Universidade do Vale do Rio dos Sinos
Estruturas de Dados em C

Ponteiros

Profa. Janaína Lemos

2020/1

Endereços e Ponteiros

- Os conceitos de endereço e ponteiro ficam ocultos em algumas linguagens.
- Na linguagem C, esses conceitos são explícitos.
- A memória RAM de um computador é uma sequência de bytes que armazenam um de 256 possíveis valores. Esses bytes são numerados sequencialmente e o número de um byte é o seu endereço.

Endereços e Ponteiros

- Cada dado na memória do computador ocupa um certo número de bytes consecutivos. Por exemplo: um ***char*** ocupa 1 byte e um ***int*** ocupa 4 *bytes* em muitos computadores.
- O número exato de bytes de um dado é fornecido pelo operador ***sizeof***: a expressão ***sizeof (int)***, por exemplo, dá o número de bytes de um **int** no seu computador.
- Cada dado na memória tem um endereço.
- Por exemplo, após a declaração:

```
char letra;
int num;
```

- As variáveis **letra** e **num** podem ter os endereços 0028FF17e 0028FF18.

O operador de endereço &

- O endereço de uma variável é dado pelo operador &.
- Se var é uma variável então &var é o seu endereço.

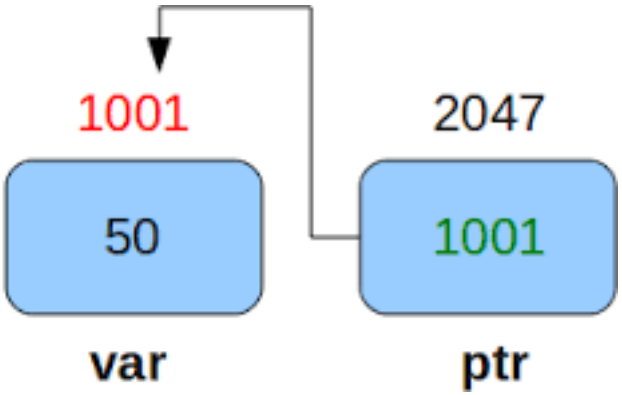
- Exemplo:

```
int var;
scanf("%d", &var);
```

- Nesse caso a função *scanf* lê um inteiro do teclado e armazena o valor lido no **endereço da variável var**.

Ponteiros

- Um ponteiro é um tipo especial de variável que armazena endereços.




```
int var = 50;
/* var recebe o valor 50
*/
int *ptr = NULL;
/* declara um ponteiro */

ptr = &var;
/* ptr recebe o endereço
de var (1001) */
```

Ponteiros

- Exemplo: impressão do valor de uma variável ponteiro (um endereço):

```
#include <stdio.h>
int main()
{
   int var;
   int *ptr = NULL;

   ptr = &var;
   printf("O endereço de var é: %p\n",
ptr);
}
```

Ponteiros

- Ponteiros devem ser inicializados com o valor especial **NULL**, que não é o endereço de lugar algum.
- A constante **NULL** está definida em **stdlib.h** e seu valor é **zero** na maioria dos computadores.

Como declarar um ponteiro?

- Existem diversos tipos de ponteiros: ponteiros para char, int, float, etc. O computador precisa saber de que tipo de ponteiro você está falando. Para declarar um ponteiro, escreva:

  tipo *nome_da_variavel;

- **Exemplo:**

```
int *pi = NULL;
char *pc = NULL;
```

- pi é um ponteiro para int e pc é um ponteiro para char.

O operador *

- Se um ponteiro p tem valor diferente de **NULL** (ou seja, contém um endereço válido) então *p é o valor contido nesse endereço .

  Exemplo:

```
#include <stdio.h>
int main()
{
   int x=5;
   int *px = NULL;

   px = &x;
   printf("Valor de px (end. de x): %p,
Valor contido em x: %d\n", px, *px);
}
```

O operador *

- Resultado da execução do exemplo:

```
Valor de px: 0028FF18

Valor de x: 5
```

Exemplo

- **Como fazer c = a + b usando ponteiros?**

```
#include <stdio.h>
int main()
{
   int a = 3, b = 7, c = 0;
   int *p = NULL, *q = NULL;
   p = &a;
   q = &b;
   c = *p + *q;
   printf("Valor de c: %d\n", c);
   // O valor de c será 10
}
```

Ponteiros e vetores

- Os elementos de um vetor de char possuem endereços *consecutivos* na memória do computador. Mas para os outros tipos de dados os endereços nem sempre são consecutivos, uma vez que cada elemento do vetor pode ocupar vários bytes.
- Porém, o compilador se encarrega de ocultar os detalhes para que a diferença entre os endereços de elementos consecutivos pareça ser sempre 1.

Ponteiros e vetores

- **Exemplo:**

```
char nome[20], *ps = NULL;
int val[10], *pi = NULL;
```

```
ps = nome; //é o mesmo que ps = &nome[0]
pi = val;  //é o mesmo que pi = &val[0]
```

```
ps++;  //ps irá apontar para nome[1], ou
seja, avançou 1 byte
pi++;   //pi irá apontar para val[1] ],
ou seja, avançou 4 bytes
```

Exercício 1:

- Qual dos trechos de código a seguir está correto? Justifique sua resposta.

a)

```
int main(){
   char s[1000];
   char *p = NULL;

   printf("s: ");
   gets(s);

   p = s[0];
   printf("s: %s\n", p);
}
```

b)

```
int main(){
   char s[1000];
   char *p = NULL;

   printf("s: ");
   gets(s);

   p = &s[0];
   printf("s: %s\n", p);
}
```

Exercício 2:

- Construa um programa que leia uma palavra do teclado, mostre na tela o endereço da *string* onde ela foi armazenada e após, mostre na tela a terceira e a quarta letra dessa palavra. Use um ponteiro para armazenar os endereços dessas letras.