Instituto Federal de Educação, Ciência e Tecnologia do Amazonas

Campus Manaus - Distrito Industrial (CMDI)

Linguagem C

Prof. Isaac Benjamim Benchimol
ibench@ifam.edu.br

Instituto Federal de Educação, Ciência e Tecnologia do Amazonas

Campus Manaus - Distrito Industrial (CMDI)

Ponteiros

Ponteiros

- Um ponteiro é uma variável que contém um endereço de memória. Esse endereço é normalmente a posição de uma outra variável na memória.
- Sintaxe: ***tipo* *nome_ponteiro;**
- *tipo* do ponteiro define que tipo de variável o ponteiro pode apontar.
- Operadores de Ponteiros:
  - & Operador unário que retorna o endereço na memória do seu operando.
  - * Operador unário que retorna o valor da variável localizada no endereço que o segue.

Exemplos

- `int y, x=0, *px, *py;`

  `px=&x;` "px aponta para x"

  `y=*px;` "y recebe conteúdo de x, ou seja, 0"

  `py=px;` "py também aponta para x"

  `*py=3;` "variável apontada por py, ou seja, x, recebe 3"

- ```
  float x=10.1, y;
  int *p;
  p=&x;     /* erro */
  y=*p;
  printf("%f",y);
  ```

- ```
  int x, *p;
  x=10;
  *p=x;     /* cuidado, ponteiro perdido */
                  /* (contém lixo)
  ```

Incremento e Decremento de Ponteiros

- Os ponteiros não são necessariamente incrementados e decrementados em uma unidade, mas pelo tamanho do tipo apontado pelo ponteiro.
- Os operadores ++ e -- possuem precedência sobre o * e operadores matemáticos.
- Qual a diferença entre

```
p++;        (*p)++;        *(p++);
```

Incremento e Decremento de Ponteiros

p++; incrementa o ponteiro, ou seja, o endereço(**p** passa a apontar para a posição de memória imediatamente superior).

(*p)++; incrementa o conteúdo apontado por **p**, ou seja, o valor armazenado na variável para a qual **p** está apontando.

***(p++);** acessa o valor apontado por **p** e incrementa **p**, como em p++.

Ponteiros e Matrizes

- Há um grande relacionamento entre ponteiros e matrizes.
- Versões com ponteiros são mais rápidas.
- Os ponteiros possuem mais facilidades de manipulação.
- Ao contrário das matrizes, os ponteiros podem ser incrementados e decrementados diretamente.

Exemplo

- Lendo um vetor com ponteiro:

```
main()
{
int a[5]={0,1,2,3,4}, *p, x;
p=&a[0];  /* mesmo que p=a */
x=*(p+2); /* x recebe a[2] */
printf("%d", x);
}
```

Exemplo

- Construindo um vetor com ponteiro:

```
main(){
int vet[10], *p, i;
p=vet;     // p aponta para vet
for (i=0;i<10;i++)   *p++=i;

p=vet;
for (i=0;i<10;i++)
     printf("%d", *p++);
}
```

Exemplo

- Ponteiro com string:

```
main(){
   char str[80], *p;
   printf("Digite string em letras maiúsculas:");
   gets(str);
   printf("Eis a string em letras minúsculas:");
   p=str;  /* p aponta para o primeiro
       elemento de str */
   while (*p) printf("%c", tolower(*p++));
}
```

Exemplo

- Matrizes de Ponteiros:

```
main(){
char *erro[2]={"arquivo
   inexistente\n",
                "erro na leitura\n"};
printf ("%s", erro[0]);
printf ("%s", erro[1]);
}
```

Exercícios

1. Usando ponteiros, faça um programa que imprima uma string invertida.
2. Usando ponteiros, faça um programa que preencha uma matriz 10x10 de inteiros com os números de 1 a 100.
3. Usando ponteiros, faça um programa que leia duas strings e as concatene numa terceira

Instituto Federal de Educação, Ciência e Tecnologia do Amazonas

Campus Manaus - Distrito Industrial (CMDI)

Alocação Dinâmica de Memória

Alocação Dinâmica de Memória

- Um programa em C pode usar duas formas de armazenar informações na memória do computador.
- A primeira maneira usa **variáveis globais e locais**. Variáveis globais têm seu armazenamento fixo durante todo o tempo de execução do programa. Variáveis locais são armazenadas na pilha do programa. Entretanto essa maneira requer que o programador saiba, com antecedência, o montante de armazenamento necessário para cada situação.
- A segunda maneira é através de **alocação dinâmica** da área de memória livre (heap) do programa.
- As funções **malloc()** e **free()** formam o sistema de alocação dinâmica de C.

malloc()

- void *malloc(unsigned size)

Retorna um ponteiro `void` para o primeiro byte de uma região da memória de tamanho *size* da área de alocação dinâmica. O ponteiro `void` deverá ser atribuído a um ponteiro do tipo desejado (usar *cast* explícito). Se não houver memória suficiente para satisfazer o pedido, **malloc()** retornará um ponteiro NULL.

free()

- free (void *p)

  O oposto de malloc(), **free()** devolve ao sistema a memória previamente alocada. **free()** nunca deve ser chamada com um argumento inválido, porque isto fará com que o computador destrua a lista livre.

Exemplo

```
main() {  */ Aloca memória para 50 inteiros /*
    int *p, t;
    p=(int *)malloc(50*sizeof(int));
    if (!p) printf("Memória Insuficiente!");
      else {
        for (t=0; t<50; t++) *(p+t)=t;
        for (t=0; t<50; t++) printf("%d", *(p+t));
        free(p)
   }
}
```

Exercício

1. Escreva um programa que aloque espaço para cinco strings inseridas pelo usuário.
2. Escreva uma função getstruct() que aloque memória para uma estrutura `addr` e retorne o ponteiro para a memória alocada:

```
struct addr {
   char nome[40];
   char rua[40];
   char cidade[30];
   char esrado[3];
   char cep[10];
};
```