Programação II

Josimar Viana

Funções

Uma função no C tem a seguinte forma geral:

tipo_de_retorno nome_da_função (declaração_de_parâmetros)

{

corpo_da_função

}

Funções

```c
int mensagem () /* Funcao simples: so imprime Ola! */
{
        printf ("Ola! ");
        return(0);
}
int main ()
{
        mensagem();
        printf ("Eu estou vivo!\n");
        return(0);
}
```

```c
#include <stdio.h>
int square (int x) /* Calcula o quadrado de x */
{
        printf ("O quadrado e %d",(x*x));
        return(0);
}
int main ()
{
        int num;
        printf ("Entre com um numero: ");
        scanf ("%d",&num);
        printf ("\n\n");
        square(num);
        return(0);
}
```

Vetores

```
int main (int argc, char* argv[]){
    setlocale(LC_ALL, "");

    char nome1[20]="Josimar Viana", nome2[20];

    for (int i=0;nome1[i]; i++){
        nome2[i]=nome1[i];
    }
    printf("%s", nome2);
    return 0;
}
```

Ponteiros

```
int x = 100;
```

1. Ao declararmos uma variável x como acima, temos associados a ela os seguintes elementos:
   - Um nome (x)
   - Um endereço de memória ou referência (0xbfd267c4)
   - Um valor (100)
2. Para acessarmos o endereço de uma variável, utilizamos o operador &

*analogia com endereço

Endereço	Valor
00000000	??
00000001	??
00000002	??
00000003	??
00000004	??
00000005	??
00000006	??
00000007	??
00000008	??
00000009	??
0000000A	??
0000000B	??
0000000C	??
0000000D	??

- A memória está formada por várias células.
- Cada célula contém um endereço e um valor.
- O tamanho do endereço e o tamanho do valor dependem da arquitetura do computador (32/64 bits)

Endereço	Valor
0000000D	??

Ponteiros

Endereço	Valor
00000000	??
00000001	??
00000002	??
00000003	??
00000004	??
00000005	??
00000006	??
00000007	??
00000008	??
00000009	??
0000000A	??
0000000B	??
0000000C	??
0000000D	??

} i

```
int main()
{
 → char i;
    return 0;
}
```

- Declaro um caracter chamado *i*.
- Os caracteres ocupam 1 byte na memória (para uma arquitetura de 32 bits)

Ponteiros

Endereço	Valor
00000000	??
00000001	
00000002	
00000003	
00000004	??
00000005	??
00000006	??
00000007	??
00000008	??
00000009	??
0000000A	??
0000000B	??
0000000C	??
0000000D	??

i

```
int main()
{
 → int i;
    return 0;
}
```

- Declaro um número inteiro chamado *i*.
- Os inteiros ocupam 4 bytes na memória (para uma arquitetura de 32 bits)

Ponteiros

Endereço	Valor
00000000	??
00000001	
00000002	
00000003	
00000004	
00000005	
00000006	
00000007	
00000008	??
00000009	??
0000000A	??
0000000B	??
0000000C	??
0000000D	??

i

```
int main()
{
 → double i;
    return 0;
}
```

- Declaro um número flutuante de dupla precisão chamado *i*.
- Os flutuantes de dupla precisão ocupam 8 bytes na memória (para uma arquitetura de 32 bits)

Ponteiros

- Há vários tipos de ponteiros:
  - ponteiros para caracteres
  - ponteiros para inteiros
  - ponteiros para ponteiros para inteiros
  - ponteiros para vetores
  - ponteiros para estruturas

Ponteiros

- Utilizamos o operador unário *

```
int *p_int;
char *p_char;
float *p_float;
double *p_double;
```

Ponteiros

Endereço	Valor	
00000000	??	c
00000001		
00000002		
00000003		
00000004	??	i
00000005		
00000006		
00000007		
00000008	??	f
00000009		
0000000A		
0000000B		
0000000C	??	d
0000000D		
0000000E		
0000000F		

```
int main()
{
 → char* c;
 → int* i;
 → float* f;
 → double* d;
    return 0;
}
```

- Declaração de quatro ponteiros(*c*, *i*, *f* e *d*). Cada ponteiro de um tipo diferente(*char*, *int*, *float*, *double*).
- Todos eles ocupam o mesmo espaço na memória, 4 bytes.
- Isso acontece porque todos eles armazenam endereços de memória, e o tamanho de um endereço de memória é o mesmo para todos os tipos.

Ponteiros

- Na declaração anterior os ponteiros ainda não foram inicializados, isto significa que eles apontam para um lugar indefinido (inclusive apontando para uma porção de memória no SO);
- Para utilizar um ponteiro ele deve ser inicializado (assim como qualquer variável no C);
- **Para sabermos o endereço de uma variável basta usar o operador &.**

```
int count=10;
int *pt;
pt=&count;
```

Ponteiros

Endereço	Valor
00000000	15
00000001	
00000002	
00000003	
00000004	??
00000005	??
00000006	??
00000007	??
00000008	??
00000009	??
0000000A	??
0000000B	??
0000000C	??
0000000D	??
0000000C	??
0000000D	??

i

```
int main()
{
    int i;
→   i = 15;
    char c = 's';
    int *p = &i;
    *p = 25;
    return 0;
}
```

- A variável *i* recebe o valor 15. Esse valor 15 é colocado no campo valor da memória alocada previamente para a variável i.
- Lembrem que essa notação com o 15 na ultima casa é apenas didática na verdade esse valor é tudo em binário

Ponteiros

Endereço	Valor	
00000000	15	i
00000001		
00000002		
00000003		
00000004	s	} c
00000005	??	
00000006	??	
00000007	??	
00000008	??	
00000009	??	
0000000A	??	
0000000B	??	
0000000C	??	
0000000D	??	
0000000C	??	
0000000D	??	

```c
int main()
{
    int i;
    i = 15;
→   char c = 's';
    int *p = &i;
    *p = 25;
    return 0;
}
```

- A variável *c* do tipo **char** é criada e inicializada com o valor 's'.

Ponteiros

Endereço	Valor	
→00000000	15	i
00000001		
00000002		
00000003		
00000004	s	} c
00000005	00	} p
00000006	00	} p
00000007	00	} p
00000008	00	} p
00000009	??	
0000000A	??	
0000000B	??	
0000000C	??	
0000000D	??	
0000000C	??	
0000000D	??	

```c
int main()
{
    int i;
    i = 15;
    char c = 's';
 → int *p = &i;
    *p = 25;
    return 0;
}
```

- Ponteiro de inteiro declarado.
- O nome desse ponteiro é p e ele é inicializada no momento de sua criação.
- O valor que esse ponteiro recebe é o endereço da variável $i(\&i)$ que nesse caso é o endereço 00000000.
- Dizemos que p aponta para i.

Ponteiros

Endereço	Valor	
00000000	25	i
00000001		
00000002		
00000003		
00000004	s	c
00000005	00	p
00000006	00	p
00000007	00	p
00000008	00	p
00000009	??	
0000000A	??	
0000000B	??	
0000000C	??	
0000000D	??	
0000000C	??	
0000000D	??	

```
int main()
{
    int i;
    i = 15;
    char c = 's';
    int *p = &i;
→   *p = 25;
    return 0;
}
```

- Finalizando, fazemos uma atribuição.
- Colocamos 25 no valor apontado por *p*. Como visto no slide anterior *p* aponta para *i*
- Desse modo, colocamos 25 no valor da variável *i*.

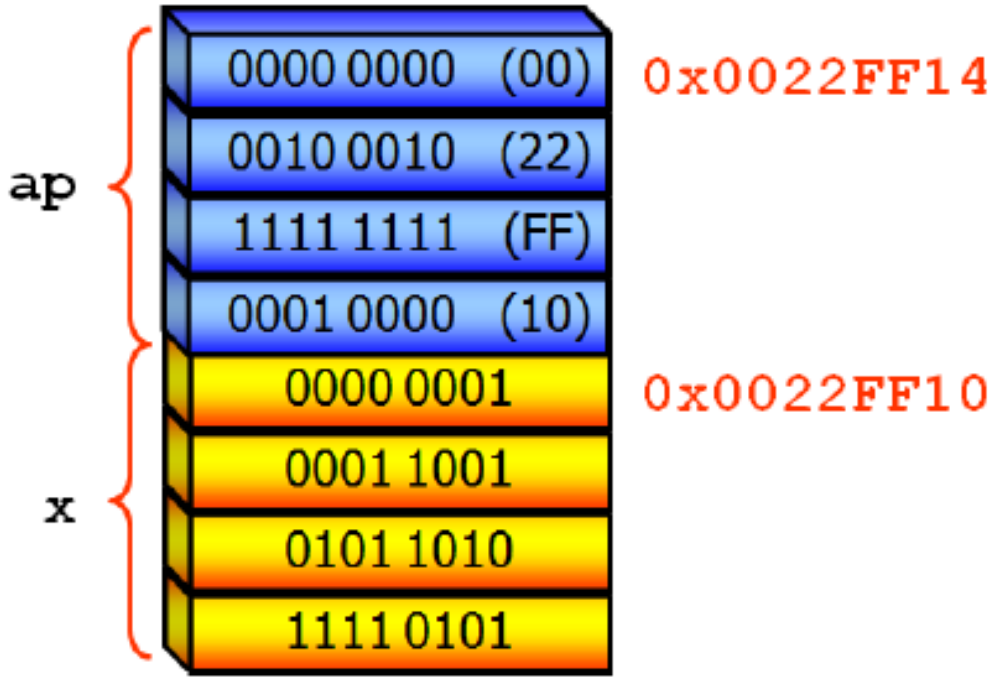
Ponteiros

- Um apontador é uma variável que pode armazenar endereços de outras variáveis

```
int x;
int *px; //apontador para inteiros
px = &x; //px aponta a x
```

Ponteiros

```
int x;
int *ap;      // apontador para inteiros
ap = &x;      // ap aponta para x
```



Ponteiros

```
int x;
int* px = &x;

*px = 3;
```

*px pode ser usado em qualquer contexto que x seria.

Ponteiros

```c
#include <stdio.h>
int main ()
{
        int num,valor;
        int *p;
        num=55;
        p=&num; /* Pega o endereco de num */
        valor=*p; /* Valor e igualado a num de uma maneira indireta */
        printf ("\n\n%d\n",valor);
        printf ("Endereco para onde o ponteiro aponta: %p\n",p);
        printf ("Valor da variavel apontada: %d\n",*p);
        return(0);
}
```

Ponteiros

```c
#include <stdio.h>
int main ()
{
        int num,*p;
        num=55;
        p=&num; /* Pega o endereco de num */
        printf ("\nValor inicial: %d\n",num);
        *p=100; /* Muda o valor de num de uma maneira indireta */
        printf ("\nValor final: %d\n",num);
        return(0);
}
```

Operações permitidas
p1=p2.
*p1=*p2.

Passagem por valor

```
void nao_troca(int x, int y)
{
    int tmp;

    tmp = x;
    x = y;
    y = tmp;
}
```

Apenas uma cópia de x e y é passada para a função

Passagem por referência

```
void troca(int *x, int *y)
{
    int tmp;

    tmp = *x;
    *x = *y;
    *y = tmp;
}
```

Ponteiros e vetores

- Ponteiros e vetores tem uma ligação muito forte;

*nome_da_variável é equivalente a nome_da_variável[0]

Ponteiros e vetores

```
#include <stdio.h>
int main ()
{
    int matrx [10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
    int *p;
    p=matrx;
    printf ("O terceiro elemento do vetor e: %d",p[2]);
    return(0);
}
```

Podemos ver que **p[2]** equivale a ***(p+2)**.

Ponteiros e vetores

```
int main ()
{
    float matrx [50][50];
    int i,j;
    for (i=0;i<50;i++)
                for (j=0;j<50;j++)
                matrx[i][j]=0.0;
    return(0);
}
```

```
int main ()
{
    float matrx [50][50];
    float *p;
    int count;
    p=matrx[0];
    for (count=0;count<2500;count++)
      {
              *p=0.0;
              p++;
      }
    return(0);
}
```

Ponteiros e vetores

```
void imprime_vetor1(int v[], int n) {
    int i;
    for (i = 0; i < n; i++)
        cout << v[i] <<" ";
    cout << endl;
}

void imprime_vetor2(int* pv, int n) {
    int i;
    for (i = 0; i < n; i++)
        cout << pv[i] << " ";
    cout << endl;
}
```

Ponteiros e vetores

```
void imprime_vetor3(int *pv, int n) {
    int i;
    for (i = 0; i < n; i++) {
        cout << *(pv + i) << " ";
    }
    cout << endl;
}
```

Ponteiros e vetores

```
#include <stdio.h>
void StrCpy (char *destino,char *origem)
{
while (*origem)
        {
        *destino=*origem;
        origem++;
        destino++;
        }
*destino='\0';
}
int main ()
{
    char str1[100],str2[100],str3[100];
    printf ("Entre com uma string: ");
    gets (str1);

    StrCpy (str2,str1);
    StrCpy (str3,"Voce digitou a string ");
    printf ("\n\n%s%s",str3,str2);
    return(0);
}
```