª, e assim por diante até processar a última acção.
- Em C, uma sequência é um bloco de instruções que começa com { e termina com }



fluxograma duma sequência

Seleccção com 2 vias

- Uma estrutura de **seleccção** é também designada por estrutura de **decisão**.
- Neste caso, o fluxo de processamento segue por 1 das 2 vias, dependendo do valor lógico (verdadeiro ou falso) da expressão avaliada no início da estrutura.
- Se o fluxo de processamento só passa por 1 via, então só uma das acções é realizada ou processada.
- Em C, uma estrutura de seleccção com 2 vias é a instrução **if-else**.



fluxograma
duma seleccção de 2 vias

Problema: Calcular o maior de dois números inteiros x e y.

Exemplo em C: if-else

```
#include <stdio.h>

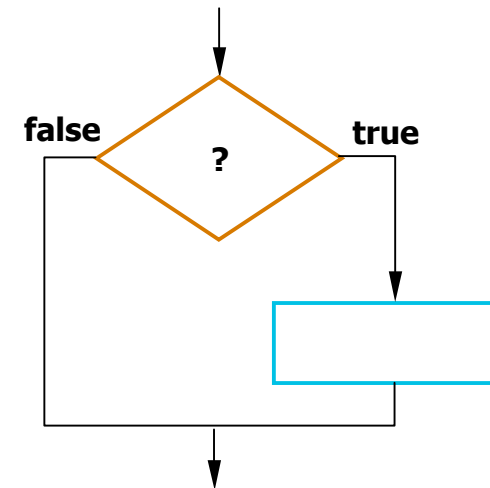
void main()
{
    int x, y, maior;

    scanf("%d%d\n", &x, &y);
    if (x > y)
        maior = x;
    else
        maior = y;
    printf("O maior dos dois inteiros = %d\n", maior);
}
```

if

Seleccção com 1 via

- Neste caso, se a expressão lógica tiver resultado **false**, nenhuma acção é processada dentro da estrutura de seleccção.
- Só é processada uma acção dentro da estrutura de seleccção se a expressão lógica for **true**; daí, o nome de seleccção com 1 via.
- Em C, uma estrutura de seleccção com 1 via é a instrução **if**.



fluxograma
duma seleccção de 1 via



Exemplo em C: if

Problema: Ler e escrever uma nota entre 0 e 20.0 valores. Caso a nota esteja no intervalo [9.0,9.5[, ela deve ser rectificada para 9.5.

```
#include<stdio.h>

void main()
{
    int nota;

    printf("Introduza o valor da nota: ");
    scanf("%f",&nota);

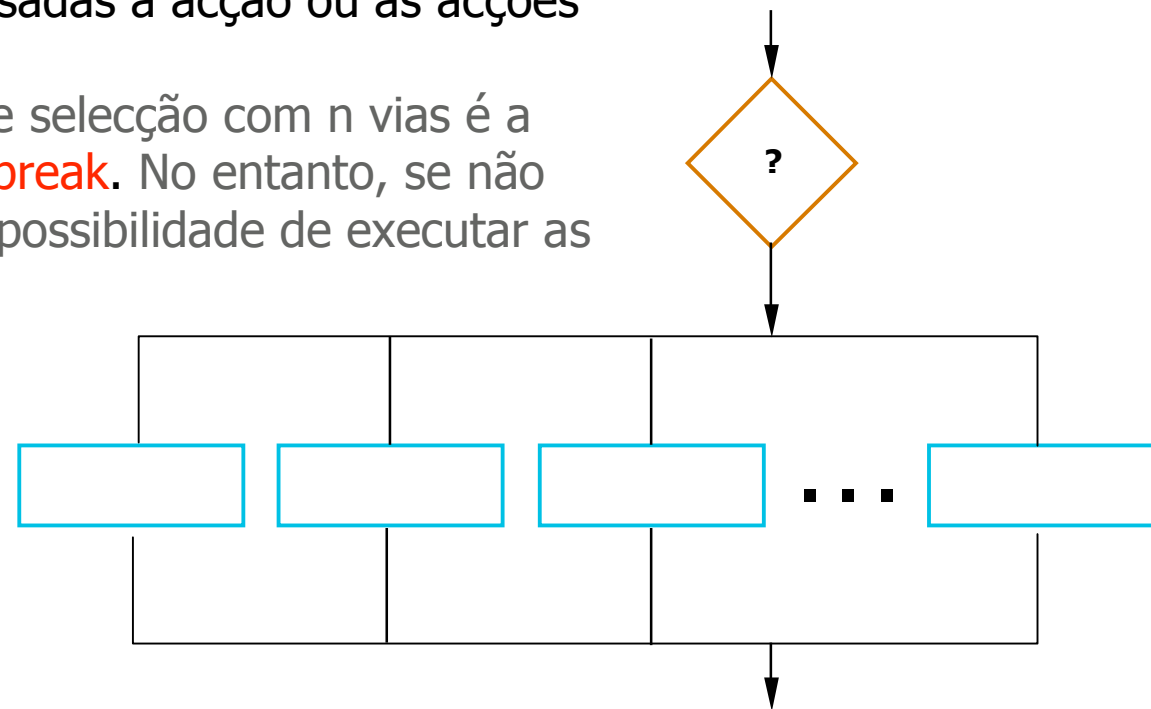
    if ( (nota >= 9.0) && (nota < 9.5) )
        nota = 9.5;

    printf("A nota = %f\n", nota);
}
```

Seleccção c/ n-vias

- Neste caso, a decisão não é feita com base numa expressão lógica porque há mais do que 2 resultados possíveis.
- Também só são processadas a acção ou as acções encontradas numa via.
- Em C, uma estrutura de selecção com n vias é a instrução **switch** com **break**. No entanto, se não usarmos o **break**, há a possibilidade de executar as acções de várias vias.

fluxograma
duma selecção de n vias





Exemplo em C: switch

```
#include <stdio.h>

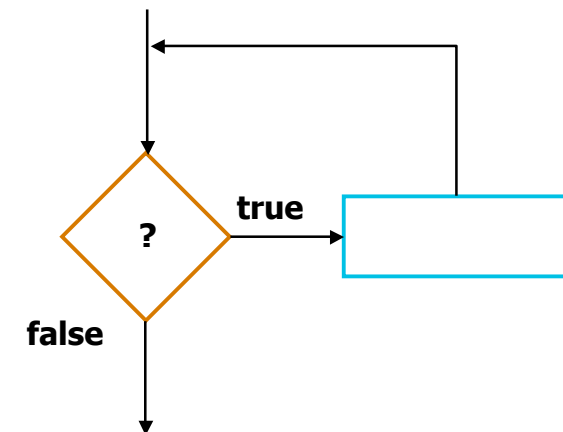
void main()
{
    int count;

    scanf ("%d", &count);

    switch count
    {
        case 5: printf ("5\n"); break;
        case 4: printf ("4\n"); break;
        case 3: printf ("3\n"); break;
        case 2: printf ("2\n"); break;
        case 1: printf ("1\n");
    }
}
```

Repetição c/ teste à cabeça

- Neste caso, também há a necessidade de tomar uma decisão com base no valor lógico duma expressão.
- No entanto, a mesma acção será executada repetidamente enquanto o resultado da expressão lógica se mantiver verdadeiro (true).
- O teste (da expressão lógica) precede a acção. Diz-se, por isso, que o *teste é à cabeça*.
- O teste é importante porque funciona como uma condição de paragem (a false) dos ciclos or repetições.
- Em C, uma estrutura de repetição deste tipo é a instrução **while**.



fluxograma
duma repetição c/ teste à cabeça



Exemplo em C: **while**

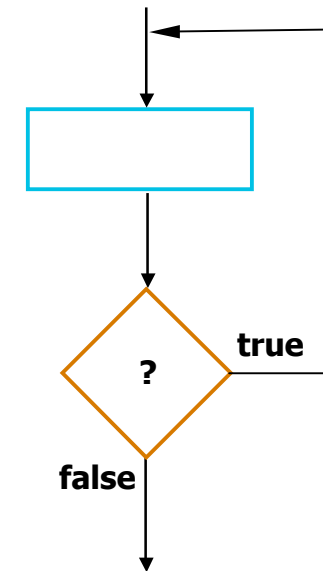
```
#include <stdio.h>

void main()
{
    int sum, k=1;

    sum = 0; // inicializacao da soma
    while (k <=100)
    {
        sum = sum + k;
        k = k + 1; //actualiza a variavel do ciclo
    };
    printf("A soma =%i\n", sum);
}
```

Repetição c/ teste à cauda

- Esta estrutura de repetição é em tudo idêntica à anterior. A diferença é que o teste é feito após o processamento da acção
- O teste (da expressão lógica) sucede a acção. Diz-se, por isso, que o *teste é à cauda*.
- Em C, uma estrutura de repetição deste tipo é a instrução **do-while**.



fluxograma
duma repetição c/ teste à cauda



Exemplo: do - while

```
#include <stdio.h>

main()
{
    int sum, k=1;

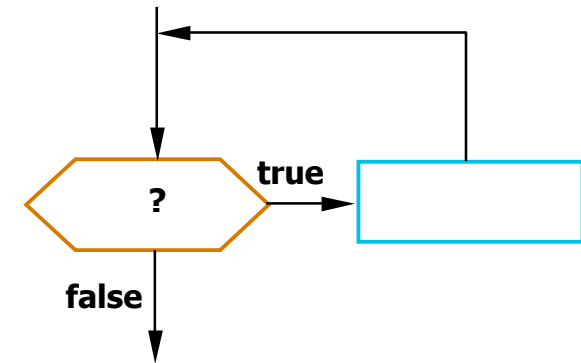
    sum = 0;           // inicializacao da variavel do ciclo
    do
    {
        sum = sum + k;
        k = k + 1;     //actualiza a variavel do ciclo
    }
    while (k<=100);
    printf("A soma =%i\n", sum);
}
```

Repetição

for

c/ número pré-definido de ciclos

- Esta estrutura de repetição é em tudo idêntica às anteriores.
- O teste é feito à cabeça.
- A diferença é que é logo à partida especificado o número de ciclos (ou iterações) que serão efectuados, i.e. o número de vezes que a acção será processada.
- Em C, uma estrutura de repetição deste tipo é a instrução **for**.



fluxograma
duma repetição c/ pré-definido de ciclos

Exemplo em C: for

```
#include <stdio.h>

void main()
{
    int        i;           // variavel do ciclo
    int        sum;

    sum = 0;               // inicializacao da soma
    for (i=1; i <=100; i++)
        sum = sum + i;
    printf("A soma = %i\n",sum);
}
```

inicialização

teste de paragem

actualização



Programação estruturada

1. Correspondência entre fluxograma e programa;
2. Uso das 3 estruturas fundamentais de controlo:
Sequência-Seleccção-Repetição;
3. As estruturas usadas devem ter um início e um final;
4. Programa escrito com indentação (realce), espaços em branco e comentários para facilitar a leitura do mesmo;
5. Eliminação das transferências incondicionais (os gotos do Fortran);
6. Desenho descendente e segmentação em módulos (ou funções);
7. Construção de módulos (ou funções) de tamanho adequado;
8. Declaração do domínio (scope) de acção das variáveis locais (dentro das funções) e globais (o programa inteiro);
9. Documentação do programa.