Linguagem C
Introdução à programação em C

Emanuel Camacho

earc96@hotmail.com
a33878@ubi.pt

2015

Mapa de conceitos

Conceitos básicos

A linguagem C, tais como todas as outras, recebe, processa, e devolve os dados fornecidos. Resultados incompatíveis com os esperados, significam um grave erro no algoritmo, ou seja, a mal utilização das potencialidades do programa em que se programa.

Conceitos básicos

A estrutura de um programa em C é sempre do tipo :

```
diretivas para o compilador
main()
{
        declarações;
        instruções;
}
```

As diretivas são bibliotecas que armazenam funções predefinidas. Quando as queremos utilizar temos de incluir as bibliotecas que contêm essas mesmas funções no cabeçalho.

Exemplo

Para utilizar a função predefinida sin(), temos de incluir a biblioteca que contém essa mesma função, neste caso é a math.h. Assim, tem de ser escrito no cabeçalho #include <math.h> antes do programa principal, na zona das diretivas para o compilador.

Conceitos básicos

Em C, a leitura de dados é dada pela função scanf(), o processamento é dado por um conjunto de instruções e a devolução de dados é dada pela função printf(). Estas funções essenciais encontram-se na biblioteca <stdio.h>, ou seja, para as utilizar, tem de se escrever #include <stdio.h> no cabeçalho.

Exemplo.: Ler um valor e mostrá-lo.

```
#include <stdio.h>
main()
{
        int a;
        scanf("%d",&a);
        printf("O valor é %d",a);
}
```

Aspectos a reter

- Tivemos de escrever #include <stdio.h> pois estamos a utilizar funções predefinidas da biblioteca <stdio.h> (scanf() e printf()).
- Tivemos de escrever *int a* para declarar uma variável do tipo inteiro. Se fosse do tipo real, seria *float a*.
- Em relação ao scanf, o "%d" significa que a variável vai receber um valor inteiro. Se fosse do tipo real, seria "%f". O segundo parâmetro, &a, significa que o valor inserido irá ser armazenado na variável *a*.
- Em relação ao printf, vai ser escrito no ecrã o valor da variável *a*, que é do tipo inteiro.

Exemplo.: Ler dois valores reais, somá-los, e mostrar o resultado.

```
#include <stdio.h>
main()
{
        float a,b;
        scanf("%f %f",&a,&b);
        printf("A soma é %f",a+b);
}
```

Exemplo.: Ler dois valores reais, somá-los, e mostrar o resultado.

```
#include <stdio.h>
main()
{
        float a,b,soma;
        scanf("%f %f",&a,&b);
        soma = a+b;
        printf("A soma é %f",soma);
}
```

Exemplo.: Ler um carater e mostrá-lo.

```
#include <stdio.h>
main()
{
        char a;
        scanf("%c",&a);
        printf("O caracter é %c",a);
}
```

Outros operadores importantes em C

Símbolos	**Significado**
&&	Interseção
||	Reunião
!	Negação
<	Menor
>	Maior
<=	Menor ou Igual
>=	Maior ou Igual
==	Comparação (igualdade)
!=	Comparação (diferença)
%	Resto da divisão

Instruções condicionais

As instruções condicionais em C, têm uma grande importância pois podemos obter diferentes resultados, dependendo das restrições impostas anteriormente.

Existem três estruturas condicionais:

- Estrutura *if*;
- Estrutura *else*;
- Estrutura *else if*;

Instruções condicionais

Exemplo.: Realizar a divisão entre dois números.

```
#include <stdio.h>
main()
{
        float a,b,div;
        scanf("%f %f",&a,&b);
        if ( b == 0 )
        {
                printf("A divisão não é possível!");
        }
        else
        {
                div = a/b;
                printf("A divisão é %f",div);
        }
}
```

Aspectos a reter

- *if(b == 0)*, significa que se o *b* for igual a zero, vai executar o que está imediatamente abaixo entre chavetas.
- *else*, significa o contrário do referido no *if* imediatamente antes, ou seja, quando *b* for diferente de zero.
- Dentro dos *if*'s deve ser sempre usado == em vez de =, pois estamos a comparar dois valores. Se colocarmos = estamos obrigatoriamente a dizer que os valores são iguais, e por isso, o programa selecionará sempre essa restrição.
- Quando temos dois *if*'s e um *else*, este *else* refere-se ao *if* imediatamente antes, ou seja, ao segundo *if*. Para que isto não aconteça, devemos usar uma outra estrutura representada no exemplo que se segue, em que o último *else*, refere ao contrário de tudo o que foi referido antes.

Exemplo.:

```
#include <stdio.h>
main()
{
        int x;
        scanf("%d",&x);

        if ( x > 0 )
                printf("Número positivo.");
        else if ( x < 0 )
                printf("Número negativo");
        else
                printf("O número é zero");
}
```

Nota : Não é necessária a colocação de chavetas, pois dentro das estruturas condicionais só existe uma linha de código.

Instruções de repetição

As instruções de repetição em C, têm uma grande importância na medida em que podemos processar informação usando ciclos, dependendo das condições dadas inicialmente.

Existem três estruturas principais de repetição:

- Estrutura *for*;
- Estrutura *while*;
- Estrutura *do while*;

Exemplo.: Contagem crescente desde 1 até *a* (Usando *for*)

```
#include <stdio.h>
main()
{
        int a,i;
        scanf("%d",&a);

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

Aspectos a reter

- Foi necessário declarar duas variáveis, em que uma é variável de controlo, i, e a outra irá estabelecer um máximo,a.
- Em relação ao *for*, no primeiro argumento, o $i = 1$ indica que o ciclo se inicia com i = 1. No segundo argumento, é dada a condição, ou seja, será realizado o que está dentro do *for*, enquanto a condição for verdadeira. O último argumento indica que em cada ciclo, o i aumentará uma unidade em cada looping.
- Simulação do programa :
  - Considerando a = 3.
  - Entrando no *for*, i = 1, e como i <= a, é mostrado o valor de i, neste caso, 1.
  - Entrando no *for*, i passa a ser 2, devido ao i++, e como i <= a, é mostrado o valor de i, neste caso, 2.
  - Entrando no *for*, i passa a ser 3, devido ao i++, e como i <= a, é mostrado o valor de i, neste caso 3.
  - Entrando no *for*, i passa a ser 4, devido ao i++, e como i já não é menor ou igual a a, não iremos entrar novamente no ciclo.

Exemplo.: Contagem decrescente desde *a* até 1 (Usando *for*)

```
#include <stdio.h>
main()
{
        int a,i;
        scanf("%d",&a);

        for ( i = a ; i >= 1 ; i-- )
        {
                printf("%d\n", i);
        }
}
```

Aspectos a reter

- Foi necessário declarar duas variáveis, em que uma é variável de controlo, i, e a outra irá estabelecer o começo da contagem decrescente, a.
- Em relação ao *for*, no primeiro argumento, o $i = a$ indica que o ciclo se inicia com i = a. No segundo argumento, é dada a condição, ou seja, será realizado o que está dentro do *for*, enquanto a condição for verdadeira. O último argumento indica que em cada ciclo, o i diminuirá uma unidade em cada looping.
- Simulação do programa :
  - Considerando a = 2.
  - Entrando no *for*, i = 2, e como i >= 1, é mostrado o valor de i, neste caso, 2.
  - Entrando no *for*, i passa a ser 1, devido ao i−−, e como i >= 1, é mostrado o valor de i, neste caso, 1.
  - Entrando no *for*, i passa a ser 0, devido ao i−−, e como i já não é maior ou igual a 1, não iremos entrar novamente no ciclo.

Exemplo.: Contagem crescente desde 1 até *a* (Usando *while*)

```
#include <stdio.h>
main()
{
        int a;
        scanf("%d",&a);
        int i = 1;

        while ( i <= a )
        {
                printf("%d\n", i);
                i++;
        }
}
```

Aspectos a reter

- Foi necessário declarar duas variáveis, em que uma é variável de controlo, *i*, e a outra irá estabelecer o começo da contagem crescente, *a*.
- Em relação ao *while*, este é constituído somente por um argumento, que corresponde à condição imposta, ou seja, enquanto a condição for verdadeira, o que está dentro do *while* será executado.
- Simulação do programa :
  - Considerando a = 2.
  - Entrando no *while*, como i <= a, o valor de i será mostrado, e este aumentará para 2, devido ao i++.
  - Entrando no *while*, como i <= a, o valor de i será mostrado, e este aumentará para 3, devido ao i++.
  - Entrando no *while*, como i já não é menor ou igual a 2, então as instruções dentro do *while* já não serão executadas.

Exemplo.: Contagem decrescente desde *a* até 1 (Usando *while*)

```
#include <stdio.h>
main()
{
        int a;
        scanf("%d",&a);
        int i = a;

        while ( i >= 1 )
        {
                printf("%d\n", i);
                i--;
        }
}
```

Aspectos a reter

- Foi necessário declarar duas variáveis, em que uma é variável de controlo, *i*, e a outra irá estabelecer o começo da contagem decrescente, *a*.
- Em relação ao *while*, este é constituído somente por um argumento, que corresponde à condição imposta, ou seja, enquanto a condição for verdadeira, o que está dentro do *while* será executado.
- Simulação do programa :
  - Considerando a = 2.
  - Entrando no *while*, i = 2, como i >= 1, o valor de i será mostrado, e este diminuirá para 1, devido ao i−−.
  - Entrando no *while*, como i >= 1, o valor de i será mostrado, e este diminuirá para 0, devido ao i−−.
  - Entrando no *while*, como i já não é maior ou igual a 1, então as instruções dentro do *while* já não serão executadas.
- **Nota :** *int i = a;* terá de vir imediatamente depois de *scanf("%d",&a);* senão *i* iria receber um valor aleatório que não correspondia ao valor inserido de *a*.

Exemplo.: Dar um valor a x se $x \in [a, b]$ (Usando *do while*)

```
#include <stdio.h>

main()
{
        int a,b;
        int x;

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

        do{
                printf("Insira um número entre %d e %d.\n",a,b);
                scanf("%d",&x);
        }while ( x < a || x > b );
}
```

Aspectos a reter

- Foi necessário declarar duas variáveis de controlo (a,b) que irão estabelecer os limites do intervalo.
- Em relação ao *do while*, este é constituído somente por um argumento, que corresponde à condição imposta, ou seja, enquanto a condição for verdadeira, o que está dentro do *do while* será executado.
- **Nota :** Na primeira entrada no *do while* mesmo que a condição não se verifique, o que está dentro da estrutura será executado. Esta situação por vezes origina erros graves de algoritmo.
- Simulação do programa :
  - Considerando a sequência no *input* : $1 \rightarrow 4 \rightarrow 5 \rightarrow 2$.
  - Intervalo criado : [1,4].
  - Entrando no *do while*, $x = 5$, e como 5 não pertence ao intervalo, entramos novamente no *do while*.
  - Entrando no *do while*, $x = 2$, e como 2 pertence ao intervalo, x passa a ser efetivamente 2.
- **Nota :** Este programa é funcional para $a < b$.

Funções

As funções em C, têm uma grande importância na medida em que podemos "poupar" linhas de código e desenvolver programas visualmente mais simples/intuitivos de processamento mais rápido.

A estrutura de uma função é :

- tipo nome (argumento1, argumento2, ...)
  - Exemplo : int maior (int N1, int N2)

- **Nota :** Quando o tipo da função é *void*, não existe devolução de um valor ao *main*, ou seja, o resultado já é mostrado dentro da função, como podemos ver no exemplo que se segue.

Exemplo.: Dado um x, determinar $f(x)$, com $f(x) = x^2 - 3x + 5$

```
#include <stdio.h>
void function ( int x )
{
        int f;
        f = x*x - 3*x + 5;
        printf("f(%d) = %d",x,f);
}
main()
{
        int a;
        scanf("%d",&a);
        function(a);
}
```

Aspetos a reter

- No cabeçalho da função (void function(int x)), *void* significa que a função não irá devolver um valor, *function* é o nome da função e *int x* indica que a função deve receber um valor inteiro x. Este valor de x virá do *main*, ou seja, receberá o valor de a.
- No *main*, depois de ter sido inserido um valor para a, a linha de código *function(a);* "chama" a função, pelo que o valor de a é "lançado" para a função chamada.
- Como é óbvio, o *printf()* mostrará o valor de f, só depois de este ter recebido o valor de a proveniente do *main*.
- Simulação do programa :
  - Considerando a = 2.
  - A função é chamada ao usar *function(a)*.
  - A função recebe o inteiro a do *main*, passando a designar-se *int x*.
  - Na função *function*, o f é calculado e mostrado.

Exemplo.: Dado um x, determinar $f(x)$, com $f(x) = x^2 - 3x + 5$

```
#include <stdio.h>
int function ( int x )
{
        int f;
        f = x*x - 3*x + 5;
        return f;
}
main()
{
        int a,b;
        scanf("%d",&a);
        b = function(a);
        printf("b(%d) = %d",a,b);
}
```

Aspetos a reter

- No cabeçalho da função (int function(int x)), *int* significa que a função devolve um valor inteiro, *function* é o nome da função e *int x* indica que a função deve receber um valor inteiro x. Este valor de x virá do *main*, ou seja receberá o valor de a.
- O *return f* significa que a função irá devolver ao b no *main* o valor de f, depois de a ter chegado à função.
- No *main*, depois de ter sido inserido um valor para a, a linha de código $b = function(a);$ "chama" a função, pelo que o valor de a é "lançado" para a função chamada.
- Como é óbvio, o *printf()* mostrará o valor de b, só depois de este ter recebido o valor de f da função.
- Simulação do programa :
  - Considerando a = 2.
  - A função é chamada ao usar $b = function(a)$.
  - A função recebe o inteiro a do *main*, passando a designar-se *int x*.
  - Na função *function*, o f é calculado e devolvido ao *main*.
  - Por fim, b recebe o valor devolvido de f e mostra-o.

Outra possível resolução seria :

```
#include <stdio.h>

main()
{
        int a,b;
        scanf("%d",&a);
        b = a*a - 3*a + 5;
        printf("b(%d) = %d",a,b);
}
```

Vetores e Matrizes

As matrizes e vetores em C, têm uma grande importância na medida em que podemos armazenar informação de forma organizada.

Os vetores e matrizes em C têm a seguinte estrutura :

$$\text{Vetor (com } n \text{ elementos)} : (a_0, a_1, \ldots, a_{n-1}) \text{ ou } \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

$$\text{Matriz (com } n \text{ linhas e } m \text{ colunas)} : \begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,m-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,m-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n-1,0} & a_{n-1,1} & \cdots & a_{n-1,m-1} \end{bmatrix}$$

Exemplo.: Declarar e preencher um vetor.

```
#include <stdio.h>
main()
{
        int v[2];
        int i;

        for( i = 0 ; i < 2 ; i++ )
        {
                scanf("%d",&v[i]);
        }
}
```

Aspetos a reter

- *int v[2]* significa que foi declarado um vetor, com dois elementos, que só poderá ser preenchido com números inteiros.
- O *for* é usado para percorrer o vetor, de modo a preenchê-lo.
- O *scanf()* recebe os valores inseridos, armazenando-os num *v[i]*, em que i corresponde à posição ($i \in [0,1] \wedge i \in \mathbb{N}_0$).
- Simulação do programa :
  - Considerando a sequência no *input* : $3 \rightarrow 4$.
  - Entrando no *for*, como i = 0, o valor inserido irá ser armazenado no *v[0]*, ou seja, na posição 0.
  - Entrando no *for*, como i = 1, devido ao i++, o valor inserido irá ser armazenado no *v[1]*, ou seja, na posição 1.
  - Entrando no *for*, como o *i* já não é menor que 2, então o que está dentro do *for* não será executado.
- Vetor construído : $(3, 4)$.
- **Nota :** Para mostrar o vetor, troca-se o *scanf* pelo *printf*.

Exemplo.: Declarar e preencher uma matriz.

```
#include <stdio.h>
main()
{
        int m[2][2];
        int i,j;

        for( i = 0 ; i < 2 ; i++ )
        {
                for( j = 0 ; j < 2 ; j++ )
                {
                        scanf("%d",&m[i][j]);
                }
        }
}
```

Aspetos a reter

- *int m[2][2]* significa que foi declarada uma matriz, com quatro elementos (2×2), que só poderá ser preenchida com números inteiros.
- Os dois *for*'s são usados para percorrer a matriz, de modo a preenchê-la. O primeiro *for*'s é responsável pelas linhas e o segundo pelas colunas.
- O *scanf()* recebe os valores inseridos, armazenando-os num *m[i][j]*, em que i corresponde à posição na linha ($i \in [0,1] \wedge i \in \mathbb{N}_0$) e j à posição na coluna ($j \in [0,1] \wedge j \in \mathbb{N}_0$).
- Esquema da matriz :

$$\begin{bmatrix} m[0][0] & m[0][1] \\ m[1][0] & m[1][1] \end{bmatrix}$$

Aspetos a reter (Continuação)

- Simulação do programa :
  - Considerando a sequência no *input* : $3 \rightarrow 4 \rightarrow 5 \rightarrow 6$.
  - Entrando no primeiro *for*, e consequentemente no segundo, como i = 0 e j = 0, o valor inserido (3) irá ser armazenado no *m[0][0]*.
  - Entrando no segundo *for*, como i = 0 e j = 1, devido ao j++, o valor inserido (4) irá ser armazenado no *m[0][1]*.
  - Entrando no segundo *for*, como o *j* já não é menor que 2, então o que está dentro do *for* não será executado, voltando ao primeiro.
  - Entrando pela segunda vez no primeiro *for*, e consequentemente no segundo, como i = 1, devido ao i++, e j = 0, o valor inserido (5) irá ser armazenado no *m[1][0]*.
  - Entrando no segundo *for*, como i = 1 e j = 1, devido ao j++, o valor inserido (6) irá ser armazenado no *m[1][1]*.
  - Entrando no segundo *for*, como o *j* já não é menor que 2, então o que está dentro do *for* não será executado. Como o *i* também já não é menor que 2, então o que está dentro do primeiro *for* não será executado também.
- **Nota :** Para mostrar a matriz, troca-se o *scanf* pelo *printf*.

Caracteres e Strings

As string em C, têm uma grande importância na medida em que são um tipo de dados definido como um vetor de carateres.

- A estrutura de uma string é :
  - $(c_0, c_1, c_2, \ldots, \text{'\textbackslash 0'})$
- Existem quatro funções para ler e mostrar strings :
  - **scanf** - só lê uma palavra. Não é necessário o operador & na sintaxe do scanf.
  - **printf** - mostra a string.
  - **gets** - funciona como um *scanf()*.
  - **puts** - funciona como um *printf()* + '\n'.

Exemplo.: Escrever um nome e mostrá-lo.

```
#include <stdio.h>
main()
{
        char nome[50];
        char apelido[50];
        puts("Escreva o seu nome:");
        gets(nome);
        puts("Escreva o seu apelido:");
        gets(apelido);
        printf("Nome Completo : %s %s.",nome,apelido);
}
```

Aspetos a reter

- *char nome[50]* significa que foi declarada uma string (vetor de caracteres) com 50 elementos, incluindo o caracter terminador '\0', ou seja, podem ser utilizados no máximo até 49 elementos da string.
- *gets* recebe o nome e o apelido inserido transformando-os em duas strings.

- **Notas :**
  - Os caracteres são símbolos que estão guardados numa "base de dados", entre os quais estão o "espaço" e o *enter*, ou seja, um espaço ou um enter mal inserido irá contar como caracter.
  - Para mais símbolos e outras informações consulte a tabela ASCII.

Exemplo.: Determinar o número de caracteres de uma string.

```
#include <stdio.h>
main()
{
        char s[50];
        int i = 0; int contador = 0;
        puts("Escreva uma palavra:");
        scanf("%s",s);

        while ( s[i] != '\0' )
        {
                contador = contador + 1;
                i++;
        }
        printf("%s tem %d caracteres.",s,contador);
}
```

Aspetos a reter

- Foi necessário declarar uma variável de controlo (i) e um contador, que determinará quantos caracteres existem na string. Como é óbvio, este iniciar-se-á em 0.
- Em relação ao conteúdo que está dentro do *while*, este só será executado enquanto *s[i]* não corresponder ao caracter terminador.
- Simulação do programa :
  - Considerando a palavra "a!Y" no *input*.
  - Com i = 0, *s[0]* = 'a'. Como 'a' é diferente do caracter terminador, entramos no *while*, passando o contador a ser 1 e o valor de i aumenta para 1, devido ao i++.
  - Com i = 1, *s[1]* = '!'. Como '!' é diferente do caracter terminador, entramos no *while*, passando o contador a ser 2 e o valor de i aumenta para 2, devido ao i++.
  - Com i = 2, *s[2]* = 'Y'. Como 'Y' é diferente do caracter terminador, entramos no *while*, passando o contador a ser 3 e o valor de i aumenta para 3, devido ao i++.
  - Com i = 3, *s[3]* = '\0'. Como '\0' é igual ao caracter terminador, o que está dentro do *while* não será executado.

Estas são algumas das funções utilizadas na manipulação de strings. Para ter acesso a mais funções, e à respetiva sintaxe, faça uma breve pesquisa, referenciando a biblioteca <string.h>

Algumas funções utilizadas na manipulação de strings

Funções	**Resultado da função**
strcmp	Compara duas strings
strcpy	Copia uma string para outra
strlen	Devolve o comprimento de uma string
strcat	Concatena duas strings

Nota: Para utilizar as funções acima indicadas é necessário incluir a biblioteca <string.h>.

Ficheiros de Texto

Os ficheiros de texto em C, têm uma grande importância na medida em que podemos armazenar informação de modo organizado em ficheiros de texto (extensão : .txt).

Modos de abertura de ficheiros

Modo	**Significado**
r	Abertura para leitura
w	Abertura para escrita (apaga o texto existente)
a	Abertura para adicionar texto ao existente
r+	Abertura para leitura e escrita
w+	Abertura para leitura e escrita
a+	Abertura para leitura e adicionar texto ao existente

Nota : O ficheiro terá de estar no mesmo diretório do programa.

Exemplo.: Colocar uma palavra num ficheiro de texto.

```
#include <stdio.h>
main()
{
        FILE *file;
        char nome[50];
        file = fopen("ficheiro.txt","w");
        puts("Escreva o seu nome:");
        gets(nome);
        fprintf(file, "%s", nome);
        fclose(file);
}
```

Aspetos a reter

- *FILE *file* significa que foi declarado um ficheiro de texto.
- *file = fopen("ficheiro.txt","w")* significa que o ficheiro foi aberto no modo *w*.
- *fprintf(file, "%s", nome)* é a função que irá colocar no ficheiro o que foi inserido na string *nome*.
- *fclose(file)* faz com que o ficheiro fique fechado em segurança.

Exemplo.: Ler os valores de um ficheiro e mostrá-los.

```
#include <stdio.h>
main()
{
        FILE *file;
        int x;
        file = fopen("ficheiro.txt","r");

        while ( !feof( file ) )
        {
                fscanf(file, "%d", &x);
                printf("%d ", x);
        }
        fclose(file);
}
```

Aspetos a reter

- *file = fopen("ficheiro.txt","r")* significa que o ficheiro foi aberto no modo *r*.
- *while (!feof(file))* significa que enquanto não chegar ao fim do ficheiro, o que está dentro do while será executado.
- *fscanf(file, "%d", &x)* significa que serão lidos os valores do ficheiro, um a um, até chegar ao fim do ficheiro. O *printf("%d ", x)* encarrega-se de mostrá-los no ecrã.
- *fclose(file)* faz com que o ficheiro fique fechado em segurança.

Considerações Finais

Com esta apresentação, espero ter-vos incutido as principais ideias que regem as bases da Linguagem C. Como é óbvio isto não é suficiente para uma compreensão detalhada da mesma, pelo que devem procurar fazer pesquisas sobre novas estruturas, novas funções, novas técnicas, e desta forma abrirem horizontes.

Uma técnica que recomendo, caso exista *bugs*, é a utilização exaustiva de *printf()*'s como detetor de erros nos vossos programas.

Sugiro o *CodeBlocks* como software de programação, pelo seu design intuitivo.

Obrigado,
Emanuel António Rodrigues Camacho