Vetores de Strings

Vetores de Strings

Se fizermos um vetor de strings estaremos construindo um vetor de vetores.

Esta estrutura é uma matriz bidimensional de **char**'s.

Podemos ver a forma geral de uma vetor de strings como sendo:

char nome_da_variável [num_de_strings][compr_das_strings];

UNIVASF

Vetores de Strings

Aí surge a pergunta: como acessar uma string individual?

Fácil. É só usar apenas o primeiro índice. Então, para acessar uma determinada string faça:

nome_da_variável [índice]

Vetores de Strings

Exemplo: O programa a seguir declara um vetor de string's, o inicializa com string's fornecidas através da entrada padrão e no final de seu processamento o retorna na saída padrão.

```c
#include <stdio.h>
int main ()
{
   char strings [5][100];
   int count;
   for (count=0;count<5;count++)
   {
      printf ("\n\nDigite a %dº string do vetor: ", count+1);
      scanf ("%99s", strings[count]);
   }
   printf ("\n\nAs strings que voce digitou foram:\n");
   for (count=0; count<5; count++)
      printf ("%s\n",strings[count]);
}
```

Vetores de Strings

Exercício:

Construa um programa que, com base no exemplo anterior, além de ler as 5 string's do vetor de strings leia mais uma string, a qual ele verificará se pertence ao vetor, caso esta pertença ele retornará a posição da string no vetor, caso contrario ele retornará uma mensagem indicando que ela não se encontra no vetor.

UNIVASF

```c
#include <stdio.h>
#include <stdlib.h>
int main () {
   char strings [5][100],string [100];
   int count,count2;
   for (count=0;count<5;count++) {
      printf ("\n\nDigite a %dº string do vetor: ", count+1);
      scanf ("%99[^\n]",strings[count]); /*Se necessário limpar o buffer*/
   }
   printf ("\nEntre com a string que voce deseja saber se pertence ao vetor:");
   scanf ("%99[^\n]", string); /*Se necessário limpar o buffer*/
   for (count=0;count<5;count++)
      for (count2=0;count2<100;count2++) {
         if (strings[count][count2]!=string[count2])
            break;
         if (string[count2]=='\0') {
            printf ("\nA string %s esta na posicao %d do vetor    de strings.\n", string, count+1);
            exit(0);
         }
      }
   printf ("\nA string %s nao esta contida no vetor de strings.\n", string);
}
```

```c
#include <string.h>
#include <stdio.h>
int main () {
   char strings [5][100],string [100];
   int count;
   for (count=0;count<5;count++) {
      printf ("\n\nDigite a %dº string do vetor: ", count+1);
      gets (strings[count]);
   }
   puts ("\nEntre com a string que voce deseja  saber se pertence ao vetor:");
   gets(string);
   for (count=0;count<5;count++)
      if (!strcmp(strings[count],string))
         break;
   if (count<5)
      printf ("\nA string %s esta na posicao %d do vetor    de strings.\n", string, count+1);
   else
      printf ("\nA string %s nao esta contida no vetor de   strings.\n", string);
}
```

Vetores de Strings

Funções de entrada e saída formatada que trabalham sobre strings

- sprintf e sscanf

sprintf e sscanf são semelhantes a printf e scanf. Porém, ao invés de escreverem na saída padrão ou lerem da entrada padrão, escrevem ou lêem em uma string. Suas formas gerais são:

sprintf (string_destino, string_de_controle, lista_de_argumentos);

sscanf (string_origem, string_de_controle, lista_de_argumentos);

UNIVASF

Vetores de Strings

Funções de entrada e saída formatada que trabalham sobre strings (continuação)

Estas funções são muito utilizadas para fazer a conversão entre dados na forma numérica e sua representação na forma de strings e vice-versa. No programa a seguir, por exemplo, a variável i é "impressa" em string1. Além da representação de i como uma string, string1 também conterá "Valor de i = " .

UNIVASF

Vetores de Strings

Funções de entrada e saída formatada que trabalham sobre strings (continuação)

Exemplo:

```c
#include <stdio.h>
int main()
{
    int i;
    char string1[25];
    puts( "Entre um valor inteiro: ");
    scanf("%d", &i);
    sprintf(string1,"Valor de i = %d", i);
    puts(string1);
}
```

Vetores de Strings

Funções de entrada e saída formatada quetrabalham sobre strings (continuação)

Já no próximo programa, foi utilizada a função *sscanf* para converter informações armazenadas em string1 em valores numéricos:

UNIVASF

Vetores de Strings

Funções de entrada e saída formatada que trabalham sobre strings (continuação)

Exemplo:

```c
#include <stdio.h>
int main()
{
   int i, j;
   float k;
   char string1[]= "10 20 5.89";
   sscanf(string1, "%d %d %f", &i, &j, &k);
   printf("Valores lidos: %d, %d, %.2f", i, j, k);
}
```

Vetores de Strings

Funções de entrada e saída formatada que trabalham sobre strings (continuação)

Exercício:

Construa um programa que declare um vetor de strings e outro de inteiros, ambos com 10 elementos. O programa deve inicializar os vetores com valores fornecidos pelo usuário através da entrada padrão. Depois, deve acrescentar os inteiros, do vetor de inteiros, no final das strings correspondentes no vetor de strings. Ao termino do processamento o vetor de strings, com seus valores atualizados, deve ser apresentado na saída padrão.

UNIVASF

```c
#include<stdio.h>
#include<string.h>
int main() {
   char strings[10][100],aux[10];
   int inteiros[10],i;
   for (i=0;i<10;i++) {
      printf("\nEntre com a string[%d]: ",i+1);
      scanf("%99[^\n]", strings[i]);
      printf("\nEntre com o inteiro[%d]: ",i+1);
      scanf("%d",&inteiros[i]);
   }
   for (i=0;i<10;i++) {
      sprintf(aux,"%d",inteiros[i]);
      strcat(strings[i],aux); /*verificar possível falha de segmentação*/
   }
   for (i=0;i<10;i++)
      printf("\n%s\n",strings[i]);
}
```

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

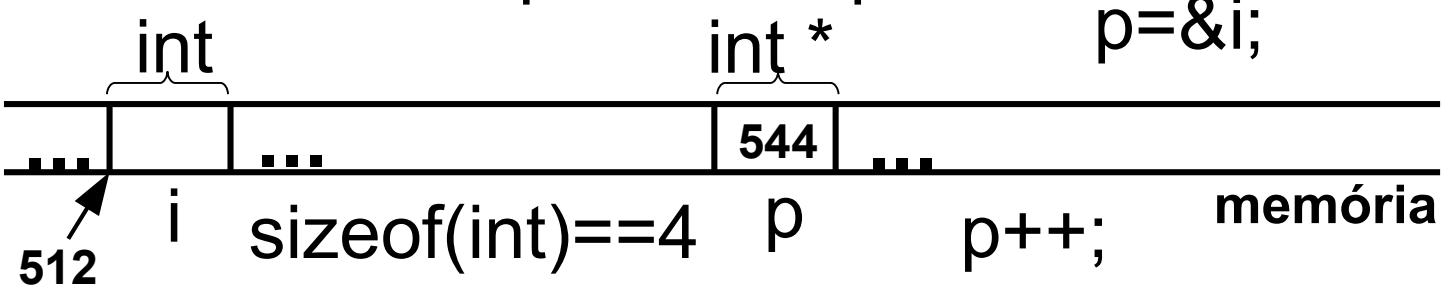
Ponteiros

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--;
```

UNIVASF

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)?

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);
}
```

UNIVASF

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;
}
```

UNIVASF

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;
```