FORÇA AÉREA
PORTUGUESA

FUNÇÕES

FUNÇÕES > exercício

Escreva um programa que tenho o seguinte output.
Use ciclos.

FUNÇÕES

Exemplos de funções que já vimos?

`printf`, ...?

FUNÇÕES

Exemplos de funções que já vimos?
`printf`, ...?

Reutilização de código

Organização do programa

FUNÇÕES > Características

nome único

recebe parâmetros

tem um tipo (pode ser `void`)

retorna valor

FUNÇÕES > Características

nome único

tem um tipo
(pode ser `void`)

recebe
parâmetros

retorna valor

definição de uma função

```
tipoX nome(tipo1 param1, ..., tipoN paramN){
    //...
    return valor; // valor é do tipo tipoX
}
```

FUNÇÕES > Características

nome único

recebe parâmetros

tem um tipo (pode ser `void`)

retorna valor

variáveis locais

automaticamente inicializados

FUNÇÕES

```
char shift_char(char c){
  char new_c;
  new_c = c + 3;
  return new_c;
}
```

Nome da função?
Tipo da função?
Parâmetros?
Tipos dos parâmetros?
O que retorna?
Tipo do valor retornado?

FUNÇÕES

```
char shift_char(char c){
  char new_c;
  new_c = c + 3;
  return new_c;
}
```

Nome da função?	`shift_char`
Tipo da função?	`char`
Parâmetros?	`c`
Tipos dos parâmetros?	`char`
O que retorna?	new_c
Tipo do valor retornado?	char

FUNÇÕES

```
double square_root(double x, double guess, double error){
  double new_guess;
  if (absolute_value(guess * guess - x) < error) return guess;

  new_guess = (guess + x / guess) / 2;

  return square_root(x, new_guess, error);
}
```

Nome da função?
Tipo da função?
Parâmetros?
Tipos dos parâmetros?
O que retorna?
Tipo do valor retornado?

FUNÇÕES

```
double square_root(double x, double guess, double error){
  double new_guess;
  if (absolute_value(guess * guess - x) < error) return guess;

  new_guess = (guess + x / guess) / 2;

  return square_root(x, new_guess, error);
}
```

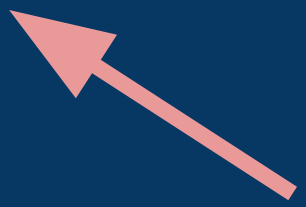
Nome da função?	`square_root`
Tipo da função?	`double`
Parâmetros?	`x, guess, error`
Tipos dos parâmetros?	`double, double, double`
O que retorna?	valor retornado por `square_root`
Tipo do valor retornado?	`double`

FUNÇÕES

```
double square_root(double x, double guess, double error){
  double new_guess;
  if (absolute_value(guess * guess - x) < error) return guess;

  new_guess = (guess + x / guess) / 2;

  return square_root(x, new_guess, error);
}
```



Quando uma função se chama a si mesma, estamos a usar **recursão**.

FUNÇÕES > Características

executadas quando invocadas

invocadas por outras funções

parâmetros recebidos são copiados e guardados em variáveis locais

programa é “suspenso” até funcão terminar

FUNÇÕES > Características

tarefa bem definida

caixa negra

o mais independentes possível

FUNÇÕES > Parâmetros

Em C, os parâmetros de tipos primitivos são tratados como variáveis locais inicializadas no momento da chamada da função.

```
int calcular_factorial(int num){
  return (num == 0) ? 1 : num *
calcular_factorial(num - 1);
}

int main(){
  int numero = 5;
  printf("%d", calcular_factorial(numero));
}
```

Depois de terminada a execução de uma determinada função, todas as suas variáveis locais são destruídas.

FUNÇÕES > Parâmetros

Os parâmetros de uma função podem ter qualquer tipo de dados:

```
função (char y,  int a,  float k, TipoXPTO  varX)
```

Qualquer expressão válida pode ser enviada como argumento para uma função.

```
sqrt(1.4 * some_value - 4)
```

FUNÇÕES > Parâmetros

O número e o tipo de valores enviados para a função tem de corresponder ao número e tipo de parâmetros existentes no cabeçalho da função.

```c
int func(int x, int y){
  return x + y;
}

int main(){
  int numero = 5;
  int val = func(numero);
}
```

```
main.c: In function 'main':
main.c:7:13: error: too few
arguments to function 'func'
   int val = func(numero);
             ^~~~
main.c:1:5: note: declared here
 int func(int x, int y){
     ^~~~

exit status 1
```

FUNÇÕES > `return`

O **`return`** termina a execução de uma função;

Se estiver dentro da função **`main`**, termina programa;

A seguir a **`return`** pode ser colocada qualquer expressão válida;

Uma função pode conter várias instruções `return`, mas apenas uma é executada.

```
int modulo(int x){
  if (x >= 0) return x;
  return -x;
}
```

FUNÇÕES > protótipo

Cabeçalho da função seguido de ;

Serve para o compilador saber qual o tipo de retorno das funções.

```c
int main(){
  int numero = -5;
  printf("%d\n", modulo(numero);
}

int modulo(int x){
  if (x >= 0) return x;
  return -x;
}
```

```
warning: implicit declaration of
function ‘modulo’
[-Wimplicit-function-declaration
]|
```

FUNÇÕES > protótipo

Cabeçalho da função seguido de ;

Serve para o compilador saber qual o tipo de retorno das funções.

```
int modulo(int x);

int main(){
  int numero = -5;
  printf("%d\n", modulo(numero));
}

int modulo(int x){
  if (x >= 0) return x;
  return -x;
}
```

FUNÇÕES > exercício

Escreva um programa que tenha o seguinte output. Use a função `main` disponibilizada e escreva a função `cprint`.

```
int main(){
    cprint('+', 3);
    cprint('*', 5);
    cprint('-', 7);
    cprint('+', 5);
    cprint('*', 3);
}
```

```
+++
*****
-------
+++++
***
```

FUNÇÕES > exercício

Escreva um programa que some 2 números inteiros (introduzidos pelo utilizador) e grave o resultado numa variável global.

Utilize uma função para efetuar a soma e a gravação do resultado;

Imprima o resultado na função `main`.

FUNÇÕES > funções privadas

Apenas acessíveis dentro do ficheiro que lhe deu origem (**múltiplos ficheiros**);

```
static void apresenta_valor(int valor){ // Apresenta valores desde 0 até "valor"

    int i = 0;

    for(i = 0; i <=valor; i = i + 1)
        printf("%d\n", i);
}
```