Funções

Programação I
2021.2022

Teresa Gonçalves
tcg@uevora.pt

Departamento de Informática, ECT-UÉ

Sumário

Controlo de fluxo

Funções

Definição e utilização

Porquê usar funções?

Como programar?

Controlo de fluxo

Controlo de fluxo

Fluxo normal

As instruções são executadas de forma sequencial

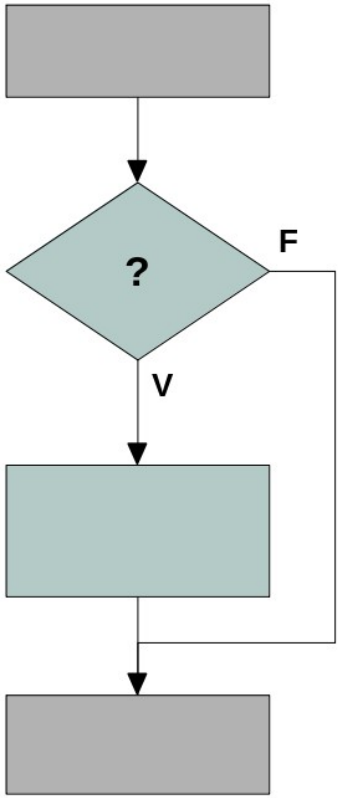
Estruturas de controlo

Permitem transferir o controlo para outras instruções

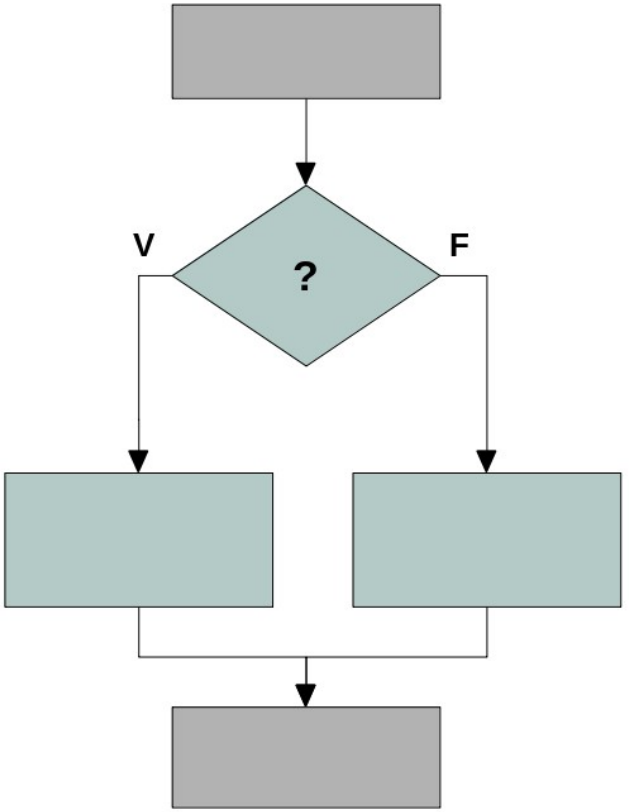
Definem diferentes sequências possíveis

Condicional

if

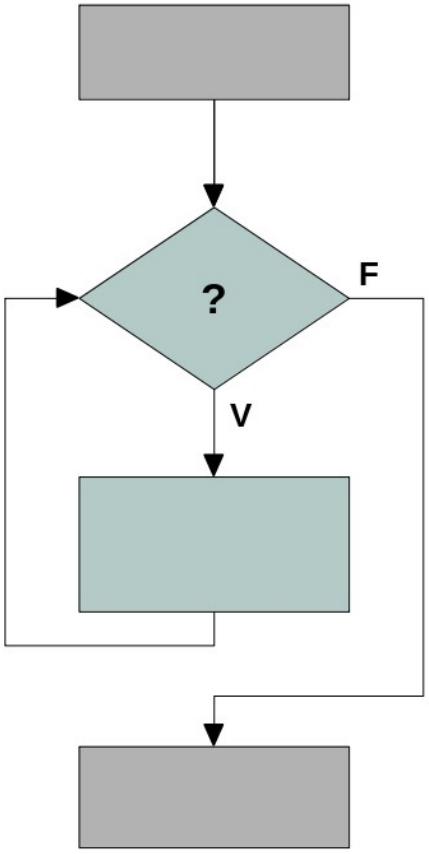


if-else

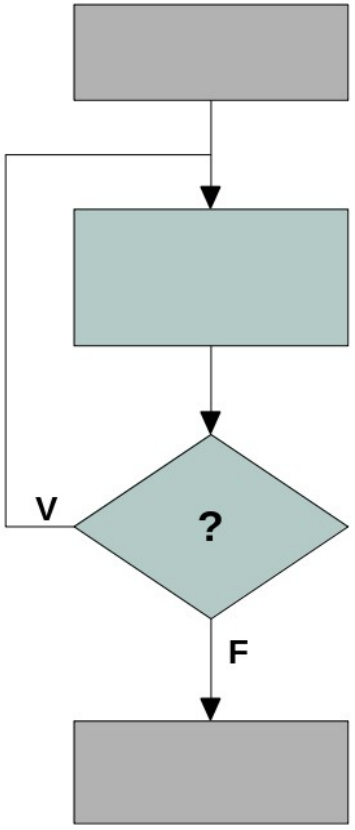


Iteração

while



do-while



Características

Condicional

- Pode não existir um else
- Podem existir inúmeros else if
- Apenas é executado um dos ramos
- Apenas as instruções referentes à 1ª condição verdadeira são executadas

Iteração

- O valor das variáveis da condição devem ser alteradas no corpo do `while`
- Nem sempre é fácil verificar a convergência

Erros frequentes

Condicional

Condições não consideram todos os valores válidos ou consideram alguns inválidos

Iteração

Variáveis não inicializadas

Valor para terminar ciclo é valor válido

Ciclos infinitos

Exercício – onde está o erro?

```
if( idade>=65 )
  printf( “Idade maior que 65.”);
```

```
int x, total;
x=1;
while( x<=10 ){
    total = total+x;
    x=x+1;
}
```

```
while( y>0 ){
    printf( “%d\n”, y );
    y=y+1;
}
```

Funções

O que temos?

Variáveis

Guardar dados

Operadores

Manipular dados

Estruturas de controlo

Definir fluxos alternativos do programa

Como permitir a reutilização do código?

Função

É uma sequência de instruções com nome que realiza uma computação

Realiza **uma tarefa** pequena e **bem definida**, implementada com toda a generalidade

Torna o desenvolvimento, teste e manutenção mais fáceis

É uma abstração

Não é necessário saber como funciona para poder ser usada

Características

A função recebe valores, trabalha sobre esses valores e retorna um valor

- Tem um nome
- Recebe argumentos
- Devolve um resultado

É executada sempre que o seu nome é invocado

Uma função

```
/* funcao que calcula o maximo divisor comum entre 2
inteiros */


int mdc( int u, int v ){
    int tmp;
    while( v!=0 ){
        tmp=u%v;
        u=v;
        v=tmp;
    }
    return u;
}
```

Ao utilizador da função apenas interesssa saber **o que a função faz**, e não como o faz...

Definição e utilização

Definição de uma função

Especifica

- o nome
- os parâmetros
- a sequência de instruções a executar

Possui

- Cabeçalho: nome, parâmetros
- Corpo: instruções a executar

Exemplo

```
int print_disciplinas(int d1, int d2)
{
  printf("%d\n", d1);
  printf( "%d\n", s2);
  return 2;
}
```

cabeçalho

corpo

Utilização (chamada da função)

A chamada da função provoca um desvio no fluxo de execução

Processo

Salta para o corpo da função

Executa as instruções lá existentes

Regressa, retomando o ponto onde tinha ficado

Exemplo

```
int n, dobro;
...
n = print_disciplinas(12, 15);
dobro = 2*n;
...
```

Mais informação

A definição cria a função

Uma função tem de ser definida antes de ser chamada

As instruções só são executadas quando a função é chamada

Ao analisar um programa devemos seguir o fluxo de execução

Resultado da função

Instrução `return`

Resultado indicado à direita da instrução

O programa continua na instrução seguinte à invocação da função

Parâmetros e Argumentos

Parâmetro

Nome utilizado na função para referir o valor passado como argumento

Argumento

Valor fornecido a uma função aquando da sua invocação

Exemplo

```
int e_par(int n){
   int res;
   if( n%2 == 0 )
     res=1;
   else
     res=0;
   return res;
}
```

n é parâmetro

```
int x;
int par;
...
scanf("%d", &x);
par = e_par(x);
if( par )
  print( "É par!" );
```

x é argumento

Visibilidade

Variáveis e parâmetros são locais

São apenas visíveis na função onde foram definidos

Exemplo

```
int soma_dois_valores( int a, int b ){
    int soma;
    soma=a+b;
    return soma;
}
...
printf("%d\n", soma);
```

Variável não declarada...
... Erro!

Visibilidade

Como as variáveis são locais

Podem existir funções diferentes com variáveis com o mesmo nome...

... no entanto, são entidades diferentes!

Utilização de funções

O argumento pode ser uma expressão

É feita a sua avaliação antes da execução do código da função

A função pode chamar-se a ela própria

Função recursiva

Composição de funções

Operação matemética sobre funções

```
f o g (x) = f (g(x))
```

As linguagens de programação usam esta operação

```
int y;
y=2;
x = sqr(sqr(sqr(y)));
```

Exemplo

```
f( g(1+2,h(3)), 4 );
```

Avalia g(1+2, h(3))

- Avaliar 1+2
- Avaliar h(3)
- Executar a função h com argumentos: 3
- Executar a função g com argumentos: 3 e resultado de h(3)

Executar a função f com argumentos: resultado de g(...) e 4

Porquê usar funções?

Porquê usar funções?

Torna o programa mais legível

Torna mais fácil fazer debugging

Permite analisar cada uma das partes em separado

Pode tornar o programa mais pequeno ao eliminar código repetido