Programação Imperativa (CC)

Departamento de Ciência de Computadores

—-/—-

Armando Matos

Índice

Introdução 3	O "`main`" com parametros! 150
Uso de computadores (Linux) 4	Ficheiros 156
Algoritmos 11	Inteiros de precisão arbitrária 179
Instruções do C 68	Matrizes 189
Caracteres e "strings" 73	Números pseudo-aleatórios 208
O pré-processador 92	Resolução de alguns problemas 210
Vectores 102	
"Strings" 137	

Nota. Prevê-se que estes slides sejam alterados durante o semestre por forma a refletir o programa do corrente ano lectivo.
Em particular, deverá ser incluída uma parte sobre estruturas (`struct`) e suas aplicações.

Uso dos computadores

Algumas notas

Uso dos computadores

Muitos programas...

- Terminal e "shell"
- Editor de texto
- Compilador da linguagem C
- Os programas que nós fazemos
  - Como?

Terminal e "shell"

- Exemplos de comandos `ls`, `cd`, `mkdir`, `cp`, `mv`, `cat`, `more`, `rm`, `rmdir`.
- Estrutura em árvore dos ficheiros
- Redirecção da entrada padrão e da saída padrão

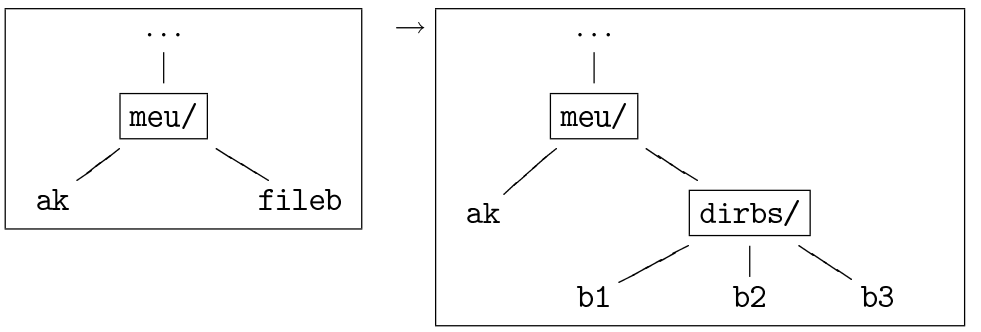
Um editor de texto: `emacs`

- Abertura e fecho de ficheiros.
- Movimento do cursor
- Pesquisa e substituição
- `emacs`: um editor poderoso

Compilador de C: `gcc`

- O que faz?
- Exemplo
- Programas escritos em C e sistemas Unix/Linux

⇒AGORA: Exercício⇐



Problema. Apresente uma sequência de comandos que efectue a transformação indicada. Suponha que inicialmente está a trabalhar no directório `meu`. Os ficheiros `b1`, `b2` e `b3` são todos idênticos ao ficheiro (inicial) `fileb`.

Algoritmos

Algoritmos: Representação: linguagem informal I

Linguagem informal

```
//--- Que escreve?
//--- variáveis usadas: p,n,i (inteiras)
leia n;
coloque 1 em p;
para i=1,2,...,n
  coloque p*i em p;
escreva(p);
```

Seguir o algoritmo para n=4

Algoritmos: Representação:
linguagem informal II

Linguagem informal

```
leia n;
p=1;
para i=1,2,...,n{
  p=p*i;
}
escreva(p);
```


...instruções ...

- Atribuição: o valor de uma expressão é calculado; esse valor é colocado numa posição de memória (identificada por uma variável)
- Instrução condicional
- Instrução de ciclo
- Comentário

Linguagem C – I

C:

```
/*- Programa que calcula o factorial -*/
main(){
  int n,p,i;
  scanf("%d",&n);
  p=1;
  for(i=1;i<=n;i++){
    p=p*i;
  }
 printf("Factorial de %d = %d\n",n,p);
}
```


⇒AGORA: Exercício⇐

O que imprime o seguinte programa (valores finais de `n` e de `p`)?

```
//-- Programa que calcula o factorial
main(){
  int n,p,i;
  printf("Valor de n?");
  scanf("%d",&n);
  p=1;
  while(n>=2){
    p=p*n;
    n--;     // mesmo que n=n-1 ou n-=1;
  }
 printf("Factorial de %d = %d\n",n,p);
}
```


C – III

```
//-- função que calcula o factorial
int factorial(int m){
  if(m==0)
    return(1);
  return(m*factorial(m-1));
}
main(){
  int n;
  printf("Valor de n? ");
  scanf("%d",&n);
  printf("Fact. de %d = %d\n",n,factorial(n));
}
```

Seguir o programa!

Linguagem Prolog – programação lógica

```
%--predicado fact(N,F): f é o factorial de n
fact(0,s(0)).
fact(s(N),F) :- fact(N,Fn), produto(s(N),Fn,F).

produto(0,A,0).
produto(s(A),B,C) :- produto(A,B,C1), soma(C1,B,C).

soma(0,A,A).
soma(s(A),B,s(C)) :- soma(A,B,C).
```


Linguagem Prolog

Utilizando:

```
   ?- soma(A,B,s(0)).
A = 0,
B = s(0) ? ;
A = s(0),
B = 0 ? ;

   ?- fact(s(s(s(0))),A).
A = s(s(s(s(s(0))))) ? ;
no
```

Linguagem Haskell – programação funcional

```
fact :: Num a => a->a
fact 0 = 1
fact n = n* fact (n-1)
```

Outro exemplo

```
map :: (a->b) -> [a] -> [b]
map op [] = []
map op (a:r) = (op a):(map op r)
```

Exemplo: `map fact [1,2,3,4]  =>  [1,2,6,24]`

Algumas instruções do C

Algumas instruções do C – resumo

- `Var = Exp;`
- `return Exp;`
- `break;`
- `if(Cond) Instr`
- `if(Cond) Inst1 else Inst2`
- `while(Cond) Instr`
- `do Instr while(Cond);`
- `for(Exp1; Exp2; Exp3) Instr`
- `switch(Expr){ case Exp1: Insts1... default: Instsn}`

Palavras reservadas do C: `if`, `else`, `while`, `do`, `for`, `switch`, `case`.

Atribuição em C

Forma (simplificada...)

```
<variável> = <expressão>;
```

Na realidade "as atribuições fazem parte das expressões"

Atribuição em C

```
int factorial(int m){
  if(m==0)
    return(1);  // (o "else" não é preciso)
  return(m*factorial(m-1));
}
main(){
  int x,xix=2,varia38x=8,a;
  double u=1.1;
  char c;
  x=20; xix = xix*38;  varia38x *= 2;
  a=factorial(3)*(4+7);
  a=a+'c';       // conversão de tipos...
  u=u*2.1;  c='c'+2;
```


Atribuição em C

...continuação ...

```
  printf("x=%d, xix=%d, varia38x=%d,
          a=%d u=%lf, c=%c\n",
         x,xix,varia38x,a,u,c);
}
-------------------------------------------------
x=20, xix=76, varia38x=16,
a=165 u=2.310000, c=e
```

Trocando os conteúdos de 2 variáveis

```
int a,b;
a=8;
b=2;
a=b;
b=a;
```

O que imprime?

$\Rightarrow$ Uma solução correcta

Nota importante

O que é um programa em C?

(Essencialmente) é uma sequência de definições de funções (tendo uma delas o nome de "`main`") – e de variáveis.

Já vimos exemplos!

A Matemática e as linguagens imperativas...

Em Matemática e em C o significado do sinal "=" é muito diferente:

Exemplos

- $x = y + z$: Em Matemática e em C
- $x = x + 2$: Em Matemática (é sempre falso) e em C (uma atribuição)

Como é o teste de igualdade em C? Resposta: "==". Exemplo:

```
if(x==y+1)
  x=x+2;
```


Exemplo de instrução "`if...else`"

```
if(x>y)
  max=x;
else
  max=y;
```

A pré-condição mais forte correspondente à pós-condição

$$(x > y \wedge \texttt{max} = x) \vee (x \leq y \wedge \texttt{max} = y)$$

é "`V`" (verdade), isto é, esta instrução coloca em m o maior dos valores x e y (independentemente dos valores iniciais).

A instrução "`if`" e os testes

Operadores relacionais: funções $\mathbb{N} \times \mathbb{N} \to \{F, V\}$. São

`<` `<=` `>=` `>` `==` `!=`

Conectivos lógicos

`!` `&&` `||`

Instrução "`if`"

```
if(teste) instrução;
```

ou

```
if(teste) {instrução;...;instrução};
```


A instrução "`if`" e os testes

Instrução "`if...else`"

```
if(teste) instrução else instrução
```

Em vez de "`instrução`" pode sempre colocar-se um grupo de instruções entre chavetas:

```
{instrução;...;instrução}
```

Exemplos de instruções "`if`"

```
if(!(x==0) && y>0)
  x=x/y;  // divisão inteira
else
  print("Erro nos dados!\n");


if(x%2==0)
  print("x é par\n");
else
  print("x é ímpar\n");
```


Ciclos – instrução `while`

Relembra-se que em vez de "`instrução`" pode sempre colocar-se um grupo de instruções entre chavetas

```
while(teste) instrução
```

Semântica.

Qual o valor impresso por (exercício)

```
scanf("%d",&x);
y=0;
while(x!=0){
  y+=x;
  x--;
}
printf("%d\n",y);
```

Ciclos – instrução `while` – exemplo

Problema. Escreva um programa que lê sucessivamente inteiros até ser lido um inteiro negativo (que serve como indicação de terminação; esse programa deve imprimir o maior dos inteiros lidos.

Ciclos – instrução `while` – exemplo

Uma solução:

```
/* x é o inteiro lido; max o maior dos
   inteiros já lidos                    */
main(){
  int x, max;
  scanf("%d",&x);
  max=x;
  while(x>=0){
    scanf("%d",&x);
    if(x>max) max=x;
  }
  printf("%d\n",max);
}
```

processamento de dados – `while`

Quando o programa termina com um valor especial lido...

```
.......
<ler n>
.......
while(...n...){
  <processa n>
}
<finalmentes>
```


Instrução `while` – outro exemplo

Problema: ler caracteres (instrução `getchar()`) até encontrar "`.`" e imprimir o número de letras lidas.

Primeiro uma função que diz se `c` é uma letra (retorna 1) ou não (retorna 0).

Instrução `while` – outro exemplo

```
int letra(int c){ //-- c é inteiro
  if(c>='a'&&c<='z' || c>='A'&&c<='Z')
     return(1);
  else
    return(0);
}
```

Outra alternativa:

```
int letra(int c){
  return(c>='a'&&c<='z' || c>='A'&&c<='Z');
}
```


Outro exemplo – continuação

Recordar: ler caracteres (instrução `getchar()`) até encontrar "`.`" e imprimir o número de letras lidas.

```
main(){
  int c,soma;
  soma=0;
  c=getchar();
  while(c!='.'){
    if(letra(c))
      soma=soma+1;
    c=getchar();
  }
  printf("Tem %d letras\n",soma);
}
```

Ciclos – outra solução `do...while(...)`

Problema: Determinar o maior valor de uma sequência de valores não negativos. O primeiro valor negativo termina a sequência.

Ciclos – outra solução `do...while(...)`

```
main(){
  int x, max=-1;
  do{
    scanf("%d",&x);
    if(x<0)
      break;
    if(x>max)
      max=x;
  }
  while(1);    //-- má programação...
  printf("%d\n",max);
}
```

Ciclos – outra solução `do...while(...)`

Temas:

teste 1

instrução `do...while();`

instrução `break;`

Ciclos – outra solução: `for`

```
main(){
  int x, max=-1;
  for(max=-1;scanf("%d",&x),x>=0;)
    if(x>max)
      max=x;
  printf("%d\n",max);
}
```

Ciclos – outra solução: `for`

Forma:

```
for(exp1;exp2;exp3)
  instrução
```

```
exp1,exp2,exp3
  exp1: executada no início (1 vez)
  exp2: teste de fim
  exp3: executada sempre após "instrução''
```


Ciclos – for

Novidades

- Uma expressão pode ter partes separadas por vírgula. O valor é o da última expressão.
- Uma expressão pode conter atribuições, ex:`x=1+(y=7);`

Exemplo frequente da utilização do "`for`":

```
for(i=0;i<n;i++){
  ...
}
```

Processamento de dados: `for`

```
.......
scanf("%d",&n);
.......
for(i=0;i<n;i++){
  //-- n vezes!
  <processa n>
}
<finalmentes>
```


Instrução `switch`

Permite a selecção de uma das alternativas. Forma

```
switch(Expr){
  case Exp1: Insts1
  case Exp2: Insts2
  ...
  default: Instsd
}
```

Instrução `switch`

Funcionamento: A **`Expr`** é calculada. Se for igual a **`Exp1`**, o grupo de instruções **`Insts1`** é executado,..., etc.
Se não for igual a nenhuma das expressões, é executado **`Instsd`**.

- As expressões erb+Exp1+, erb+Exp2+,...devem ser constantes.
- Cada grupo de instruções **`Insts1`**, **`Insts2`** é normalmente terminado com a instrução **`break`**. Se não for a execução continua com as instruções à frente.
- A parte de **`default`** é opcional.

Instrução `switch` - continuação

```
switch(c){
case 'a':
  calcula();
  break;
case 'q':
  termina();
  break;
default:
  printf("Erro no comando\n");
}
```

Algumas instruções do C – resumo

- `Var = Exp;`
- `return Exp;`
- `break;`
- `if(Cond) Instr`
- `if(Cond) Inst1 else Inst2`
- `while(Cond) Instr`
- `do Instr while(Cond);`
- `for(Exp1; Exp2; Exp3) Instr`
- `switch(Expr){ case Exp1: Insts1... default: Instsd}`

Palavras reservadas do C: `if`, `else`, `while`, `do`, `for`, `switch`, `case`.

⇒AGORA: Exercício⇐

Exercício 1: Escreva um programa que leia o inteiro n e n inteiros e imprima a soma dos n inteiros.
Exemplo: Dados: `3 4 -1 2`, Resultado: `5`.

Exercício 2: Escreva um programa que leia o inteiro n e imprima o resultado de $1 + 2 + \cdots + n$.
Exemplo: Dados: `4`, Resultado: `10`.
Nota: pode usar um ciclo ou directamente uma expressão dessa soma.

O segundo maior...

Exercício Escreva um programa que leia o inteiro n e n inteiros positivos e imprima o segundo maior desses n inteiros.
Exemplo: Dados: `8 3 4 3 5 5 4 1 2`,
Resultado: `4`.

Uma solução

Variáveis:

```
n:    número de elementos a ler
x:    último elemento lido
max1: maior elemento já lido
max2: segundo maior elemento já lido
```


Uma solução – exercício 3

Estratégia:

```
  ler n; ler o primeiro elemento -> max1
  fazer n-1 vezes:
    ler x
    em função de x, corrigir (eventualmente) max1 e max2
  imprimir max2
}
```

Uma solução – exercício 3

```
main(){
  int x, max1, max2=-1,i,n;
  // x: lido, max1: maior, max2: 2o maior
  scanf("%d",&n); scanf("%d",&max1);
  for(i=2;i<=n;i++){
    scanf("%d",&x);
    // Ver se e' o maior de todos:
1.  if(x>max1){max2=max1;max1=x;}
    else  // Ver se e' so' maior que max2:
2.  if(x<max1 && x>max2) max2=x;
    // Nos outros casos nada se faz;
  }
  printf("%d\n",max2);
}
```


Uma solução – segundo maior

Questões

- Se trocarmos as instruções 1. e 2.?
- Se retirarmos "`x<max1`"?
- Se `n` = 1?
- Se `n` = 2, valores `4 4`?

– Exercício - combinações I –

Escreva um programa que leia sucessivamente pares de inteiros não negativos `m` e `n` e escreva $\binom{m}{n}$, o número de combinações de `m` objectos tomados `n` a `n`. O programa termina quando for dado um valor negativo para `m`.

O cálculo de $\binom{m}{n}$ deve ser efectuado por uma função
`int comb(int m,int n);`.

– Exercício - combinações I –

Nota. Esse número de combinações está na linha `m`, coluna `n` do triânculo de Pascal.

```
    0  1  2  3  4  5
  ------------------
0 | 1  0  0  0  0  0
1 | 1  1  0  0  0  0
2 | 1  2  1  0  0  0
3 | 1  3  3  1  0  0
4 | 1  4  6  4  1  0
5 | 1  5 10 10  5  1
```

– Combinações II –

Primeiro vamos supor que a função `comb` já está feita e vamos escrever a função `main()`.

– Combinações II –

```
main(){
  int m,n;
  printf("m? ");  scanf("%d",&m);
  printf("n? ");  scanf("%d",&n);
  while(m>=0){
    printf("(%d,%d) = %d\n",m,n,comb(m,n));
    printf("m? ");  scanf("%d",&m);
    printf("n? ");  scanf("%d",&n);
  }
}
```

– Combinações III –

comb, versão I. Usa a conhecida igualdade

$$\binom{m}{n} = \binom{m-1}{n} + \binom{m-1}{n-1}$$

```
//-- Combinacoes (a,b)
int comb(int a, int b){
  if(b==0)
    return(1);
  if(a==0)
    return(0);
  return comb(a-1,b)+comb(a-1,b-1);
}
```

Função recursiva.

– Combinações IV –

comb, versão I. Usa a conhecida igualdade

$$\binom{m}{n} = \frac{m!}{n!(m-n)!}$$

– Combinações IV –

```
//-- Factorial de n
int fact(int n){
  int i,p=1;
  for(i=2;i<=n;i++)
    p=p*i;
  return(p);
}

//-- Combinacoes (a,b)
int comb(int a, int b){
  return(fact(a)/(fact(b)*fact(a-b)));
}
```

(20,10)=11 Porquê???

– Combinações V –

Usa a igualdade

$$\binom{m}{n} = \frac{m \times (m-1) \times \ldots \times (m-n+1)}{n!}$$

e aumenta a eficiência usando $\binom{a}{b} = \binom{a}{a-b}$.

```
//-- Combinacoes (a,b)
int comb(int a,int b){
  int i,num=1,den=1;
  if(a-b<b) b=a-b;
  for(i=a;i>=a-b+1;i--) num*=i;
  for(i=2;i<=b;i++) den*=i;
  return(num/den);
}
```

– Combinações VI –

Notas

- Nomes dos parâmentros
- Parâmentros: passados por valor, vars. locais
- `b = b<a-b?b:a-b;`
- Espaços no texto do programa...

Números em vírgula flutuante – algumas notas

Números em vírgula flutuante – algumas notas

Notas

- Usar preferencialmente o tipo `double`. Nas instruções de leitura e de escrita usar o formato `%lf` (número "floating point" longo).
- Se usar funções matemáticas da biblioteca, colocar no início do programa "`#include <math.h>`" (ver por exemplo `man sin`).
- Compilar com "`gcc -lm teste.c`"; `-lm` "chama" a biblioteca de funções matemáticas (`teste.c` é o programa a compilar).

Números em vírgula flutuante – exemplo I

Ler um número e imprimir o seu seno.

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

#define PI 3.14159264

main(){
  double u,ur;
  scanf("%lf",&u);
  ur=u*PI/180.0;
  printf("%lf graus, seno=%lf\n",u,sin(ur));
}
```

Números em vírgula flutuante – exemplo II

Tabela de cosenos de ângulos 0, 5,..., 90 (graus). Pretende-se:

```
ANGULO   COSENO
--------------
     0  0.00000
     5  0.08716
    10  0.17365
   ...  .......
    85  0.99619
    90  1.00000
```


Números em vírgula flutuante – exemplo II (cont)

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

#define PI 3.14159264

main(){
  int ang; double ar;
  printf("ANGULO   COSENO\n");
  printf("--------------\n");
  for(ang=0;ang<=90;ang+=5){
    ar=ang*PI/180.0;
    printf("%6d %8.5lf\n",ang,sin(ar));
  }
}
```

Leitura e escrita de caracteres

Já sabemos que podemos ler caracteres com uma instrução do tipo `scanf("%c",&c)}`.

Mas uma forma mais simples e primitiva é com a função `getchar()`. Por exemplo com

$$\texttt{c} = \texttt{getchar}();$$

o código do caracter lido é colocado no inteiro `c`.

Note-se que `getchar()` dá um resultado inteiro. Isso permite reconhecer o fim de um ficheiro – quando o valor retornado é `-1` (também representado por `EOF`).

Leitura e escrita de caracteres

Com o comando `a.out < fich1 > fich2`, o que faz o seguinte programa?

```
main(){
  int ch;
  ch=getchar();
  while(ch!=EOF){
    ch = (ch+1)%256;
    putchar(ch);
    ch=getchar();
  }
}
```

Experimente!

Leitura e escrita de caracteres

Forma mais compacta (relembremos: as expressões podem incluir atribuições):

```
main(){
  int ch;
  while((ch=getchar())!=EOF){
    ch = (ch+1)%256;
    putchar(ch);
  }
}
```

Experimente!

Ficheiros - veremos mais tarde...

Nos sistemas Unix há sempre 3 ficheiros "stream" abertos (o utilizador pode abrir outros – veremos como):

- `stdin`, a entrada padrão ("standard input"), normalmente o `teclado`.
- `stdout`, a saída padrão ("standard output"), normalmente o `monitor`.
- `stderr`, a saída de erro padrão ("standard error"), normalmente o `monitor`.

Sinónimos

$$\texttt{car} = \texttt{getchar}(); \quad \equiv \quad \texttt{car} = \texttt{getc}(\texttt{stdin});$$
$$\texttt{putchar}(\texttt{car}); \quad \equiv \quad \texttt{putc}(\texttt{car}, \texttt{stdin});$$

Processando caracteres...

```
main(){
  int c=getchar();
  while(c!=EOF){
    ...
    c=getchar();
  }
  ...
}
```

Processando caracteres...

```
main(){
  int c;
  while((c=getchar())!=EOF){
    ...
  }
  ...
}
```


Exercícios com caracteres

Exercício: Escreva um programa que leia os caracteres de um ficheiro (redireccionando a entrada padrão) e imprima em linhas separadas: o número total de caracteres, o número de letras e o número de dígitos.

Uma solução – número de caracteres, letras e dígitos

```
#include <stdio.h>
int letra(int c){
  return((c>='a'  && c<='z') ||
         (c>='A'  && c<='Z'));
}

int digito(int c){
  return(c>='0'  && c<='9');
}
```


Exercício: completar!

```
main(){
  int c,...
  while((c=getchar())!=EOF){
    .....
    if(letra(c))
      .....
    else
      if(digito(c))
        .....
  }
  printf("%d carac., %d letras, %d digitos\n",...,...,...);
}
```

Exercícios com caracteres

Exercício: Escreva um programa que imprima uma tabela dos códigos "ascii" e dos correspondentes caracteres no seguinte formato e entre os limites indicados:

```
Código    Caracter
32
33           !
34           "
.....
127
```


Uma solução – tabela ASCII

```
#include <stdio.h>

main(){
  int i;
  for(i=32;i<128;i++)
    printf("%4d   %1c\n",i,i);
}
```

Uma solução – tabela ASCII

```
Resultado:
  32
  33   !
  34   "
  ...
 125   }
 126   ~
 127
```


Uma solução – tabela ASCII mais bonita

Vamos escrever um programa que imprime uma tabela do seguinte tipo:

Uma solução – tabela ASCII mais bonita

```
      2 3 4 5 6 7 8 9 0 1
    --------------------
 30 |   ! " # $ % & ' ( )
 40 | * + , - . / 0 1 2 3
 50 | 4 5 6 7 8 9 : ; < =
 60 | > ? @ A B C D E F G
 70 | H I J K L M N O P Q
 80 | R S T U V W X Y Z [
 90 | \ ] ^ _ ` a b c d e
100 | f g h i j k l m n o
110 | p q r s t u v w x y
120 | z { | } ~
```


Uma solução – tabela ASCII – continuação

```
#include <stdio.h>

main(){
  int i;
  printf("       2 3 4 5 6 7 8 9 0 1\n");
  printf("     --------------------  ");
  for(i=32;i<128;i++){
    if(i%10==2)
      printf("\n %3d | ",(i/10)*10);
    printf("%1c ",i);
  }
}
```

Exercícios com caracteres

Exercício: Escreva um programa que leia os caracteres de um ficheiro "source" de uma página `html` e imprima os nomes dos comandos que encontrar; cada comando deve aparecer numa linha separada. Entende-se por *comando* a sequência de letras que se segue a um sinal "<".

Exercícios com caracteres

Sugestão Sugere-se começar por representar um diagrama de estados que inclua as seguintes situações (admite-se que não ocorre `<...<...`):
(1) Fora de `<...>`, (2) Já leu `<` e está a processar a sequência de letras a seguir a `<`, (3) Dentro de `<...>` mas o nome do comando já foi lido. Por exemplo:

```
Ficheiro:  Ei-lo: <IMG ALT="...
Estado:    1111111122233333...
```

```
main(){ int c,fora=1,palavra;
  while((c=getchar())!=EOF){
    if(c=='<'){fora=0; palavra=1;}
    else
      if(!fora && palavra && letra(c)) // letra do comando
        putchar(c);
      else
        if(!fora && palavra && !letra(c)){
          //-- acabou um comando
          putchar('\n'); palavra=0; if(c=='>') fora=1;
        }
        else
         if(!fora && c=='>') fora=1;
  }
}
```


O pré-processador

Pré-processador

Linhas começadas por "#".

```
#define Aa 100
```

Substitui todas as ocorrências de "`Aa`" por "`100`".

```
#define MAX 1000
#define ITERS 2*MAX

main(){
  int vec[MAX],i,k;
  for(i=0;i<MAX;i++)
    if(vec[k]>ITERS)...
}
```


Pré-processador

Bons hábitos:

1. Definir todas as constantes com `define`; não as usar directamente.
   Porquê?
   Importante: Legibilidade, menos erros, modificações mais fáceis.
2. Nomes definidos devem ser em maiúsculas.

Referência aos `define` com parâmentros, à expansão de macros.

Referência ao `#include ....`

Pré-processador – experiência...

Um programa (com erros!) em `expand.c`

```
#define MAX 10;

int x[MAX];

main(){
  MAX = 20;
  x += MAX+1;
}
```


Pré-processador – experiência...

Um programa (com erros!) em `expand.c`

```
#define maior(a,b) a>=b?a:b

int calcula(int i){
  int x=4+maior(i,5);
  return(x);
}

#include "batatas"
```

Pré-processador – experiência...

O ficheiro `batatas`

```
int k(int n){
 return(n);
}
```


...O resultado de `gcc -E expand.c`

```
# 1 "expand.c"
int x[10; ];

main(){
  10;  = 20;
  x += 10; +1;
}

int calcula(int i){
  int x=4+  i >= 5 ? i : 5  ;
  return(x);
}
```

Pré-processador – experiência...

O ficheiro `batatas`

```
# 1 "batatas" 1

int k(int n){
 return(n);
}
# 17 "expand.c" 2
```


Pré-processador

Bons hábitos:

1. Todas as constantes devem ser definidas com `defies` e não usadas directamente. Porquê?
   Importante Legibilidade, menos erros, modificações mais fáceis.
2. Nomes definidos devem ser em maiúsculas.

Referência aos `include`'s com parâmentros, à expansão de macros.

Referência ao `#include ....`

Dar nomes aos tipos

```
typedef int booleano;
```

A partir daqui `booleano` é sinónimo de `int`.

Exemplo:

```
booleano grande(int a){
  return(a>=MAX);
}
```

Forma (um pouco mais) geral

```
typedef <tipo> <nome>;
```


Vectores

Pesquisa e ordenação – introdução

Nota: usamos os termos "array" e "vector" como sinónimos.

Sobre o uso de "arrays"

```
  #include<stdio.h>
  #define MAXIMO 100
  main(){
    int i;
    int a[MAXIMO];    // indice de 0 a 99; limites nao testados!
    for(i=0;i<MAXIMO;i++) a[i]=i/2;
    a[10]=5;  i=11;  a[i+1]=i+3;
    for(i=0;i<=14;i++)    printf("a[%2d]=%2d  ",i,a[i]);
  }
/* Resultado:
a[ 0]= 0  a[ 1]= 0  a[ 2]= 1  a[ 3]= 1  a[ 4]= 2
a[ 5]= 2  a[ 6]= 3  a[ 7]= 3  a[ 8]= 4  a[ 9]= 4
a[10]= 5  a[11]= 5  a[12]=14  a[13]= 6  a[14]= 7
*/
```


Vectores e apontadores

```
main(){
  int i=10, v[10] = {1,2};
  printf("i     = %d\n",i);
  printf("v[1]  = %d\n",v[1]);
  printf("v[3]  = %d\n",v[3]);
  printf("&i    = %u\n",&i);
  printf("v     = %u\n",v);
  printf("&v[0] = %u\n",&v[0]);
  printf("&v[1] = %u\n",&v[1]);
}
-----------------------------------
i     = 10          v[1]  = 2
v[3]  = 0           &i    = 3221224152
v     = 3221224112  &v[0] = 3221224112 &v[1] = 3221224116
```

Pesquisa sequencial

Problema: Procurar se um valor dado `x` existe num vector `a[0]`,..., `a[n-1]`. A resposta pode ser

1. Um índice `i` tal que $x =$ `a[i]`.
2. `-1` se não houver nenhum `i` nessas condições.

Seja o seguinte "array", $n = 7$

i	0	1	2	3	4	5	6
$a[i]$	8	5	6	1	3	9	4

Se $x = 9$ a resposta deve ser 5 pois $a[5] = 9$.

Exercício 1 *Quais deveriam ser as respostas para os seguintes valores de x: 0, 8 e 1?* ◇

Pesquisa: um algoritmo

Ideia: Para $i = 0, 1, \ldots, n-1$ comparamos $a[i]$ com x. Se forem iguais, o resultado é i. Se nunca forem iguais, o resultado é -1.

Em linguagem C são possíveis muitas variantes, usando "`for`", "`while`", "`break`"... Por exemplo

Pesquisa...continuação

```
int pesquisa(int, int [], int);

main(){
  int a[]={8,5,6,1,3,9,4}, n=7,
      x, pos;
  scanf("%d",&x);
  pos=pesquisa(x,a,n);
  if(pos>=0)
    printf("Posicao = %2d\n",pos);
  else
    printf("Nao ocorre em a[ ]\n");
}
```


Pesquisa...continuação

```
int pesquisa(int x,int v[], int m){
  int i;
  for(i=0;i<m;i++)
    if(v[i]==x) return(i);
  return(-1);
}
```

Exercício 2 *Escreva funções de pesquisa nas seguintes condições. Teste os seus programas!*

Usando a instrução de ciclo "`while`*" sem utilizar "*`break`*".*
Uma solução (parte da função).

```
i=0;
while(i<n && a[i]!=x)
  i++;
/* Pela semantica do "&&" a comparacao
   a[n]!=x nunca se faz */
```

◇

Exercício 3 *Simplifique e melhore a eficiência do programa anterior, criando uma "sentinela", isto é, começando por colocar* `x` *em* `a[n]`*, supondo-se que esta posição está livre.*
Uma solução (parte da função).

```
i=1; a[n]=x;
while(a[i]!=x) /* termina sempre! */
  i++;
return(i<n? i: -1);
```

◇

Exercícios com "arrays"

Exercício 4 *Escreva uma função*

```
void roda(int v[],int n)
```

que "roda" um vector de uma unidade para a direita, passando o último valor para a primeira célula. Exemplo de uma transformação efectuada: | *8* *5* *6* *1* *3* | → | *3* *8* *5* *6* *1* |

◇

Exercício 5 *Escreva uma parte de um programa que troque simetricamente os n elementos de um "array", isto é,* $a[0]$ *troca com* $a[n-1]$, $a[1]$ *troca com* $a[n-2]$, *etc.*
Por exemplo,
| *8* *5* *6* *1* *3* | → | *3* *1* *6* *5* *8* | ◇

Exercícios com "arrays"

Exercício 6 *Escreva uma função*

```
void insere(int x,int v[],int n)
```

que insira um elemento x na posição i de um "array" com n elementos, deslocando previamente os elementos $a[i],\ldots,a[n-1]$ *para a direita de uma unidade (ficam* $n+1$ *elementos).*
Por exemplo, se o "array" é | *8* *5* *6* *1* *3* |, $x = 12$ *e* $i = 3$ *deve resultar* | *8* *5* *6* *12* *1* *3* | ◇

Exercícios com "arrays"

Exercício 7 *São dados 2 "arrays" a[ ] e b[ ] com respectivamente m e n elementos. Pretende-se imprimir os elementos que pertencem à intersecção dos conjuntos representados pelos 2 "arrays". Por exemplo, se os "arrays" são* | 8 | 6 | 5 | 1 | 3 | *e* | 2 | 1 | 4 | 6 |, *deve ser impresso 6 e 1.*

Sugestão: *utilizando uma função de pesquisa, procure cada um dos elementos de a[ ] em b[ ],*

```
int pesquisa(int x,int v[],int n)
```

◇

Pesquisa binária

Se o vector onde se pretende procurar x estiver *ordenado* a pesquisa pode ser efectuada de forma mais eficiente.

Como se procura um telefone numa lista?

Pesquisa binária

Seja m o índice $i/2$ (mais ou menos a meio). Comparemos x com $a[m]$.
– Se $x = v[m]$, *eureka*, a resposta é m.
– Se $x < v[m]$, o valor x só pode estar nos índices compreendidos entre 0 e $m-1$ (inclusivé).
– Se $x > v[m]$, o valor x só pode estar nos índices compreendidos entre $m+1$ e $n-1$ (inclusivé).
Só com uma comparação ou encontramos x ou eliminamos cerca de $n/2$ elementos!

Pesquisa binária – um passo da iteração

Sejam em cada passo a e b os índices entre os quais pode estar x. Isto é, pode ser $x = v[a]$ ou...ou $x = v[b]$ (não sabemos!)

- Inicialmente $a = 0$, $b = n-1$.
- Se for $a > b$ o intervalo é vazio, x não está em $v[\,]$!
- Seja $m = (a+b)/2$ (divisão inteira). Comparemos x com $a[m]$
  1. Se $x = v[m]$, `retorna` m
  2. Se $x < v[m]$, b passa a ser $m-1$
  3. Se $x > v[m]$, a passa a ser $m+1$

Pesquisa binária – algoritmo iterativo

```
int pb(int x,int v[], int n){
  a=0; b=n-1;
  while(a<=b){
    m=(a+b)/2;
    if(x==v[m])
      return(m);
    if(x<v[m])
      b=m-1;
    else
      a=m+1;  /* quando x>v[m] */
  }
  return(-1);
}
```


Pesquisa binária – algoritmo recursivo

Versão recursiva,
(chamada de "fora": `pb(x,0,n-1,v,n)`):

```
int pb(int x, int a, int b,int v[], int n){
  if(a>b)
    return(-1);
  m=(a+b)/2;
  if(x==v[m])
    return(m);
  if(x<v[m])
    pb(x,a,m-1,v,n);
  else
    pb(x,m+1,b,v,n);
}
```

Pesquisa: nota sobre a eficiência

- Pesquisa sequêncial: no máximo n comparações (de x com $v[i]$). Porquê?
- Pesquisa binária: no máximo cerca de $\log n$ comparações (de x com $v[m]$). Porquê?

Compare a eficiência dos 2 métodos – número de comparações – para

$$n = 10, 1000, 1000\,000, 1000\,000\,000$$

Programa de ordenação por selecção

Método para ordenar os `n` valores `a[1]`,...,`a[n]`:

```
Para i = 0, 2, ..., n-2
  Calcule min, o índice do
       menor elemento entre i e n-1
  troque a[min] <-> a[i]
```

Programa de ordenação por selecção **Programa:**

```
#include<stdio.h>
main(){
  int i,n=9;
  int a[9] = {0,2,9,15,-1,7,9,4,6};
  /* indice de 0 a 8                      */
  for(i=0;i<n-1;i++){
    int j, min=i, t;
    /* min: indice do menor entre i+1 e n */
    for(j=i+1;j<n;j++)
      if(a[j]<a[min]) min=j;
    /* Agora trocamos a[i] <-> a[min] */
    t=a[i]; a[i]=a[min];  a[min]=t;
  }
```


Programa de ordenação por selecção

Programa:

```
  for(i=1;i<=n;i++)
    printf("%2d ",a[i]);
}


/* Resultado: -1  0  2  4  6  7  9  9 15 */
```

Exercício 8 *Reorganize o programa anterior dividindo-o nas seguintes funções:*

```
void ler(int v[],int &n)  - lê o vector
int  minimo(int v[],int n,int k) -
      índice do mínimo v[k],...,v[n-1]
void ordena(int v[],int n) -
      ordena v[]; utiliza minimo(...)
main()  - utiliza ler() e ordena()
```

◇


```
i     0  1  2  3  4  5  6  7  8  |
a[i]  0  2  9 15 -1  7  9  4  6  |
-----------------------------------------------
a[i]  0  2  9 15 -1  7  9  4  6  |0..8: min=4
a[i] -1| 2  9 15  0  7  9  4  6  |1..8: min=4
a[i] -1  0| 9 15  2  7  9  4  6  |2..8: min=4
a[i] -1  0  2|15  9  7  9  4  6  |3..8: min=7
a[i] -1  0  2  4| 9  7  9 15  6  |4..8: min=8
a[i] -1  0  2  4  6| 7  9 15  9  |5..8: min=5
a[i] -1  0  2  4  6  7| 9 15  9  |6..8: min=6
a[i] -1  0  2  4  6  7  9|15  9  |7..8: min=8
a[i] -1  0  2  4  6  7  9  9|15  |7..8: min=8
-----------------------------------------------
 Ababou: se os n-2 primeiros estao ordenados,
 o a[n-1] esta' no sitio!
```

Explicação

Para estimar a eficiência, podemos contar o número $c(n)$ de vezes que a comparação da instrução (uma das que é mais vezes efectuada...) `if(a[j]<a[min])...` é efectuada.

$$c(n) = (n-1) + (n-2) + \cdots + 1 = n(n-1)/2$$

Dizemos que se trata de um algoritmo de ordem $O(n^2)$. A análise de algoritmos pode ser estudada noutra disciplina do curso.
Nota: Há muitos outros métodos de ordenação, alguns mais eficientes que este!

Ideias de ordenação ⇒ Exercícios!

Problema. Ordenar `a[ ]` com `n` elementos
(em `a[0]`, `a[1]`,..., `a[n-1]`)

Há muitos modos de ordenar...

Os exercícios seguintes consistem em transcrever a descrição informal para uma função em linguagem C.

Ideia 1: Contar os menores que ele

Exercício 9 *Usamos um vector auxiliar* `b[]` *onde vai ficar o resultado. Para cada elemento* `a[i]` *contamos quantos elementos de* `a[]` *são menores que ele; sejam* `m`.

Colocamos `a[i]` *em* `b[m]`.

Implemente este método de ordenação

- *Supondo que todos os* `b[m]` *são distintos.*
- *No caso geral (pode supor que os* `a[i]` *não são negativos).*

◇

Ideia 3: contagem de ocorrências

Exercício 10 *Suponhamos que os* `a[i]` *estão entre 0 e* `MAX-1` *e seja* `c[ ]` *um vector inicialmente com 0's.*

Cada valor `a[i]` $\Rightarrow$ *incrementa* `c[a[i]]`.

No fim percorre-se o vector `c[ ]`. ◇

Ideia 3: troca de vizinhos

Método da bolha, "bubblesort"

Exercício 11

– *Comparamos* `a[0]` *com* `a[1]`*; trocamos se* `a[0]`>`a[1]`

– ...

– *Comparamos* `a[n-2]` *com* `a[n-1]`*; trocamos se* `a[n-2]`>`a[n-1]`

Repetimos este processo (analisando `a[1..n-2]`*,* `a[2..n-2]`*...), até que não haja necessidade de trocar nenhum par de valores.* ◇

Ordenação não determinística I

Método para ordenar os `n` valores `a[1]`,...,`a[n]`:

```
Dados: a[] com elementos a[0]...a[n-1]
Resultado:
   a[] ordenado por ordem não decrescente

enquanto houver a[i]>a[i+1]
  seleccione um desses i
  troque a[i] <-> a[i+1]
```

Nota. O método da bolha ("bubblesort") é deste tipo.

Ordenação não determinística II

Perguntas:

1. Quando (e se) acaba está ordenado?
2. Acaba sempre?
3. Quantas trocas tem que efectuar (no máximo) qualquer programa deste tipo?

Respondendo às perguntas...

Perguntas:

Quando (e se) acaba, o vector está ordenado?
Sim porque então é sempre $a[i] \leq a[i+1]$ para $0 \leq i \leq n-2$.

Respondendo às perguntas...

Acaba sempre?
Representemos por $N(i)$, $0 \leq i < n$, o número de índices j tais que $j < i$ e $a[j] > a[i]$ ("inversões"). Por exemplo:

```
Pos:     0 1 2 3 4
        -------------
Valor: | 5 2 3 6 1 |
        -------------
N(i):    0 1 1 0 4
```

e seja N a soma de todos os $N(i)$ ($N = 6$ no exemplo); N é pois o número de pares (j, i) com $j < i$ e $a[j] > a[i]$

Respondendo às perguntas...

Proposição Quando, no algoritmo não determinístico que estamos a estudar, se efectua uma troca `a[i] <-> a[i+1]`, o valor de N diminui de uma unidade.
Demonstre!
Corolário O algoritmo acaba sempre (como "convém").
Demonstre!
Corolário O algoritmo efectua exactamente N trocas.
Demonstre!
Corolário O tempo de execução do algoritmo é pelo menos proporcional a n^2.
Demonstre!

Uma implementação do "bubblesort"

```
void bs(int a[],int n){
  int i,t,acabou=0;
  while(!acabou){
      acabou=1;
      for(i=0;i<=n-2;i++)
        if(a[i]>a[i+1]){
          t=a[i];a[i]=a[i+1];a[i+1]=t;
          acabou=0;
        }
  }
}
```

Variações e melhorias...

"Mergesort"

Este método de ordenação (muito eficiente, mesmo no pior caso), foi explicado nas aulas teóricas, ver "exemplos das aulas teóricas" na página da disciplina.

"String": vector de caracteres

String: vector de caracteres

- Um "string" ou cadeia de caracteres é um vector (array) de caracteres. Termina com o caracter de código 0 – que não é o caracter `'0'` (código 48)
- "`char *p;`" ou "`char p[ ];`" "`p`" é um apontador para caracter – mas mais nenhum espaço é reservado.
- "`char p[10];`": "`p`" aponta para um espaço de 10 caracteres – o primeiro é `char p[0]`, o último é `char p[9]`. Coloquemos um "string" em `p p[0]='a'; p[1]='n'; p[2]='a'; p[3]='!'; p[4]=0;`

String: vector de caracteres

- "`char *p; p="batatas e couves";`": `p` passa a apontar para (o início) de uma cadeia de caracteres.
- "`char *a[10];`": vector de 10 posições, cada uma das quais é um apontador para caracteres = vector de "stringues".
  `a[2]="vou!!!"`
- "`char a[5][10];`": vector de "stringues" com espaço reservado. Cada `a[i]` é uma "linha" de 10 caracteres.
  `a[2][0]='v'; a[2][1]='o';... a[2][6]=0;`

Strings: Exercícios

Definiu-se

```
char p; char *q; char a[10]; char b[2][5];
```

Comente as seguintes instruções, escrevendo, se erradas, possíveis versões correctas com o mesmo lado esquerdo.

`b[2][0]='v';`	`p='v';`
`p="v";`	`a="array";`
`q[0]=' ';`	`strcpy(a,"array");`
`strcpy(b[1],"xato");`	`strcpy(q,"array");`
`strcpy(a,"array");` `i=strcmp(a,"arrayzinho");`	`strcpy(a,"array");` `i=strlen(a);`

Strings: Exercícios

Exercício 12 *São lidos 2 "strings". O programa deve imprimir "*`menor`*" "*`maior`*" ou "*`igual`*" quando o primeiro "string" seja respectivamente maior, menor ou igual que o segundo (por ordem lexicográfica).*

Exemplo:

```
? abra
? abracadabra
menor
```

◇

Possível resolução (compara)

```
//-- Retorma -1, 0 ou 1 conforme s1<s2, s1=s2 ou s1>s2
int compara(char s1[],char s2[]){
  int i=0;
  while(s1[i]!=0 && s2[i]!=0){
    if(s1[i]<s2[i]) return(-1);
    if(s1[i]>s2[i]) return(1);
    i++;
  }
  if(s1[i]==0 && s2[i]==0) return(0);
  if(s1[i]==0)             return(-1);
  return(1);
}
```

Possível resolução (main)

```
main(){
  char s1[MAX], s2[MAX];
  int t;
  printf("? "); scanf("%s",s1);
  printf("? "); scanf("%s",s2);
  t=compara(s1,s2);
  if(t==-1)
    printf("menor\n");
  else
    if(t==1)
      printf("maior\n");
    else
      printf("igual\n");
}
```


Outra possível resolução (compara)

```
//-- Retorma -1, 0 ou 1 conforme seja
//   s1<s2, s1=s2 ou s1>s2
int compara(char *s1,char *s2){
  while(*s1!=0 || *s2!=0){
    if(*s1<*s2)
      return(-1);
    if(*s1>*s2)
      return(1);
    s1++; s2++;
  }
  return(0);
}
```

Strings: Exercícios

Exercício 13 *São lidos 2 "strings". O programa deve imprimir todas as ocorrências do primeiro no segundo.*

Exemplo:

```
? aa
? aaabaa
Posicao 0
Posicao 1
Posicao 4
   --- Explicação, ocorrências de 'aa' em 'aaaba' ---
aaabaa  |   .aa.abaa     a.aa.baa     aaab.aa
012345       0             2              4
```

◇

Possível resolução (prefixo)

```
//-- 1 ou 0 conforme s1 seja ou nao
//   prefixo de s2
int prefixo(char s1[],char s2[],int m){
  int i=0;
  while(s1[i]!=0 && s2[m+i]!=0){
    if(s1[i]!=s2[m+i])
      return(0);
    i++;
  }
  if(s1[i]==0)
    return(1);
  return(0);
}
```

Possível resolução (main)

```
main(){
  char s1[MAX], s2[MAX];
  int n;
  printf("? "); scanf("%s",s1);
  printf("? "); scanf("%s",s2);
  for(n=0;s2[n]!=0;n++){
    if(prefixo(s1,s2,n))
      printf("Posicao %d\n",n);
  }
}
```


Strings: Exercícios

Exercício 14 *Alterar o programa de ordenação por selecção com vista a ordenar um vector de "strings" inicializado de forma apropriado, por exemplo,*

```
char *v[]={"nuvem","ceu","mar"};
```

◇

Exercício 15 *Indicar todas as linhas (número) de um ficheiro em que ocorre um "string" dado. O "string" está na primeira linha do ficheiro. Exemplo:*

```
qq
aqqq
abcda dqaq
aabb..ppqqrr..
```

Resultado:

```
qq ocorre nas linhas: 1 3
```

◇

Parametros vindos da "shell" (Unix)

O "main" com parametros!

```
main(int n, char *v[]){
  int i;
  for(i=0;i<n;i++)
    printf("Par %d: %s\n",i,v[i]);
}
-------------
Resultado:
$ a.out bife estrelado
Par 0: a.out
Par 1: bife
Par 2: estrelado
```


O "main" com parametros!

Experimente:

```
$ a.out "bife estrelado"
$ a.out *
```

O "`main`" com parametros!

Queremos escrever um programa com o seguinte comportamento

```
$ a.out 111 222
Produto= 24642
```


O "`main`" com parametros!

```
//- programa que multiplica 2 inteiros
main(int npars, char *par[]){
  int a,b;
  if(npars!=3){
    printf("Erro!\n"); exit(1);
  }
  sscanf(par[1],"%d",&a);
  sscanf(par[2],"%d",&b);
  printf("Produto= %d\n",a*b);
}
```

Como detectar erros? Por exemplo, "`$ a.out ab xx`".

O "`main`" com parametros!

Exercício 16 *Experimentar o comando Unix* `echo`.
Implementar um comando idêntico ao `echo` *chamado* `lista`. ◇

Nota importante. Os comandos do Unis – `echo`, `cat`, `ls`... – foram todos feitos como neste exercício: programas escritos em linguagem C (ou C++)!

Ficheiros

Instruções com ficheiros

- Declaração de ficheiros
- Abertura de ficheiros
- Fecho
- Leitura de ficheiros
- Escrita em ficheiros

Ficheiros e redirecção...

A maneira mais fácil de usar ficheiros: redirecção do `stdin` e `stdout`
Exemplo: Copiar um ficheiro para outro (se este existia, é destruído...).

```
main(){
  int c;
  while((c=getchar())!=EOF)
    putchar(c);
}
```

Ficheiros e redirecção...

Uso:

```
$ gcc teste.c -o copy
$ copy < aaa.tex > lixo
```

O redireccionamento tanbém funciona com as instruções de I/O `scanf`, `printf`, ...

Nomes de ficheiros fixos...

Exemplo: Copiar o ficheiro "`aaa`" para o ficheiro "`lixo`".

Ficheiros: Uso explícito...

```
main(){
  FILE *f1, *f2;
  int c;
  f1=fopen("aaa","r");
  f2=fopen("lixo","w");
  if(f1==NULL || f2==NULL){
    printf("Erro de ficheiros\n");
    exit(1);
  }
  while((c=getc(f1))!=EOF)
    putc(c,f2);
  fclose(f2); /* nao e' preciso... */
}
```


Nomes de ficheiros dados no comando...

Exemplo: Copiar um ficheiro para outro (se este existia, é destruído...).

O primeiro parâmetro é a origem e o segundo o destino (da cópia).

```
$ copy aaa lixo
```

```
main(int argc, char *argv[]){
  FILE *f1, *f2;
  int c;
  if(argc!=3){
    printf("copy: uso: copy <de> <para>\n");
    exit(1);}
  f1=fopen(argv[1],"r");  f2=fopen(argv[2],"w");
  if(f1==NULL || f2==NULL){
    printf("Erro de ficheiros\n");exit(1);}
  while((c=getc(f1))!=EOF) putc(c,f2);
}
```



```
$ copy aaa lixo
```

Uso de ficheiros ("streams") "bufferizados"

- Definição: `FILE *f`
- Abertura: `fopen(ficheiro,modo)`
  `modo` inclui (entre outros) `"r"` e `"w"`.
- Fecho: `int fclose(fp)`
  Automático no fim normal da execução.
- Posicionamento: `int fseek(fp,desloc,rel)`
  `desloc`: movimento em número de "bytes"
  `rel`: inteiro, `SEEK_SET`, `SEEK_CUR`, `SEEK_END`: relativamente ao início, posição corrente ou fim do ficheiro (respectivamente).

Algumas funções relativas a ficheiros

- Leitura básica: `getc(fd)` e `getchar()` (`stdin`)
- Escrita básica: `putc(c,fd)` e `putchar(c)` (`stdout`)
- Leitura formatada:
  `fscanf(fd,formato,x,...)`
  `scanf(formato,x,...)` (`stdin`)
- Escrita formatada:
  `fprintf(fd,formato,x,...)`
  `printf(formato,x,...)` (`stdout`)
- Também de interesse: formatação de/para "strings"
  `sscanf(char *string, formato,x,...)`
  `sprintf(char *string, formato,x,...)`
  e `scanf(formato,x,...)` (`stdin`)

Programa: Números num ficheiro...

```
#include<stdio.h>
#define NAO  0
#define SIM  1
char digito(int);

main(int argc, char *argv[]){
  FILE *f;
  int c;
  char escrevendo;
  if(argc!=2){printf("Uso: nums ficheiro\n"); exit(1);}
  f=fopen(argv[1],"r");
  if(f==NULL){printf("Erro de ficheiros\n");  exit(1);}
}
```


Programa: Números num ficheiro - cont.

```
  escrevendo=NAO;
  while((c=getc(f))!=EOF){
    if(digito(c)){
      if(!escrevendo){escrevendo=SIM; putchar('\n');}
      putchar(c);
    }
    else escrevendo=NAO;
  }
}

char digito(int c){
  return(c>='0' && c<='9');
}
```

Programa: Número num "string"...

Exercício 17 *Escreva uma função*
`int get_num(char *a)`
que dê como resultado o primeiro número presente no "string" `a`.
Um número (inteiro) é uma sequência de 1 ou mais dígitos possivelmente precedida do sinal `+` *ou* `-`. *Se não existir nenhum número em* `a`, *a variável global* `erro` *deverá tomar o valor 1.*
Por exemplo, se `a` = `"batman-33a222"`, *o resultado deverá ser* `-33` *(inteiro).* ◇

Uma solução...

```
BOOLEAN digito(int);
BOOLEAN sinal(int);

int get_num(char *a){
  int res=0, i=0, factor=1, soma=0;
  while(a[i]!=0 && !digito(a[i]) && !sinaol(a[i])) i++;
  if(a[i]==0)  {erro=1; return(0);} // Nao tem digitos?
  if(a[i]=='-'){factor=-1; i++;}    // Sinal -?
  if(a[i]=='+') i++;                // Sinal +?
  if(!digito(a[i])){erro=1; return(0);} // erro!
  while(a[i]!=0 && digito(a[i])){
    soma = soma*10+(a[i]-'0'); i++;}
  return(factor*soma);
}
```

Uma solução...

```
BOOLEAN digito(int c){
  return(c>='0' && c<='9');
}

BOOLEAN sinal(int c){
  return(c=='+' || c=='-');
}
```


Exercícios I

Escreva uma função que permita a leitura dos dados (nome e telefone) de um ficheiro sem usar `fscanf`. O ficheiro deve ter um formato do seguinte tipo

```
joaquina,   016002222
ana pinto,  0671445566
joao costa, 8442222
```

Cada nome é terminado por vírgula (“,”) e cada telefone por `\n`. Os nomes e telefones podem ser precedidos por espaços. Note que um nome pode incluir (como no exemplo) espaços.

A seguinte organização em funções pode ser útil...

```
int nao_ch(ch) ---
   proximo caracter (int) que nao
   e' ch (ch pode ser espaco...)
char *palavra(ch,cht) ---
   o resultado e' a palavra
   formada por ch seguido dos
   proximas caracteres lidos ate'
   cht (exclusive').
```

Na função `palavra` o "string" resultado deve corresponder a um "array"de caracteres interno declarado como `static`, por exemplo

```
static temp[MAX]
```


Exercício 172: Uma solução

```
#include <stdio.h>
int nao_ch(int);
char *palavra(int,int);
FILE *f;

#define MAX 100
```

```
main(){
  int ch; char buf1[MAX], buf2[MAX];
  f=fopen("lista","r");  if(f==NULL) exit(1);
  while(ch!=EOF){
    ch=nao_ch(' ');
    if(ch!=EOF){
      strcpy(buf1,palavra(ch,','));
      if(ch!=EOF){
        ch=nao_ch(' ');
        strcpy(buf2,palavra(ch,'\n'));
        if(ch!=EOF)
          printf("Nome: %20s  Tel: %10s\n",buf1,buf2);
      }
    }
  }
}
```


Uma solução

```
int nao_ch(int ch){
  int c;
  while((c=getc(f))==ch && ch!=EOF);
  return(c);
}
char *palavra(int ch, int cht){
  int i=1; static char a[MAX];
  a[0]=ch;
  while((a[i]=getc(f))!=cht && a[i]!=EOF) i++;
  a[i]=0;
  return(a);
}
```

```
-------------------------------------------
Nome:              joaquina  Tel:  016002222
Nome:            ana pinto  Tel: 0671445566
Nome:           joao costa  Tel:    8442222
-------------------------------------------
```


Exercício 18 *Altere o programa de consulta por forma a que o ficheiro lido seja ordenado de modo a permitir a utilização da pesquisa binária (vantajosa em termos de eficiência).*
A estrutura geral pode ser algo como

```
main(){
ler a lista;
ordenar a lista;
ciclo{
  ler um nome N;
  se N=="0" termina;
  procura (pesq. binária) na lista o nome N
  se encontra, imprime o telefone;
  se nao, diz que nao existe esse nome
}
}
```

Notas sobre

inteiros de precisão arbitrária

Factorial

```
int fact(int n){
  int p=1,f;
  for(f=2;f<=n;f++)
    p*=f;
  return(p);
}

main(){
  int i;
  for(i=0;i<=20;i++)
    printf("fact(%3d) = %d\n",i,fact(i));
}
```

Resultados – o problema

```
fact(  0) = 1
fact(  1) = 1
fact(  2) = 2
...
fact( 12) = 479001600
fact( 13) = 1932053504
fact( 14) = 1278945280
fact( 15) = 2004310016
fact( 16) = 2004189184
fact( 17) = -288522240
...
fact( 20) = -2102132736
```


Factorial

- O problema
- Uma solução

Factoriais grandes

Representamos o inteiro num vector de modo que:
– Cada elemento do vector contem um dígito numa base que é uma potência de 10 – porquê?
– O dígito menos significativo está na posição de índice 0 – porquê?

```
1 307 674 368 000  =>
     dig[] = [0, 368, 674, 307, 1], nd=5
     pos      0    1    2    3  4
```


Impressão

A impressão exige cuidados. 1012 não deve ficar 1 12!
Vamos aproveitar as potencialidades do `printf`.

```
void imprime(int dig[],int nd){
  int i;
  printf("%4d",dig[nd-1]);
  for(i=nd-2;i>=0;i--)
    printf(" %03d",dig[i]);
  printf("\n");
}
```

Factorial em precisão (quase) arbitrária

```
main(){
  int n,i,
   dig[MAX]={0}, nd;
  scanf("%d",&n);

  nd=1;
  dig[0]=1;

  for(i=2;i<=n;i++) mult(i,dig,&nd);

  imprime(dig,nd);
}
```


Continuação – Multiplicar grande por pequeno

```
void mult(int m,int dig[],int *nd){
  int i=0,c=0,valor;
  do{
    valor=dig[i]*m+c;
    dig[i]=valor%BASE;
    c=valor/BASE;
    i++;
  } while(i<*nd || c>0);
  if(i>*nd)
    *nd=i;
}
```

200!, correndo o programa...

```
# fact
 200    (dados)
788 657 867 364 790 503 552 363 213 932 185 062 295 135 977
687 173 263 294 742 533 244 359 449 963 403 342 920 304 284
011 984 623 904 177 212 138 919 638 830 257 642 790 242 637
105 061 926 624 952 829 931 113 462 857 270 763 317 237 396
988 943 922 445 621 451 664 240 254 033 291 864 131 227 428
294 853 277 524 242 407 573 903 240 321 257 405 579 568 660
226 031 904 170 324 062 351 700 858 796 178 922 222 789 623
703 897 374 720 000 000 000 000 000 000 000 000 000 000 000
000 000 000 000 000
```


200!, correndo o programa...

Porque é que termina em tantos zeros?

Algumas notas sobre

variáveis bi-indexadas

(matrizes)

Variáveis bi-indexadas

Declarando:

```
#define MAXL 3
#define MAXC 5
int v[MAXL][MAXC];
```

Usando:

```
for(i=0;i<MAXL;i++)
  for(j=0;j<MAXC;j++)
    v[i][j]=(i+j)%10;
```

Variáveis bi-indexadas

Imprimindo:

```
for(i=0;i<MAXL;i++){
  for(j=0;j<MAXC;j++)
    printf("%3d",v[i][j]);
  printf("\n");
}
```


Continuação: um pequeno programa

```
#define MAXL 3
#define MAXC 5

void inicia(int v[][MAXC],int nl, int nc){
  int i,j;
  for(i=0;i<nl;i++)
    for(j=0;j<nc;j++)
      v[i][j]=(i+j)%3;
}
```

Continuação: um pequeno programa

```
void imprime(int v[][MAXC],int nl, int nc){
  int i,j;
  for(i=0;i<MAXL;i++){
    for(j=0;j<MAXC;j++)
      printf("%3d",v[i][j]);
    printf("\n");
  }
}
```


Continuação: um pequeno programa

```
main(){
  int v[MAXL][MAXC],i,j;
  inicia(v,MAXL,MAXC);
  imprime(v,MAXL,MAXC);
}
```

Continuação: um pequeno programa

```
  int v[2][3]={{4,5,6},{9,8,6}};

             0 1 2 3 4 5
Na memória: [4,5,6,9,8,6];

Índice de v[i][j] => j*3+i
Exemplo: v[1][2] (=6) <=> 1x3+2=5
```


Resolvendo uma equação diferencial...

Temperaturas fixadas numa parte de um rectângulo (aproximado por uma matriz):

```
9 9 9 . . . . 0
9 . . . . . . 0
8 8 8 8 8 . . 0
9 . . . . . . 0
8 8 8 . . . . 1
```

O problema: Determinar as temperaturas dos pontos marcados com "." quando se atingir uma situação de equilíbrio.

Resolvendo uma equação diferencial...

O problema podia ter sido formulado de outros modos: por exemplo, utilizando potenciais eléctricos em vez de temperaturas.

A equação diferencial satisfeita pela temperatura (ou pelo potencial eléctrico) quando aproximamos o espaço a 2 dimensões por uma matriz, pode formular-se do seguinte modo:

$$v[i][j] = \frac{v[i-1][j] + v[i][j-1] + v[i+1][j] + v[i][j+1]}{4}$$

isto é, o valor em cada ponto deve ser igual à média dos valores nos 4 vizinhos indicados:

```
  2
3 1 0
 -1
```


Resolvendo uma equação diferencial...

Vamos marcar numa matriz os pontos de temperatura fixa:

```
int t[][]={
int m[][NC]={
   {1, 1, 1, 0, 0, 0, 0, 1},
   {1, 0, 0, 0, 0, 0, 0, 1},
   {1, 1, 1, 1, 1, 0, 0, 1},
   {1, 0, 0, 0, 0, 0, 0, 1},
   {1, 1, 1, 0, 0, 0, 0, 1} };
```

...método...

Colocamos um valor inicial – por exemplo 0 – em cada célula de temperatura não fixa.

Em cada iteração o valor de cada célula é substituído pela média dos valores das 4 células vizinhas.

Quando a maior modificação não exceder um valor muito pequeno pré-fixado, por exemplo 0.001, consideramos o processo terminado.

Programa: `main` – declarações

```
main(){
  double temp[][NC]={
    {9, 9, 9, 0, 0, 0, 0, 0},
    {9, 0, 0, 0, 0, 0, 0, 0},
    {8, 8, 8, 8, 8, 0, 0, 0},
    {9, 0, 0, 0, 0, 0, 0, 0},
    {8, 8, 8, 0, 0, 0, 0, 1} },
  delta=0.001,diff;
```

Programa: `main` – declarações

```
int m[][NC]={
 {1, 1, 1, 0, 0, 0, 0, 1},
 {1, 0, 0, 0, 0, 0, 0, 1},
 {1, 1, 1, 1, 1, 0, 0, 1},
 {1, 0, 0, 0, 0, 0, 0, 1},
 {1, 1, 1, 0, 0, 0, 0, 1} },
 niter=0;
```


Programa: `main` – declarações

```
  inicia(m,temp,NL,NC);
  do{
    niter++;
    diff=iteracao(m,temp,NL,NC);
  } while(diff>delta);
  printf("Convergiu em %d iteracoes\n",niter);
  imprime(temp,NL,NC);
}
```

Programa: `inicia`

```
#define NL 5
#define NC 8

void inicia(int m[][NC],double temp[][NC],int nl,int nc){
  int i,j;
  for(i=0;i<nl;i++)
    for(j=0;j<nc;j++)
      if(!m[i][j])
        temp[i][j]=0;
}
```


Programa: `imprime`

```
void imprime(double t[][NC],int nl,int nc){
  int i,j;
  for(i=0;i<nl;i++){
    printf("\n");
    for(j=0;j<nc;j++)
      printf("  %3.1f",t[i][j]);
  }
  printf("\n");
}
```

Programa: iteração

```
double iteracao(int m[][NC],double t[][NC],
                int nl,int nc){
  double diff=0,s,y;  int i,j;
  for(i=0;i<nl;i++)
    for(j=0;j<nc;j++)
      if(!m[i][j]){
        s=0;
        if(i-1>=0) s+=t[i-1][j];  if(j-1>=0) s+=t[i][j-1];
        if(i+1<nl) s+=t[i+1][j];  if(j+1<nc) s+=t[i][j+1];
        s=s/4;     y=fabs(s-t[i][j]); if(y>diff) diff=y;
        t[i][j]=s;
      }
  return(diff);
}
```


Resultados. Com delta=0.001, obtivemos o seguinte resultado

```
Convergiu em 15 iteracoes
 ---------------                    -----
 |9.0  9.0  9.0 |4.5  2.8  1.6  0.7 |0.0|
 |    -----------                   |   |
 |9.0 |8.5  8.0  6.4  5.0  3.0  1.4 |0.0|
 |    ---------------------         |   |
 |8.0  8.0  8.0  8.0  8.0 |3.9  1.7 |0.0|
 |    ---------------------         |   |
 |9.0 |8.1  7.6  6.2  5.0  3.0  1.4 |0.0|
 |    -----------                   |   |
 |8.0  8.0  8.0 |4.2  2.7  1.7  1.0 |1.0|
 ---------------                    -----
```

Variáveis multi-dimensionais

- Generalização para mais dimensões

Números pseudo-aleatórios

Relativamente a esta matéria dada nas aulas teóricas, ver "exemplos das aulas teóricas" na página da disciplina. Indicamos apenas um índice dos temas abordados.

1. Números aleatórios em computadores determinísticos? Números pseudo-aleatórios.
2. Aplicações
   (a) Estimação de probabilidades (jogo dos dados...)
   (b) Integração definida: área entre o eixo dos x ($x \in [a, b]$) e uma função não negativa $f(x)$.
   (c) "Passeio alatório"
   (d) Simulação de um concurso

Interlúdio

Resolução de alguns problemas

Problema: Soma de alguns dígitos

São dados 2 inteiros n e b. Suponhamos que n escrito na base b é:

$$d_m d_{m-1} ... d_3 d_2 d_1$$

Escreve um programa que imprima a soma:

$$d_1 + d_3 + d_5 \cdots \ (d_m \text{ é incluído se } m \text{ for ímpar})$$

EXEMPLO: $n = 2817$, $b = 10$. O resultado deve ser 15.

NOTA: Os dados devem ser lidos com as instruções:
"`scanf("%d",&n);`" e "`scanf("%d",&b);`"
O resultado deve ser impresso com uma instrução do tipo:
"`printf("%d",s);`", não devendo haver mais instruções de saída.

Uma solução

```
main(){
  int n,b,s=0;
  scanf("%d",&n);
  scanf("%d",&b);
  while(n>0){
    s+=n%b;
    n=n/(b*b);
  }
  printf("%d",s);
}
```

Problema: Valor do primeiro maior....

Considera a sequência $X_0, X_1, \cdots$ definida por

$$X_0 = X_1 = 1$$
$$X_n = X_{n-2} - X_{n-1} \quad \text{para } n > 1$$

Escreve um programa que a partir de um valor $n > 1$ dado, imprima o primeiro valor da sequência superior a n.

EXEMPLO: Para n = 30 o resultado deve ser 34.
NOTA: Os dados devem ser lidos com a instrução: `scanf("%d",&n);`.
O resultado deve ser impresso com uma instrução do tipo:
`printf("%d",s);` não devendo haver mais instruções de saída!

Uma solução

```
main(){
  int x=1,y=1,n,t;
  scanf("%d",&n);
  while(y<=n){
    t=x-y;
    x=y;
    y=t;
  }
  printf("%d",y);
}
```

Menor que os vizinhos...

Lido $n \geq 3$, determina numa sequência de$n$ inteiros, lidos um a um, o número de valores que são menores do que os seus dois vizinhos. Por exemplo, para:

```
7 3 4 1 6 5 8 4
```

O resultado é 2, porque os números 1 e 5 da sequência verificam essa condição.

Em todo o programa, deve apenas existir uma única chamada à função printf com o formato `"%d"`.

Uma solução

```
6 2 5 3 6 7 5    =>  1  (elemento 3)
```

Uma solução

```
main(){
  int n,a,b,c,i,soma=0;
  scanf("%d",&n);
  scanf("%d",&a);
  scanf("%d",&b);
  for(i=3;i<=n;i++){
    scanf("%d",&c);
    if(b<a && b<c) soma++;
    a=b;
    b=c;
  }
  printf("%d",soma);
}
```