Introdução à Programação

1.Ano LCC-MIERSI
DCC - FCUP

Nelma Moreira

Aula 13

Estruturas

* Tipo complexo de dados que agrupa dados de tipos diferentes (simples ou complexos)
* Adequados para manipular diversas bases de dados:
  * lista telefónica em que cada elemento é constituído por: nome, morada, telefone, etc
  * biblioteca, em que para cada livro é conhecido o título, autor, ano de edição, editora, tema, isbn, etc.

* Para criar uma estrutura em C:
  * Definir um novo tipo de estrutura
  * Declarar variáveis do novo tipo

Definição duma estrutura

```
struct <tipo_de_estrutura>
{
 <tipo_de_elemento> <nome_de_elemento>;
 <tipo_de_elemento> <nome_de_elemento>;
          ...
};
```

* Uma estrutura é um conjunto de variáveis de quaisquer tipos.

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

Declaração de variávies do tipo duma estrutura

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

ou depois do tipo estar definido, como habitualmente,

```
struct ponto pontos[20], ponto_max, ponto_min;
```

As variáveis têm tipo `struct ponto` e não do tipo `ponto`

Referência a elementos duma estrutura

* Os elementos (campos) duma estrutura são referenciados com o operador .

```
<nome_da_variavel>.<nome_de_elemento>
```

* Por exemplo:

```
pp.x=5;
scanf("%d",&pp.y);
printf("%d %d", pp.x,pp.y);
```

Estruturas encaixadas

* Os elementos das estruturas podem ser estruturas.
* Por exemplo, pode-se definir um `rectângulo` usando a estrutura `struct ponto`, considerando o ponto superior direito e o ponto inferior esquerdo:

```
struct rectangulo {
        struct ponto  pie;
        struct ponto  psd;
};
```

* Se declararmos a variável

```
struct rectangulo caixa;
```

* podemos referir-nos à coordenada `x` do ponto `pie` de caixa, por

```
caixa.pie.x
```

typedef

* É um comando do C que permite dar nomes diferentes a tipos já existentes:

```
typedef int inteiro;
inteiro n;
```

* Pode-se usar o `typedef` para dar nomes a estruturas.

```
typedef struct ponto ponto _p;
ponto _p pontos[20]
```

* ou, ainda definir simultaneamente a estrutura:

```
typedef  struct ponto {
         int x;
         int y;
} ponto_p;
```

Operações em estruturas

* As únicas operações em estruturas são as de acesso a elementos (o operador .) e as de atribuição ou cópia.
* A atribuição de estruturas permite que o contéudo duma estrutura seja copiado para a outra estrutura.

```
#include<stdio.h>
int main(){
  struct exemplo {
    int a;
    float b;
    char c;
  };
  struct exemplo ex1, ex2;
  ex1.a=2;
  ex1.b=1.0/2.0;
  ex1.c='g';
  ex2=ex1;
  printf("a =%3d, b=%4.3f,  c=%c\n", ex2.a, ex2.b, ex2.c);
}
```

Inicialização

- No exemplo anterior, a variável `ex1` podia ter sido inicializada por

```
struct exemplo ex1={ 2, 0.5,'g'};
```

- O endereço duma estrutura pode ser obtido com o operador `&`

Estruturas como argumentos e resultados de funções

* A atribuição de estruturas inclui a passagem de argumentos para funções e os valores retornados de funções.

```
struct ponto cria_pt (int x,int y) {
    struct ponto tmp;
    tmp.x = x;
    tmp.y = y;
 return tmp;
}
```

* A função seguinte tem como argumentos duas estruturas, uma tipo `struct ponto` outra tipo `rectangulo`, e retorna `1` ou `0` consoante o ponto está ou não no interior do rectângulo.

```
int dentrorect (struct ponto p, struct rectangulo r) {
    return p.x >= r.pie.x && p.x < r.psd.x &&
                p.y>=r.pie.y && p.y < r.psd.y;
}
```

* Chamada:

```
i=dentrorect(pp,caixa);
```

* As estruturas são passadas por valor.

Variáveis indexadas de estruturas

* Definem-se como para qualquer outro tipo:

```
struct mes {
    char nome[];
    int dias;
} ano[]= {"",0,"Janeiro",31,"Fevereiro",28,
"Março",31,"Abril",30,"Maio",31,
"Junho",30,"Julho",31,"Agosto",31,
"Setembro",30,"Outubro",31,
"Novembro",30,"Dezembro",31};


printf("%s %d\n", ano[1].nome, ano[1].dias);
```

Simulação de um baralho de cartas

* A estrutura seguinte permite representar uma carta de jogar, por um par de strings (valor,naipe).

* Podemos dar apenas um nome:

```
struct carta {
   char valor[MAXN];
   char naipe[MAXN];
};
typedef struct carta Carta;
```

* Um baralho será representado por:

```
Carta baralho[52];
```

* Os possiveis valores e naipes de cada carta são guardados em variáveis globais:

```
char valores[][MAXN]= {"As","Duque",
"Terno","Quadra","Quina","Sena",
"Sete","Oito","Nove","Dez","Valete",
"Dama","Rei"};
```

```
char naipes[][MAXN]={"Copas","Espadas",
"Paus","Ouros"};
```

Algoritmo

* colocar as 52 cartas no baralho por ordem de naipes e para cada naipe de ás a rei
* baralhar as cartas usando a função `rand()`
* imprimir o baralho resultante

```
include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#define MAXN 10
struct carta {
  char valor[MAXN];
  char naipe[MAXN];
};

typedef struct carta Carta;

char valores[][MAXN]
={"As","Duque","Terno","Quadr
a","Quina",Sena","Sete","Oito
", "Nove","Dez", "Valete",
"Dama", "Rei"};
char naipes[][MAXN]
={"Copas","Espadas","Paus","O
uros"};

void preenche(Carta []);
void distribui(Carta []);
void baralha(Carta []);
```

```
int main(){
 Carta baralho[52];
 srand(time(NULL));
 preenche(baralho);
 baralha(baralho);
 distribui(baralho);
 return 0;
}

void preenche( Carta
baralho[]) {
  int i;
  for(i=0;i<52;i++) {

strcpy(baralho[i].valor,valor
es[i%13]);

strcpy(baralho[i].naipe,naipe
s[i/13]);
  }
}
```

```
void baralha(Carta baralho[]) {
  int i,j;
  Carta temp;
  for(i=0;i<52;i++) {
    j = rand() %52;
    temp = baralho[j];
    baralho[j] = baralho[i];
    baralho[i] = temp;
  }
}

void distribui(Carta baralho[]) {
  int i;
  for(i=0;i<52;i++)
    printf("%7s de %-7s%c",baralho[i].valor,baralho[i].naipe,
           (i+1)%3? '\t': '\n');
}
```

Execução

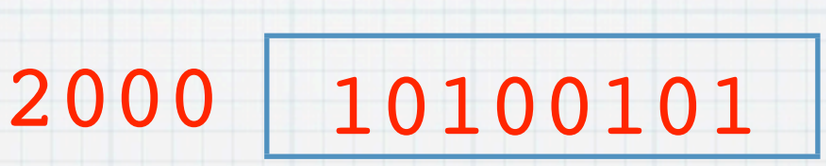
```
%baralha

Rei de Copas
Nove de Espadas
Sete de Copas
Oito de Copas
Dama de Espadas
Sete de Espadas
As de Espadas
Valete de Copas
Sena de Paus
Dez de Ouros
Duque de Ouros
Quina de Ouros
As de Copas
Valete de Ouros
Duque de Copas
Dez de Espadas
Dama de Copas
Nove de Copas
As de Paus
Quadra de Paus
Valete de Espadas
Sete de Paus
Nove de Ouros
Terno de Espadas
Quina de Copas
Oito de Espadas
Dez de Copas
Dez de Paus
Oito de Ouros
Rei de Paus
Terno de Ouros
Nove de Paus
Sena de Ouros
Terno de Paus
Quadra de Espadas
Sena de Espadas
Dama de Ouros
Quadra de Ouros
Sena de Copas
Valete de Paus
Duque de Paus
As de Ouros
Quadra de Copas
Duque de Espadas
Rei de Ouros
Terno de Copas
Oito de Paus
Sete de Ouros
Dama de Paus
Quina de Paus
Quina de Espadas
Rei de Espadas
```

Intervalo

5 minutos

Posições de memória e variáveis

* Cada posição de memória (constituída por um ou mais bytes) tem o endereço e o conteúdo



* Em C a cada nome de variável `x` está associada uma posição de memória (um ou mais bytes). O endereço de `x` é o endereço do primeiro byte.

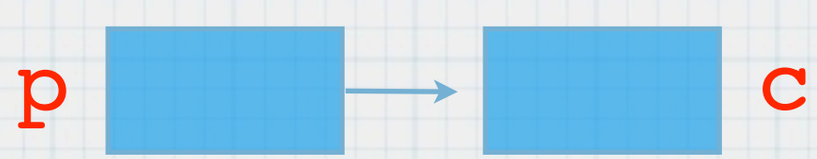
Apontadores

* Se `c` é uma variável o operador `&` dá o endereço dessa variável. Assim

```
p=&c;
```

* atribuí a `p` o endereço de `c`. Diz-se que `p` aponta ou é um **apontador** para `c`.
* Por outro lado se `p` é um apontador, o operador `*` dá o seu conteúdo

```
c=*p;
```



* Para se declarar um apontador, usa-se

```
tipo *nome_da_variável
```

* Por exemplo,

```
int *p;
```

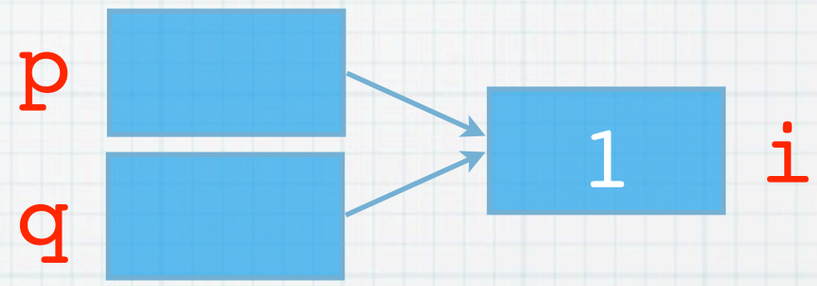
```c
#include  <stdio.h>
int main(){
    int *ip, i, z[10];
    char *clist, a[5];
    ip = &z[0];
    *ip = 3;
    ip = &i;
    printf("%d %d %d\n", i, z[0], *ip);
    clist = a;
    a[0] = 'a';
    a[1] = 0;
    puts(a);
    return 0;
}
```

A variável `ip` fica com o endereço do primeiro elemento da variáel indexada `z`. Todos os valores inteiros imprimidos são iguais.

Apontadores e atribuição

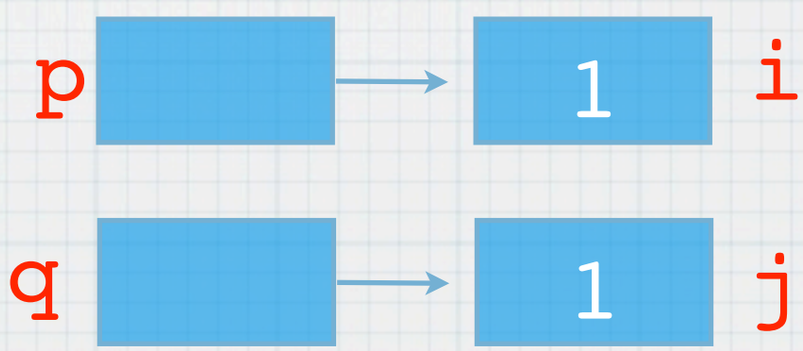
* Podemos copiar apontadores do mesmo tipo usando uma atribuição:

```
int i, j, *p, *q;
p = &i;
p = q;
*q = 1;

printf("%d %d %d", *p, *q, i);
```



* Nota que é diferente fazer

```
p = &i;
q = &j;
i = 1 ;
*q = *p;
```



Apontadores e Funções

* Numa chamada a uma função, das variáveis que são argumentos da função apenas o seu valor é passado para a função.

* Para que uma variável, passada como argumento, seja modificada pela função é necessário passar o seu endereço (o que acontece com as variáveis indexadas).

* Por exemplo, para trocar o valor de duas variáveis inteiras não é possível usar a função seguinte:

```
void troca(int a,int b) {
    int aux;
    aux=a;
    a=b;
    b=aux;
}
```

* A chamada com `troca(x,y)` não altera o valor de `x` e de `y`.

* Por outro lado, a chamada `trocap(&x,&y)` permite a alteração dos valoes das variáveis, se os argumentos, na definição da função forem definidos como apontadores:

```
void trocap(int *ap,int *bp) {
    int aux;
    aux = *ap;
    *ap = *bp;
    *bp = aux;
}
```

* Neste caso o protótipo podia ser

```
void trocap(int *,int *);
```

const para proteger varáveis

* Caso não se queira que modificar uma variável cujo apontador é passado como argumento pode-se usar o qualificador const.

```
void f(const int *p){
*p=0;
}
```

* dá erro de compilação!

Apontadores e variáveis indexadas

* Podemos associar um apontador a um elemento duma variável indexada

```
int a[10], *p;
p = &a[0]
*p = 2;
```

* Mas neste caso podemos incrementar/decrementar `p` e aceder a outros elementos da variável.
* Na realidade, `a` também é um apontador.

* `*a` é o mesmo que `a[0]`
* Se `a[i]=p` então `p+j` aponta para `a[i+j]`

```
p = &a[2]
q = p+3
p += 6
p -= 6
*a = 5
```

* O operador `++` tem precedência sobre `*`

  `*p++ = j;` equivale a `*(p++) = j;`
* Somar todos os elementos de `a`

```
int a[10], s=0, *p;

for(p=a; p < a+10; p++) sum +=*p;
```