

Função com ponteiros

Material de diversas fontes

Prof. Jaime Gross Garcia, Me



1

Endereços

A memória RAM de qualquer computador é uma sequência de bytes. Cada byte armazena um de 256 possíveis valores. Os bytes são numerados sequencialmente e o número de um byte é o seu endereço (= address).

Cada objeto na memória do computador ocupa um certo número de bytes consecutivos. Um char ocupa 1 byte. Um int ocupa 4 bytes e um double ocupa 8 bytes em muitos computadores. O número exato de bytes de um objeto é dado pelo operador sizeof: a expressão sizeof(int), por exemplo, dá o número de bytes de um int no seu computador.



2

Endereços

Cada objeto na memória tem um endereço. Na maioria dos computadores, o endereço de um objeto é o endereço do seu primeiro byte. Por exemplo, depois das declarações

```
char c;
```

```
int i;
```

```
int ponto;
```

os endereços das variáveis poderiam ser os seguintes

```
c      89421
```

```
i      89422
```

```
ponto  89426
```



3

Operadores para Ponteiros

- ❑ Para declararmos um ponteiro, basta utilizar o operador *(asterisco) antes do nome da variável.

- ❑ Exemplo:

```
int *p;
```

- ❑ Ponteiros são tipados, ou seja, devem ter seu tipo declarado e somente podem apontar para variáveis do mesmo tipo.



4

Operadores para Ponteiros

- ❑ Para trabalharmos com ponteiros, C disponibiliza os seguintes operadores:
 - & - Fornece o endereço de memória onde está armazenado uma variável. Lê-se "*o endereço de*".
 - * - Valor armazenado na variável referenciada por um ponteiro. Lê-se "*o valor apontado por*".



5

Definições

Variáveis : endereçam uma posição de memória que contem um determinado valor dependendo do seu tipo (char, int, float, double etc.)

```
main()
{
    long a=5;
    char ch="x";
}
```

	endereço	valor	
a →	0x0100	0x00	} 5
	0x0101	0x00	
	0x0102	0x00	
	0x0103	0x05	
ch →	0x0104	0x78	} 'x'



6

Definições

Ponteiros: são variáveis cujo conteúdo é um endereço de memória.

- Assim, um ponteiro endereça uma posição de memória que contém valores que são na verdade endereços para outras posições de memória.

```
main()
{
    long a=5;
    char ch="x";
    long *aPtr;
    aPtr = &a;
}
```

	endereço	valor	
a	0x0100	0x00	5
	0x0101	0x00	
	0x0102	0x00	
	0x0103	0x05	
ch	0x0104	0x78	'x'
aPtr	0x0105	0x00	
	0x0106	0x00	0x00000100
	0x0107	0x01	
	0x0108	0x00	



7

Alguns exemplos... (1)

```
#include <stdio.h>
main ()
{
    int num,valor;
    int *p;
    num=55;
    p=&num; /* Pega o endereço de num */
    valor=*p; /* Valor é igualado a num de uma maneira indireta */
    printf ("%d\n",valor);
    printf ("Endereço para onde o ponteiro aponta: %p\n",p);
    printf ("Valor da variavel apontada: %d\n",*p);
}
```

```
55
Endereço para onde o ponteiro aponta: 00000000023FD70
Valor da variavel apontada: 55
```

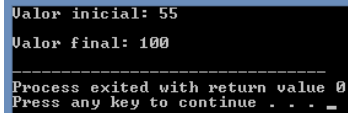
```
Process exited with return value 0
Press any key to continue . . .
```



8

Alguns exemplos... (2)

```
#include <stdio.h>
main ()
{
    int num,*p;
    num=55;
    p=&num; /* Pega o endereco de num */
    printf ("Valor inicial: %d\n",num);
    *p=100; /* Muda o valor de num de uma maneira indireta */
    printf ("\nValor final: %d\n",num);
}
```



```
Valor inicial: 55
Valor final: 100
-----
Process exited with return value 0
Press any key to continue . . . _
```



9

Operadores para Ponteiros

☐ Igualando ponteiros:

```
int *p1, *p2;
p1=p2;
```

- Repare que estamos fazendo com que p1 aponte para o mesmo lugar que p2.

☐ Fazendo com que a variável apontada por p1 tenha o mesmo conteúdo da variável apontada por p2

```
*p1=*p2;
```



10

Alguns exemplos... (3)

```
#include <stdio.h>
main ()
{
    int num,*p1, *p2;
    num=55;
    p1=&num;    /* Pega o endereço de num */
    p2=p1;      /* p2 passa a apontar para o mesmo endereço
                  apontado por p1 */
    printf("Conteudo de p1: %i\n",p1);
    printf("Valor apontado por p1: %i\n", *p1);
    printf("Conteudo de p2:%i \n",p2);
    printf("Valor apontado por p2: %i\n",*p2);
}
```

```
Conteudo de p1: 2358636
Valor apontado por p1: 55
Conteudo de p2:2358636
Valor apontado por p2: 55

-----
Process exited with return value 0
Press any key to continue . . . _
```



11

Operadores para Ponteiros

Tipo	Num de bits	Intervalo	
		Inicio	Fim
char	8	-128	127
unsigned char	8	0	255
signed char	8	-128	127
int	16	-32.768	32.767
unsigned int	16	0	65.535
signed int	16	-32.768	32.767
short int	16	-32.768	32.767
unsigned short int	16	0	65.535
signed short int	16	-32.768	32.767
long int	32	-2.147.483.648	2.147.483.647
signed long int	32	-2.147.483.648	2.147.483.647
unsigned long int	32	0	4.294.967.295
float	32	3,4E-38	3,4E+38
double	64	1,7E-308	1,7E+308
long double	80	3,4E-4932	3,4E+4932



12

Alguns exemplos... (5)

```
#include <stdio.h>
main ()
{
    float num;
    float *p;
    num=55;
    p=&num;
    printf("Conteúdo de p: %f \n",p);
    printf("Valor apontado por p: %f \n", *p);
    printf("Conteúdo de p incrementado:%f \n ", ++p);
    printf("Valor apontado por p incrementado:%f \n ", *p);
}
```

```
Conteúdo de p: 0.000000
Valor apontado por p: 55.000000
Conteúdo de p incrementado:0.000000
Valor apontado por p incrementado:0.000000

-----
Process exited with return value 0
Press any key to continue . . . _
```



13

Alguns exemplos... (6)

```
#include <stdio.h>
main ()
{
    float num;
    float *p;
    num=55;
    p=&num;
    printf("Conteúdo de p: %f \n",p);
    printf("Valor apontado por p: %f \n", *p);
    printf("Conteúdo de p incrementado:%f \n ", ++(*p));
    printf("Valor apontado por p incrementado:%f \n ", *p);
}
```

```
Conteúdo de p: 0.000000
Valor apontado por p: 55.000000
Conteúdo de p incrementado:56.000000
Valor apontado por p incrementado:56.000000

-----
Process exited with return value 0
Press any key to continue . . .
```



14

Vetores como ponteiros

- ❑ O C enxerga vetores como ponteiros
- ❑ Quando declaramos um vetor, o C aloca memória para todas as posições necessárias conforme seu tipo:
 - `int vet[10];`
- ❑ O nome do vetor pode ser atribuído a um ponteiro. Neste caso o ponteiro irá endereçar a posição 0 do vetor:
 - `int *p; p=veter;` ou
 - `int *p; p=&vet[0];`



15

Alguns exemplos... (7)

```
#include <stdio.h>
main ()
{
    int vet [4];
    int *p;
    int count, i;
    p=vet;
    for (count=0;count<4;count++)
    {
        *p=0;
        p++;
    }

    for (i=0;i<4;i++)
        printf(" %i - ", vet[i]);
}
```

```
0 - 0 - 0 - 0 -
-----
Process exited with return value 0
Press any key to continue . . . -
```



16

Ou seja....

- Tipos de passagens de parâmetros
 - Por referência: Os valores das variáveis externas não são passados para a função, mas sim os seus endereços. Usamos os caracteres **&** que indica o endereço e ***** que indica o conteúdo do apontador.



17

- Passagem de parâmetros por referência
-

```
// declarando a função
void troca(int *a, int *b){
    int temp;
    temp=*a;
    *a=*b;
    *b=temp;
}
// iniciando o programa
main()
{
    int a=2,b=3;
    printf("Antes de chamar a funcao : \na=%d\n b=%d\n",a,b);
    troca(&a,&b);
    printf("Depois de chamar a funcao: \na=%d\n b=%d\n",a,b);
    getch();
}
```



18

Vetores como ponteiros

- ❑ **Importante:** um ponteiro é uma variável, mas o nome de um vetor não é uma variável
- ❑ Isto significa, que não se consegue alterar o endereço que é apontado pelo "nome do vetor"
- ❑ Diz-se que um vetor é um ponteiro constante!
- ❑ Condições inválidas:
`int vet[10], *p;
vet++;
vet = p;`



19

Ponteiros como vetores

- ❑ Quando um ponteiro está endereçando um vetor, podemos utilizar a indexação também com os ponteiros:
- ❑ Exemplo:
`int matrxx [10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
int *p;
p = matrxx;
Printf("O terceiro elemento do vetor e: %d ", p[2]);`
- ❑ Neste caso `p[2]` equivale a `*(p+2)`



20

Porque inicializar ponteiros?

❑ Observe o código:

```
main() /* Errado - Nao Execute */
{
    int x,*p;
    x=13;
    *p=x; //posição de memória de p é indefinida!
}
```

- ### ❑ A não inicialização de ponteiros pode fazer com que ele esteja alocando um espaço de memória utilizado, por exemplo, pelo Sistema Operacional.



21

Porque inicializar ponteiros?

- ### ❑ No caso de vetores, é necessário sempre alocar a memória necessária para compor as posições do vetor.
- ### ❑ O exemplo abaixo apresenta um programa que compila, porém poderá ocasionar sérios problemas na execução. Como por exemplo utilizar um espaço de memória alocado para outra aplicação.

```
main() {
    char *pc; char str[] = "Uma string";
    strcpy(pc, str); // pc indefinido
}
```



22

Alocação dinâmica de memória

- ❑ Durante a execução de um programa é possível alocar uma certa quantidade de memória para conter dados do programa
- ❑ A função **malloc (n)** aloca dinamicamente n bytes e devolve um ponteiro para o início da memória alocada
- ❑ A função **free(p)** libera a região de memória apontada por p. O tamanho liberado está implícito, isto é, é igual ao que foi alocado anteriormente por malloc.



23

Alocação dinâmica de memória

- ❑ Os comandos abaixo alocam dinamicamente um inteiro e depois o liberam:

```
#include <stdlib.h>
int *pi;
pi = (int *) malloc (sizeof(int));
...
...
free(pi);
```
- ❑ A função malloc não tem um tipo específico. Assim, (int *) converte seu valor em ponteiro para inteiro. Como não sabemos necessariamente o comprimento de um inteiro (2 ou 4 bytes dependendo do compilador), usamos como parâmetro a função sizeof(int).



24

Alocação dinâmica de vetores

```
#include <stdlib.h>
main()
{
    int *v, i, n;
    scanf("%d", &n); // le n
    v = (int *) malloc(n*sizeof(int)); //aloca n
        elementos para v

    // zera o vetor v com n elementos
    for (i = 0; i < n; i++)
        v[i] = 0;

    ...
    free(v); // libera os n elementos de v
}
```



25

```
// Ordenar de ordem crescente e decrescente 3 números com ponteiros

#include <stdio.h>

int main(void)
{
    int a, b, c, *pa = &a, *pb = &b, *pc = &c, *menor = pa, *maior = pa, *medio = pa;

    printf("Entre com 3 numeros, separados por espaços: ");
    scanf(" %d %d %d", pa, pb, pc);

    if(*pb > *maior)
        maior = pb;
    if(*pc > *maior)
        maior = pc;

    if(*pb < *menor)
        menor = pb;
    if(*pc < *menor)
        menor = pc;

    if(maior != pa && menor != pa)
        medio = pa;
    else if(maior != pb && menor != pb)
        medio = pb;
    else if(maior != pc && menor != pc)
        medio = pc;

    printf("Em ordem crescente: %d %d %d\n", *menor, *medio, *maior);
    printf("Em ordem decrescente: %d %d %d\n", *maior, *medio, *menor);

    return 0;
}
```

26

Fazer um programa em C++ que chama uma função por referência (com ponteiros) para reajustar o preço em 20% e calcular o valor do reajuste.

Solicitar o preço e apresentar o novo preço e o valor do reajuste.



27

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

void reajusta20( float *, float *); /* protótipo */

int main()
{
    float val_preco, val_reaj;

    do
    {
        printf("\nInsira o preco atual: ");
        scanf("%f", &val_preco);
        reajusta20(&val_preco, &val_reaj); /* Enviando endereços */
        printf("\nO preco novo e %.2f\n", val_preco);
        printf("O aumento foi de %.2f\n", val_reaj);
    } while( val_preco != 0.0);
    system("PAUSE");
    return 0;
}

/* reajusta20() */
/* Reajusta o preço em 20% */
void reajusta20(float *preco, float *reajuste) /* Recebendo ponteiros */
{
    *reajuste = *preco * 0.2;
    *preco *= 1.2;
}
```



28