

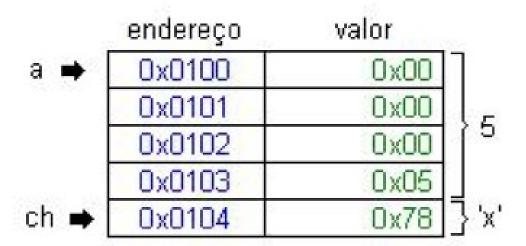
Ponteiros em C

Programação Estruturada Prof: Delcino P. Jr.

Definições

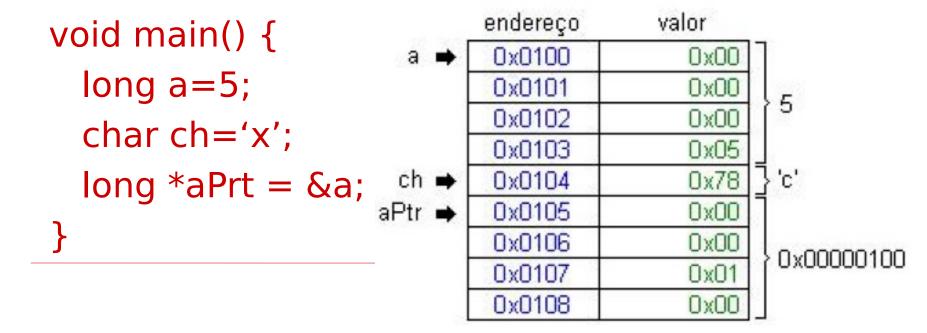
Variáveis : endereçam uma posição de memória que contem um determinado valor dependendo do seu tipo (char, int, float, double, ...)

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



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.



Declaração de 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.

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

```
void main() {
                                                    valor
                                     endereço
  long a=5;
                                      0x0100
                                                       0x00
                                      0x0101
                                                       0x00
  char ch='x';
                                      0x0102
                                                       0x00
  long *aPrt = &a;
                                      0x0103
                                                       0x05
  printf("%d",*aPrt);
                                                       0x78
                              ch \Rightarrow
                                      0x0104
  printf("%p",aPrt);
                                      0x0105
                            aPtr →
                                                       0x00
                                      0x0106
                                                       0x00
  printf("%p",&aPrt);
                                                               0x00000100
                                      0x0107
                                                       0x01
                                      0x0108
                                                       0x00
```

- O que será impresso na tela?
 - **-** 5
 - 0x0100
 - 0x0105

Alguns exemplos... (1)

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

Alguns exemplos... (2)

```
#include <stdio.h>
main ()
 int num,*p;
 num=55;
 p=# /* 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);
```

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;
```

Alguns exemplos... (3)

```
#include <stdio.h>
main ()
  int num,*p1, *p2;
  num=55;
  p1=# /* Pega o endereco de num */
  p2=p1; /*p2 passa a apontar para o mesmo endereço apontado por p1 */
  printf("Conteúdo de p1: %p",p1);
  printf("Valor apontado por p1: %d",*p1);
  printf("Conteúdo de p2: %p",p2);
  printf("Valor apontado por p2: %d",*p2);
```

Alguns exemplos... (4)

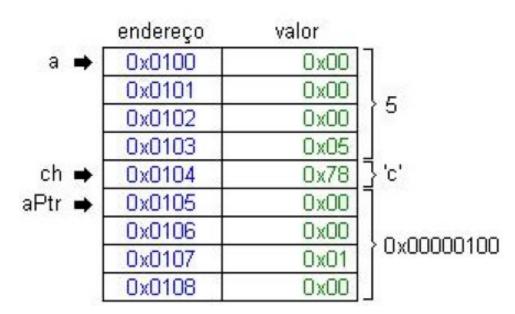
```
#include <stdio.h>
main ()
  int num,*p1, *p2;
  num=55;
  p1=# /* Pega o endereco de num */
 *p2=*p1; /* p2 recebe o valor apontado por p1 */
  printf("Conteúdo de p1: %p",p1);
  printf("Valor apontado por p1: %d",*p1);
  printf("Conteúdo de p2: %p",p2);
  printf("Valor apontado por p2: %d",*p2);
```

Soma de Ponteiro (endereço)

```
#include <stdio.h>
#include <stdlib.h>
void main(){
int *a, *b, *c;
int x, y, z;
x=10; y=20; z=30;
a=&x; b=&v;
c=a; c=c+2;
printf(" endereco de X %p \n",&x); printf(" endereco de Y %p \n",&y); printf(" endereco de Z %p
   \n",&z);
printf(" conteudo de X %d \n",*a); printf(" conteudo de Y %d \n",*b); printf(" conteudo %d \n",*c);
```

- Incremento/Decre mento:
 - Apontar para o próximo valor do mesmo tipo para o qual o ponteiro aponta:

```
long *aPtr, a=5;
aPtr=&a;
aPtr++;
```



- Qual será o valor endereçado por aPtr+ + ??
 - Se aPtr é long, como o long ocupa 4 bytes, aPtr irá apontar para o endereço 0x00000104
 - Este é o principal motivo que nos obriga a definir um tipo para um ponteiro!!!



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

Alguns exemplos... (5)

```
#include <stdio.h>
main ()
 long num;
 long *p;
 num=55;
 p=#
 printf("Conteúdo de p: %p",p);
 printf("Valor apontado por p: %d",*p);
 printf("Conteúdo de p incrementado: %p",++p);
 printf("Valor apontado por p incrementado: %d",*p);
```

Alguns exemplos... (6)

```
#include <iostream.h>
main ()
 long num;
 long *p;
 num=55;
 p=#
 printf("Conteúdo de p: %p",p);
 printf("Valor apontado por p: %d",*p);
 printf("Conteúdo de p incrementado: %p",++(*p));
 printf("Valor apontado por p incrementado: %d",*p);
```

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=vet; ou
 - int *p; p=&vet[0];

Alguns exemplos... (8)

```
main ()
 int vet [4];
 int *p;
 p=vet;
 for (int count=0;count<4;count++)</pre>
  *p=0;
  p++;
 for (int i=0; i<4; i++)
  printf("%d - ",vet[i]);
```

Alguns exemplos... (9)

```
main ()
 float matrx [4][4];
 float *p;
 int count;
 p=matrx[0];
 for (count=0;count<16;count++)
  *p=0.0;
  p++;
```

Alguns exemplos... (10) - Strings

```
StrCpy (char *destino,char *origem)
 while (*origem)
  *destino=*origem;
  origem++;
  destino++;
 *destino='\0';
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^{s},str3,str2);
```

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;

Ponteiros como vetores

- Quando um ponteiro está endereçando um vetor, podemos utilizar a indexação também com os ponteiros:
- Exemplo:

```
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]);
```

Neste caso p[2] equivale a *(p+2)

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 S.O.

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
}
```

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.

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).

Alocação dinâmica de vetores

```
#include <stdlib.h>
main() {
  int *v, i, n;
  scanf("%d", &n); // le n
  //aloca n elementos para v
  v = (int *) malloc(n*sizeof(int));
  // zera o vetor v com n elementos
  for (i = 0; i < n; i++) \vee [i] = 0;
  // libera os n elementos de v
  free(v);
```