

Funções (Retornando arrays)

- ▶ O valor retornado por uma função não pode ser um array.
- ▶ A linguagem C não suporta a atribuição de um array para outro. Por esse motivo, não se pode ter um array como retorno de uma função (diretamente).
- ▶ Porém, C permite a atribuição entre estruturas. Isto faz com que os conteúdos das variáveis contidas dentro de uma estrutura sejam copiados para outra. Desse modo, é possível retornar um array desde que ele esteja dentro de uma estrutura.

Funções (Retornando arrays)



É possível retornar um array indiretamente, desde que ele faça parte de uma estrutura.

```
01  #include <stdio.h>
02  #include <stdlib.h>
03
04  struct vetor{
05      int v[5];
06  };
07
08  struct vetor retorna_array(){
09      struct vetor v = {1,2,3,4,5};
10      return v;
11  }
12
13  int main(){
14      int i;
15      struct vetor vet = retorna_array();
16      for (i=0; i<5; i++)
17          printf("Valores: %d \n",vet.v[i]);
18      system("pause");
19      return 0;
20  }
```

Passagem de arrays como parâmetros

- ▶ Para utilizar arrays como parâmetros de funções, alguns cuidados simples são necessários. Além do parâmetro do array que será utilizado na função, é necessário declarar um segundo parâmetro (em geral, uma variável inteira) para passar para a função o tamanho do array separadamente.
- ▶ Arrays são sempre passados por referência para uma função.
- ▶ Quando passamos um array por parâmetro, independentemente do seu tipo, o que é de fato passado para a função é o endereço do primeiro elemento do array. Exs:
 - ▶ `void imprime (int *m, int n);`
 - ▶ `void imprime (int m[], int n);`

Passagem de arrays como parâmetros

Exemplo: passagem de array como parâmetro

```
01  #include <stdio.h>
02  #include <stdlib.h>
03
04  void imprime (int *n, int m){
05      int i;
06      for (i=0; i<m;i++)
07          printf("%d \n", n[i]);
08  }
09
10  int main(){
11      int v[5] = {1,2,3,4,5};
12      imprime(v,5);
13      system("pause");
14      return 0;
15  }
```

Passagem de estruturas como parâmetros

- ▶ Uma estrutura pode ser passada como parâmetro para uma função de duas formas distintas:
 - Toda a estrutura;
 - Apenas determinados campos da estrutura.
- ▶ As regras de passagem de uma estrutura como parâmetro para uma função são as mesmas usadas para passar uma união ou uma enumeração como parâmetro. As formas são:
 - Passagem de uma estrutura por valor
 - Passagem de uma estrutura por referência
 - Passagem de um campo da estrutura por valor
 - Passagem de um campo da estrutura por referência

Passagem de uma estrutura por valor

- ▶ Para passar uma estrutura como parâmetro de uma função, basta declarar na lista de parâmetros um parâmetro com o mesmo tipo da estrutura.
- ▶ Dessa forma, teremos acesso a todos os campos da estrutura dentro da função. Ex:

```
struct ponto {  
    int x, y;  
};  
void imprime(struct ponto p){  
    printf("x = %d\n",p.x);  
    printf("y = %d\n",p.y);  
}  
main(){  
    struct ponto p1 = {10,20};  
    imprime(p1);  
}
```

Passagem de uma estrutura por referência

- ▶ Para passar uma estrutura como parâmetro por referência para uma função, usa-se o operador “*” na frente do nome da estrutura durante a declaração da função;
- ▶ Alguns cuidados devem ser tomados ao acessar os campos dentro da função. Ex:
 - 1) Utilizar o operador “*” na frente do nome da variável que representa a estrutura.
 - 2) Colocar o operador “*” e o nome da variável entre parênteses ().
 - 3) Por fim, acessar o campo da estrutura utilizando o operador ponto “.”.
- ▶ O exemplo a seguir mostra como os campos de uma estrutura passada por referência devem ser acessados.

Passagem de uma estrutura por referência – Exemplo:

Exemplo: estrutura passada por referência

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  struct ponto {
04      int x, y;
05  };
06  void atribui(struct ponto *p){
07      (*p).x = 10;
08      (*p).y = 20;
09  }
10  int main(){
11      struct ponto p1;
12      atribui(&p1);
13      printf("x = %d\n",p1.x);
14      printf("y = %d\n",p1.y);
15      system("pause");
16      return 0;
17  }
```


Operador seta



- ▶ O operador seta (->) substitui o uso conjunto dos operadores “*” e “.” no acesso ao campo de uma estrutura passada por referência para uma função.
- ▶ O operador seta (->) é utilizado quando uma referência para uma estrutura (struct) é passada para uma função. Ele permite acessar o valor do campo da estrutura fora da função sem utilizar o operador “*”.

Exemplo: passagem de estrutura por referência

	Sem operador seta	Com operador seta
01	struct ponto {	struct ponto {
02	int x, y;	int x, y;
03	};	};
04		
05	void atribui(struct ponto *p){	void atribui(struct ponto *p){
06	(*p).x = 10;	p->x = 10;
07	(*p).y = 20;	p->y = 20;
08	}	}

Recursividade

- ▶ Quando uma função chama ela mesma em seu bloco de código isso é chamado de recursividade

```
4  
5   
6 |  
7 |  
8 |  
9 |  
10 |  
11 |  
12   
13 |  
14 |  
15 |  
16 |  
17 |  
18 |
```

```
int fatorial_recursivo (int n)  
{  
    if(n==1)  
        return n;  
    return fatorial_recursivo(n-1) * n;  
}  
  
int fatorial (int n)  
{  
    int i;  
    i = n - 1;  
    for (i; i!=1; i--)  
        n = n * i;  
    return n;  
}
```

Cuidados na implementação da Recursividade

- ▶ Durante a implementação de uma função recursiva temos de ter em mente duas coisas: o critério de parada e o parâmetro da chamada recursiva:
 - Critério de parada: determina quando a função deve parar de chamar a si mesma. Se ele não existir, a função continuará executando até esgotar a memória do computador. No cálculo de fatorial, o critério de parada ocorre quando tentamos calcular o fatorial de zero: $0! = 1$.

Cuidados na implementação da Recursividade

- Parâmetro da chamada recursiva: quando chamamos a função dentro dela mesma, devemos sempre mudar o valor do parâmetro passado, de forma que a recursão chegue a um término. Se o valor do parâmetro for sempre o mesmo, a função continuará executando até esgotar a memória do computador. No cálculo de fatorial, a mudança no parâmetro da chamada recursiva ocorre quando definimos o fatorial de N em termos no fatorial de $(N - 1)$: $N! = N * (N - 1)!$.
- ▶ Cuidado:
 - Algoritmos recursivos tendem a necessitar de mais tempo e/ou espaço do que algoritmos iterativos!

Fim