Ponteiros

Ponteiros

1. O Que São

Variáveis do tipo **int** armazenam valores inteiros, as do tipo **float** armazenam números de ponto flutuante, já as do tipo **char** armazenam caracteres. Por sua vez, **ponteiros** armazenam endereços de memória e **ponteiro** também tem tipo.

Ponteiros

2. Declarando e Utilizando Ponteiros

Para declarar um ponteiro temos a seguinte forma geral:

*tipo_do_ponteiro *nome_da_variável;*

Exemplos de declarações:

```
int *pt;
char *pt2,*pt3;
```

UNIVASF

Ponteiros

2. Declarando e Utilizando Ponteiros

Um cuidado, muito importante, que deve ser tomado na manipulação de ponteiros, é o de inicializar um ponteiro antes de utilizá-lo. Pois, quando esses são declarados, apontam para um lugar indefinido.

Para atribuir um valor **válido** a um ponteiro recém criado poderíamos igualá-lo a um endereço de memória de uma variável declarada.

Mas, como saber a posição na memória de uma variável do nosso programa?

UNIVASF

Ponteiros

2. Declarando e Utilizando Ponteiros

Para saber o endereço de uma variável basta usar o operador **&**.

OBS.: Lembre-se da função scanf()

Veja o exemplo:

```
...
int cont=10;
int *pt;
pt=&cont;
…
*pt=12;
```

Após a inicialização podemos utilizar **pt**.

Podemos alterar o valor de **cont** usando **pt**.

Usaremos o operador "inverso" do operador **&**, que é o operador *.

Após **pt=&cont** a expressão ***pt** é equivalente ao próprio **cont**.

UNIVASF

```c
#include <stdio.h>
int main ()
{
   int num, valor, *p;
   num=55;
   p=&num;
   valor=*p;
   printf ("\n\n%d\n",valor);
   printf ("Endereco para onde o ponteiro aponta: %p\n", p);
   printf ("Valor da variavel apontada: %d\n",*p);
}
```

```c
#include <stdio.h>
int main ()
{
    int num,*p;
    num=55;
    p=&num;
    printf ("\nValor inicial: %d\n",num);
    *p=100;
    printf ("\nValor final: %d\n",num);
}
```

```c
#include <stdio.h>
int main ()
{
    int num,*p;
    num=55;
    p=&num;
    printf ("\nValor inicial: %d\n",num);
    printf ("Digite um valor inteiro:");
    scanf("%d", p);
    printf ("\nValor final: %d\n",num);
}
```

Ponteiros

3. Operações Aritméticas com Ponteiros

a) Atribuição

Se temos dois ponteiros, **p1** e **p2,** e quisermos que **p1** aponte para o mesmo lugar que **p2**, basta fazermos **p1=p2**.

É interessante observar que se o objetivo for que a área de memória apontada por **p1** tenha o mesmo conteúdo da área de memória apontada por **p2** deve-se fazer ***p1=*p2**.

UNIVASF

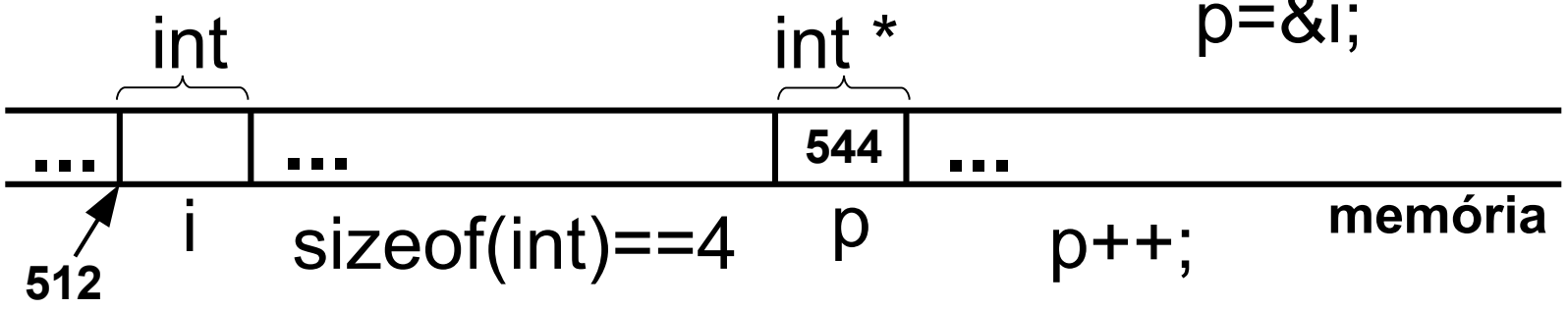
Ponteiros

3. Operações Aritméticas com Ponteiros

b) Incremento e Decremento

Quando incrementamos um ponteiro ele passa a apontar para o próximo valor do mesmo tipo para o qual o ponteiro aponta.

Esta é uma razão pela qual o compilador precisa saber o tipo de um ponteiro.



Ponteiros

3. Operações Aritméticas com Ponteiros

b) Incremento e Decremento (continuação)

O decremento funciona de forma semelhante. Supondo que **p** é um ponteiro, as operações são escritas, por exemplo, como:

```
p++;
p--;
```

Ponteiros

3. Operações Aritméticas com Ponteiros

b) Incremento e Decremento (continuação)

Estamos falando de operações com *ponteiros* e não de operações com o conteúdo das áreas de memória para as quais eles apontam.

Por exemplo, para incrementar o conteúdo da área de memória apontada pelo ponteiro **p**, faz-se:

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

UNIVASF

Ponteiros

3. Operações Aritméticas com Ponteiros

c) Soma e Subtração de Inteiros com Ponteiros

Vamos supor que você queira incrementar um ponteiro em 15 unidades. Basta fazer:

p=p+15; ou p+=15; **/*considerando que p é uma variável do tipo ponteiro*/**

E se você quiser acessar o conteúdo da memória apontada 15 posições adiante:

*(p+15);

Obs.: A subtração funciona de forma similar.

UNIVASF

Ponteiros

3. Operações Aritméticas com Ponteiros

d) Comparação entre dois Ponteiros

podemos saber se dois ponteiros são iguais ou diferentes (**==** e **!=**).

No caso de operações do tipo **>**, **<**, **>=** e **<=** estamos comparando qual ponteiro aponta para uma posição "mais alta" na memória. A comparação entre dois ponteiros se escreve como a comparação entre outras duas variáveis quaisquer:

Por exemplo, p1>p2

UNIVASF

Ponteiros

3. Operações Aritméticas com Ponteiros

Há entretanto operações que ***não*** podemos efetuar sobre um ponteiro. Não se pode dividir ou multiplicar ponteiros, adicionar dois ponteiros, adicionar ou subtrair **float**s ou **double**s a ponteiros.

UNIVASF

Ponteiros

Exercícios:

a) Explique a diferença, caso exista, entre

p++ (*p)++ *(++p)

b) O que quer dizer *(p+10)?

UNIVASF

Ponteiros

c) Qual o valor de y no final do programa? Escreva um /* comentário */ em cada comando de atribuição explicando o que ele faz e o valor da variável à esquerda do '=' após sua execução. Explique se os parênteses são realmente necessários.

```c
#include <stdio.h>
int main() {
    int *p, y, x;
    y = 0;
    p = &y;
    x = *p;
    x = 4;
    ++(*p);
    x--;
    (*p) += x++;
    printf ("y = %d\n", y);
}
```

Ponteiros e Vetores

Ponteiros e Vetores

- Vetores como ponteiros

Para que possamos compreender esta similaridade, devemos primeiro entender como a linguagem C trata vetores.

Quando declaramos um vetor da seguinte forma:

tipo_da_variável nome_da_variável [tam1][tam2] ... [tamN];

Ponteiros e Vetores

- Vetores como ponteiros

O compilador C calcula o tamanho, em bytes, necessário para armazenar este vetor. Como vimos, este tamanho é:

tam1 x tam2 x tam3 x ... x tamN x tamanho_do_tipo

O compilador então aloca este número de bytes em um espaço livre de memória. O *nome da variável* declarada é na verdade um ponteiro para o *tipo dos elementos do vetor* que aponta para o endereço inicial da área alocada.

UNIVASF

Ponteiros e Vetores

- Vetores como ponteiros

Mas, aí surge a pergunta: então como é que podemos usar a seguinte notação?

nome_da_variável[índice]

Isto pode ser facilmente explicado desde que você entenda que a notação acima é *absolutamente equivalente* a se fazer:

**(nome_da_variável+índice)*

UNIVASF

Ponteiros e Vetores

- Vetores como ponteiros

Dessa forma, um ponteiro pode ser utilizado, por exemplo, para fazer uma varredura sequencial de uma matriz. Pois, quando temos que varrer todos os elementos de uma matriz de uma forma sequencial, podemos usar um ponteiro, o qual vamos incrementando.

Qual seria a vantagem em se fazer uma varredura sequencial usando ponteiros?

UNIVASF

Ponteiros e Vetores

- Vetores como ponteiros

Considere o seguinte programa para zerar uma matriz:

```
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;
}
```

Ponteiros e Vetores

- Vetores como ponteiros

Podemos reescrevê-lo usando ponteiros:

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

Ponteiros e Vetores

Qual seria a vantagem em se fazer uma varredura sequencial usando ponteiros?

```
for (i=0;i<50;i++)
    for (j=0;j<50;j++)
        matrx[i][j]=0.0;
```

***(matrx+i*50+j)**

```
for (count=0, p=matrx[0]/*p=&matrx[0][0]*/; count<2500;count++)
    *(p++)=0.0;
```

Ponteiros e Vetores

- Vetores como ponteiros

Há uma **diferença** entre o nome de um vetor e um ponteiro que deve ser **destacada**: 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". Logo:

```
int vetor[10], *ponteiro, i;
ponteiro = &i;
/* as operações a seguir são inválidas */
vetor = vetor + 2;
vetor++;
vetor = ponteiro;
```

Ponteiros e Vetores

- Vetores como ponteiros

Se testarmos as operações anteriores em nossos compiladores. Eles darão, por exemplo, uma mensagem de erro do tipo:

- Lvalue;
- incompatible types in assignment.

UNIVASF

Ponteiros e Vetores

- Vetores como ponteiros

Exercício:

Construa um programa que declare uma matriz [3,4] de inteiros e a inicializa da seguinte forma:

$$\begin{bmatrix} 01 & 02 & 03 & 04 \\ 05 & 06 & 07 & 08 \\ 09 & 10 & 11 & 12 \end{bmatrix}$$

Depois, a imprima na saída padrão com o layout apresentado. As manipulações da matriz deve ser feitas utilizando um ponteiro.

UNIVASF

```c
#include <stdio.h>
#define nl 3
#define nc 4
int main () {
    int matriz[nl][nc],*p,i;
    for (i=0, p=&matriz[0][0];i<nl*nc;i++)
        *(p++)=i+1;
    for (i=0, p=matriz[0];i<nl*nc;i++)
        if (!(i%nc))
            printf ("| %02d ",*(p+i));
        else
            if (i%4==nc-1)
                printf("%02d |\n",*(p+i));
            else
                printf("%02d ",*(p+i));
}
```

$$\begin{bmatrix} 01_0 & 02_1 & 03_2 & 04_3 \\ 05_4 & 06_5 & 07_6 & 08_7 \\ 09_8 & 10_9 & 11_{10} & 12_{11} \end{bmatrix}$$