Introdução à Programação

1.Ano LCC-MIERSI
DCC - FCUP

Nelma Moreira

Aula 7

Tipos de inteiros em C

* Os inteiros em C podem ser com sinal ou sem sinal:E cada um pode ter vários tamanhos:

```
int i;
unsigned int u;
```

* E cada um pode ter vários tamanhos:

```
short, long

short int i; long int l;
unsigned int u; unsigned long g; unsigned short int s;
```

* os números que representam tem haver com o número de bits usados para representar um `int` : 16, 32 ou 64. (ver `limits.h`)

Tipo	Valor menor	Valor Maior
	16-bits	
short int	-32768	32767 (2^15 - 1)
unsigned short int	0	65535 (2^16 - 1)
int	-32768	32767
unsigned int	0	65535
long int	-2147483648	2147483647
unsigned long int	0	4294967295
	32-bits	
short int	-32768	32767
unsigned short int	0	65535
int	-2147483648	2147483647 (2^31 - 1)
unsigned int	0	4294967295 (2^32 - 1)
long int	-2147483648	2147483647
unsigned long int	0	4294967295

Caracteres e Código ASCII

```
  0: . . . . . . . . . . . . . . . .
 16: . . . . . . . . . . . . . . . .
 32:   ! " # $ % & ' ( ) * + , - . /
 48: 0 1 2 3 4 5 6 7 8 9 : ; < = > ?
 64: @ A B C D E F G H I J K L M N O
 80: P Q R S T U V W X Y Z [ \ ] ^ _
 96: ` a b c d e f g h i j k l m n o
112: p q r s t u v w x y z { | } ~
128: . . . . . . . . . . . . . . . .
144: . . . . . . . . . . . . . . . .
```

```
#include <stdio.h>
#define LINE 16
int main() {
   int c;
   for(c = 0;c < 256;c++) {
      if (c % LINE == 0) printf ("\n %3d: ",c);
      if(c < 32 || (c > 127 && c <256)) printf(".
");
      else   printf("%c ",c);
   }
   printf("\n");
return 0;
}
```

Tipo `char`

* O tipo `char` representa um caracter e corresponde a um byte (inteiro de 8-bits)
* Pode ser `signed (-128 a 127)` ou `unsigned (0 a 256)`
* Uma constante do tipo `char` é um caracter entre plicas. Ex: `'a', 'A', '!', '0'`
* Um caracter também pode ser especificado por `\nnn` onde `nnn` é representação em octal do seu código

```c
#include <stdio.h>
int main(){
 int x = 99, y = '\120';
 printf("%c tem o codigo
        ASCII %d\n",x,x);
 printf("O codigo ASCII
%d (octal %o) corresponde
a %c\n",y,y,y);
return 0;
}
```

Compilação e Execução:

```
% gcc cod.c
% ./a.out
c tem o codigo ASCII 99
O codigo ASCII 80 (octal
120) corresponde a P
```

Ler e escrever caracteres

* O conversor `%c` permite o `scanf` e o `printf` ler e escrever caracteres:

```
int ch;

scanf("%c", &ch);
printf("%c",ch);
```

* Por exemplo pode-se ler até ao fim da linha:

```
do{
 scanf("%c", &ch);
 } while( ch != '\n');
```

* Mas há maneiras mais simples e rápidas de ler e escrever caracteres.

Função `getchar()`

* A função `getchar()` lê um caracter e retorna um inteiro que é:
  * o código do caracter, ou
  * o valor -1 que corresponde a fim de ficheiro (ou a constante simbólica `EOF`)

```
int ch;
ch = getchar();
while ( getchar() != '\n');
```

Problema 1: Contar a's

```
#include<stdio.h>

int main() {

char c; int i = 0;

while ((c = getchar()) !=
EOF) if (c == 'a') i++;

 printf("\n Foram lidos
%d a\'s\n",i);

return 0;

}
```

Execução:

```
% gcc aa.c -o aa

% ./aa

sagHAGHGHGGAHGghshah
y1ty21wjhaajakajka

Foram lidos 7 a's
```

Contar o número de vogais de uma sequência de caracteres

Algoritmo

* Definir um contador para cada vogal: `na, ne, ni, no e nu`
* Para cada caracter lido determinar se é uma vogal e, se for, incrementar o contador correspondente.
* As vogais minúsculas e maiúsculas têm códigos diferentes e devem ser ambas consideradas e também se podem contar os restantes caracteres

```
#include <stdio.h>
int main() {
char c;
int na = 0, ne = 0, ni = 0,
no = 0, nu = 0,nc = 0;
while((c = getchar()) != EOF)
{
switch (c) {
 case 'a':  case 'A': ++na;
            break;
 case 'e':  case 'E': ++ne;
            break;
 case 'i': case 'I': ++ni;
            break;
 case 'o': case 'O': ++no ;
            break;
 case 'u': case 'U': ++nu;
            break;
 default: ++nc;
 }}
printf("\n A\t E\t I\t O\t U
\t Outros\n\n");
printf ("%d\t%d\t%d\t%d\t%d\t
\n",na,ne,ni,no,nu,nc);

return 0;

}
```

Execução:

```
% vogais < vogais.c
```

```
 A    E   I   O   U  Outros

25   26  14  10  9  405
```

Função `putchar()`

* A função `putchar()` tem como argumento um inteiro (entre 0 e 255) e escreve o caracter cujo código é esse inteiro.

```
for(c = 0;c < 256;c++)
    putchar(c);

putchar('A');

putchar('\n');
```

Problema 3: Substituir letras maiúsculas por minúsculas.

```
char maiuscula(char c) {return (c >= 'A' && c <= 'Z'); }
int main(){
  char c;
  while((c=getchar())!= EOF)
    if(maiuscula(c))
      putchar(c+('a'-'A'));  /* 'a'-'A' = 32 */
    else putchar(c);
return 0;}
```

```
$ gcc maius.c -o maius
$ ./maius
sakU8QWWUINMSA SDHH HDH SJSJKjjjk90986$%%^fqt
saku8qwwuinmsa sdhh hdh sjsjkjjjk90986$%%^fqt
```

Funções para manipular caracteres

Função	Descrição	Exemplo
`isalpha(c)`	Retorna 0 se `c` não é uma letra	`ispalpha('a')` é 1
`islower(c)`	Retorna 0 se `c` não é uma letra minúscula	`islower('A')` é 0
`isupper(c)`	Retorna 0 se `c` não é uma letra maiúscula	`isupper('a')` é 0
`isdigit(c)`	Retorna 0 se `c` não é um algarismo	`isdigit('5')` é 1
`isalnum(c)`	Retorna 0 se `c` não é uma letra nem um algarismo	`ispalnum('?')` é 0
`isspace(c)`	Retorna 0 se `c` não é um dos caracteres brancos (`' '`,`\n`,`\t`,`\r`, `\f`)	`isspace(' ')` é 1

Função	Descrição	Exemplo
`tolower(c)`	Se `c` é uma letra maiúscula, retorna a minúscula são retorna o próprio caracter	`tolower('A')` é `'a'`
`toupper(c)`	Se `c` é uma letra minúscula, retorna a maiúscula correspondente, são retorna o próprio caracter	`toupper('a')` é `'A'`

Acessiveis incluindo o cabeçalho `ctype.h`.

Contar o número de caracteres, linhas e palavras dum ficheiro

Para o algoritmo:

* contador de caracteres, `nc`: para cada caracter lido é incrementado;
* cada linha termina com um caracter `\n`: contar a ocorrência desses caracteres (`nl`).
* Uma palavra é uma sequência de caracteres entre brancos. Vamos usar a função `isspace()`.

* Como contar palavras?
* Considerar dois `estados`: `FORA` e `DENTRO` duma palavra. Quando se está `FORA` e aparece um caracter que não é um branco, passa-se a `DENTRO` e incrementa-se o contador. Se se está `DENTRO`, passa-se a `FORA` quando se encontra um branco. Dois ou mais brancos consecutivos não definem palavras. Contador: `np`

Algoritmo:

* Para cada caracter lido (diferente de -1), incrementar `nc`, se o caracter for `\n`, incrementar `nl`; se for um separador, o estado é `FORA`, senão se o estado fosse `FORA` passa a `DENTRO` e incrementa `np`.

```
#include <stdio.h>
#include<ctype.h>
#define DENTRO 0
#define FORA 1
main() {
  char c;
  int nc = 0, np = 0 ,nl =
0 , estado = FORA;
  while((c=getchar()) != EOF)
{
    ++nc;
    if (c =='\n') ++nl;
    if (isspace(c)) estado =
FORA;
    else if (estado == FORA)
{ estado = DENTRO; ++np; }
  }
  printf("\n Caracteres: %d\n
Palavras: %d \n Linhas: %d
\n",nc, np, nl);
}
```

Execução e comparação com o comando wc do UNIX:

```
%gcc contapal.c -o contapal
%contapal < contapal.c
 Caracteres: 519
 Palavras: 76
 Linhas: 25
%wc < contapal.c
      25      76     519
%
```

Intervalo

5 minutos

Tipos de dados complexos

Considera os seguintes problemas:

* Contar as ocorrências de cada caracter num ficheiro de texto.
* Determinar a maior palavra de um ficheiro de texto.
* Ordenar uma sequência de valores. Exemplo:
  * inteiros por ordem crescente
  * letras do alfabeto
  * palavras por ordem lexicográfica
* Operações sobre conjuntos finitos de inteiros
* Suponha um conjunto de alunos em cada aluno tem um nome e uma nota. Pretende-se saber qual o nome do aluno que teve melhor (ou pior) nota.

* Para cada um dos problemas anteriores não é possível, duma forma geral, representar a informação usando variáveis simples (escalares), isto é, em que se associa um nome a um só valor.

* Os tipos de dados complexos em C, construídos a partir de tipos simples, permitem associar um só nome a um conjunto de elementos, e distinguem-se por:

Variáveis indexadas

* número fixo de elementos do mesmo tipo (estáticas). O índice pode ter uma ou várias dimensões.

```
int a[10];
```

Estruturas

* elementos de tipos diferentes mas de tamanho fixo (estáticas).

```
struct ponto { int x; float y;};
```

Estruturas de dados dinâmicas

* : o número de elementos pode ser dinâmicamente modificado durante a execução.

```
int *a;
```

Variáveis Indexadas Unidimensionais

Permitem manipular um conjunto (sequência) de elementos do mesmo tipo:

a

8	5	6	1	3	9	4

Os elementos são referidos pelo seu índice i (posição):

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

* Em C uma variável indexada ao ser declarada tem de ter:
* um `tipo`, que é o tipo de cada elemento
* um `tamanho`, que é o número de elementos que a variável pode conter e cujo espaço de memória é reservado. Tem de ser uma constante inteira.
* um `nome`

```
int a[7];
```

Reserva memória para 7 inteiros

```
main() {
   int i, a[7];

   a[0] = 8;
   a[1] = 5;
   a[2] = 6;
   a[3] = 1;
   a[4] = 3;
   a[5] = 9;
   a[6] = 4;
   for(i = 0; i < 7; i++)
      printf("a[%d]= %d
           \t",i,a[i]);
   printf("\n");
}
```

Execução:

```
% vi1

a[0]= 8 a[1]= 5 a[2]= 6
a[3]= 1 a[4]= 3 a[5]= 9
a[6]= 4
```

Pode-se usar `#define MAX7` e depois `int a[MAX]`

O que imprime o seguinte programa?

```
#define MAX 10
main() {
   int i,a[MAX],b[MAX];
   a[0] = 1;
   b[MAX-1] = 1;
   for(i = 1;i < MAX;i++)
{
      a[i] = a[i-1] * 2;
      b[MAX-i-1] = a[i];
   }
   for(i = 0;i < MAX;i++)
      printf("%3d %3d %3d
\n",i,a[i],b[i]);
}
```

Execução:

```
% vi2
  0   1 512
  1   2 256
  2   4 128
  3   8  64
  4  16  32
  5  32  16
  6  64   8
  7 128   4
  8 256   2
  9 512   1
```

Inicialização de variáveis indexadas

1. Elemento a elemento.

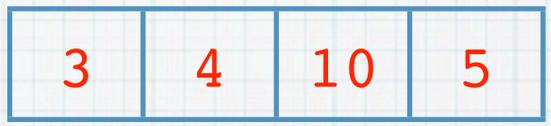
```
int a[4], b[10];
  a[0]=3; a[1]=4;
  a[2]=10; a[3]=5;
  for(i=0;i<10;i++)
     scanf("%d",&b[i]);

#define TAM 10
int s[TAM], i;
  for(i=0;i<TAM; i++)
     s[i]=2+2*i;
```

2. Na declaração das variáveis, com um sinal = seguido duma lista de valores separados por , e entre chavetas

```
int a[4]={3,4,10,5};
```


```
int a[]={3,4,10,5};
```




```
int a[4]={3,4};
```


```
int s[100]={0};
```

as 100 posições são inicializadas em 0.

* O C não verifica (não dá erro sintáctico) se é um índice é maior que o tamanho da variável. E o seu comportamento pode ser imprevisível.

```
int a[10], i;

for (i = 0; i <= 10; i++) a[i] = 0;
```

* Pode entrar em ciclo! Como?
* Se a variável `i` estiver guardada a seguir a `a[9]` e o programa ao executar `a[10]=0`, está realmente a fazer `i=0`.

Taxas de juro

Determinar o valor de 100 euros investidos a diferentes taxas de juro durante um certo período de anos.

* Se fosse só a uma taxa de juro, não era necessário uso de variáveis indexadas.

```
scanf("%d %d",&taxa,&num_anos);
valor = 100;
for(ano = 1; ano<= num_anos; ano++)
 valor += (float) taxa / 100 * valor;
```

* Vamos usar uma variável indexada int `valor[NUM_TAXAS]` que guarda para cada taxa o valor em cada ano.

```c
#define BALANCO_INICIAL 100.0
#define NUM_TAXAS 5
int main() {
  int taxa_inicial, num_anos, i, ano;
  float valor[5];
  printf("Introduzir taxa de juro:");
  scanf("%d",&taxa_inicial);
  printf("Introduzir número de anos:");
  scanf("%d",&num_anos);
  printf("\n Anos");
  for(i = 0; i < NUM_TAXAS; i++  ) {
    printf("%6d%%",taxa_inicial+i);
    valor[i] = BALANCO_INICIAL;
  }
  printf("\n");
  for (ano = 1; ano <= num_anos; ano++) {
    printf("%3d    ",ano);
    for(i = 0; i < NUM_TAXAS; i++  ) {
      valor[i] += (float) (taxa_inicial + i)/100 * valor[i];
      printf("%7.2f",valor[i]);}
    printf("\n");
  }
}
```

Execução:

```
%./taxas
Introduzir taxa de juro: 6
Introduzir número de anos: 5

 Anos      6%      7%      8%      9%     10%
  1      106.00 107.00 108.00 109.00 110.00
  2      112.36 114.49 116.64 118.81 121.00
  3      119.10 122.50 125.97 129.50 133.10
  4      126.25 131.08 136.05 141.16 146.41
  5      133.82 140.26 146.93 153.86 161.05
```

Dado um conjunto de 30 valores de 1 a 10 determinar a média, qual a frequência de cada valor e o histograma.

1,2,6,4,8,5,9,7,8,1 0,1,6,2,3,3,5,8,4,6,5,4,3,6,7,8,4,4,4,4,5

ALGORITMO

1. Supomos que os valores estão numa variável indexada `valores` e na variável indexada `frequencia` irão ser contadas as ocorrências de cada classificação: a `i`-ésima posição correspondendo ao número de ocorrências do valor `i`.

   Por exemplo, se `frequencia[4]` for `7`, significa que o valor 4 ocorreu 7 vezes.

2. Na variável inteira `soma` irá ser calculada a soma dos valores.

3. Para `r = 0...30`:

    `valores[r]` será adicionado à variável soma e o contador correspondente (`frequencia[valores[r]]`) será incrementado.

```c
#include<stdio.h>
#define TAM 30
#define FREQ 11
main() {
  int valores[TAM] =
{1,2,6,4,8,5,9,7,8,10,1,6,2,3,3,5,8,4,6,5,
                4,3,6,7,8,4,4,4,4,5};
  int frequencia[FREQ] = {0};
  int r,c, soma = 0;
  for(r = 0; r < TAM; r++){
    soma += valores[r];
    ++frequencia[valores[r]];
  }
  printf("Valores Frequencia  Histograma\n");
  for(c = 1; c < FREQ; c++) {
    printf("%4d%8d\t\t",c,frequencia[c]);
    for(r = 0; r < frequencia[c]; r++)
              printf("*");printf("\n");
  }
  printf("Media=%3.1f\n",(float)soma/TAM);
}
```

Compilacão e Execução:

```
%gcc freq.c -o freq
% freq
Valores Frequencia  Histograma
    1         2          **
    2         2          **
    3         3          ***
    4         7          *******
    5         4          ****
    6         4          ****
    7         2          **
    8         4          ****
    9         1          *
   10         1          *
Media=5.1
```