PONTEIROS – PARTE 2

Estrutura de Dados

Ponteiros como vetores

- Sabemos agora que:
  - o nome de um vetor é um ponteiro constante;
  - podemos indexar o nome de um vetor.

- Logo, podemos também indexar um ponteiro qualquer.

Ponteiros como vetores

```
#include <stdio.h>
main ()
{
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]);
}
```

OBS.: Podemos ver que p[2] equivale a *(p+2).

Passando um vetor para função

- Para passar um vetor para uma função usando linguagem C utiliza-se um ponteiro como parâmetro da função. Ao usarmos o ponteiro como parâmetro, na realidade estamos passando o endereço inicial do vetor e não os seus elementos.
- O programa a seguir recebe 10 notas e armazena-as em um vetor.
- Para efetuar o cálculo da média foi implementada uma função que recebe a quantidade de elementos do vetor e o seu endereço inicial, cujo protótipo é:
  - **float media (int n, float *vnotas);**

Passando um vetor para função

- Posteriormente, no corpo da função efetuamos o processamento que manipula os dados do vetor de modo a calcular a média.

```
float media (int n, float
*vnotas)
{
   int i;
   float m = 0, soma = 0;
   for (i = 0; i < n; i++)
      soma = soma + vnotas[i];
    m = soma / n;
    return m;
}
```

Passando um vetor para função

```
float media (int n, float *vnotas);
int main (void)
{
  float vnotas[10], media_notas;
  int i;
  for (i = 0; i < 10; i++){
    printf("Digite notas: ");
    scanf("%f", &vnotas[i]);
  }
  media_notas = media(10,vnotas);
  printf ("\nMedia = %.1f \n",
media_notas );
  system("pause");
  return 0;
}
```

```
float media (int n, float *vnotas)
{
  int i;
  float m = 0, soma = 0;
  for (i = 0; i < n; i++)
            soma = soma + vnotas[i];
   m = soma / n;
   return m;
}
```

Ponteiro para struct

- É possível criar um ponteiro para uma struct, de forma semelhante à criação de ponteiro para outros tipos de dados.
- Para acessar os dados dos membros de uma struct usamos o operador ponto .
- Tomemos como exemplo a struct abaixo.

```
typedef struct
{
  int matricula ;
  float nota;
} tAluno;
```

Ponteiro para struct

- Declaramos **a1** como sendo uma struct do tipo **tAluno**.
  - **tAluno a1 ;**
- Em seguida, criamos o ponteiro ***ptrAluno** do tipo **tAluno** que recebe o endereço de **a1**;
  - tAluno ***ptrAluno** = &a1;
- Quando temos um ponteiro para uma struct podemos acessar os membros da struct da seguinte forma:
  - **(*ptrAluno).nota**
  - Exemplo: **(*ptrAluno).nota** = 8.5;

- Na realidade estamos usando o * para dereferenciar o ponteiro e o . para acessar o membro da struct.
- O operador -> permite fazer isso de forma mais simples.
  - Exemplo: **ptrAluno->nota = 8.5;**
  - Sendo assim, ptrAluno->nota equivale a (*ptrAluno).nota

Ponteiro para struct

```
typedef struct {
  int matricula ;
  float nota;
} tAluno;
int main (void)
{
      tAluno a1 ;
      tAluno *ptrAluno = &a1;
      a1.matricula =555;
      a1.nota = 8.0;
      printf ("matricula: %d nota: %.2f
\n", a1.matricula, a1.nota);
      (*ptrAluno).nota = 8.5;
      printf ("\nmatricula: %d nota: %.2f
\n",(*ptrAluno).matricula,
(*ptrAluno).nota);
      ptrAluno->nota = 9.0;
      printf ("\nmatricula: %d
nota: %.2f \n",ptrAluno-
>matricula,ptrAluno->nota);
      getch();
      return 0;
}
```

```
char nome_cliente[61];
```

Strings

- Na prática, as strings são usadas para representar textos. Em linguagem C, ao contrário de outras linguagens, não existe um tipo de dados string nativo.
- Para representar uma string em C, devemos criar um vetor de caracteres, ou seja um vetor do tipo char.
  - char nome_cliente[61];

Strings

- Este comando cria a variável nome_cliente como um vetor de char com capacidade de armazenamento de 61 caracteres.
- Ocorre, que o último caracter de uma string, deve ser sempre o caracter nulo “\0” que serve para indicar o final da string.
- Sendo assim, em nosso exemplo temos 60 caracteres úteis para armazenar o nome, pois o \0 é o terminador da string e ocupa uma posição de armazenamento.

Strings

```
char nome_cliente[30] = "Fulano";
char nome_cliente[30] = {'F','u','l','a','n','o'};

//Inicializando uma string sem definir //o tamanho do vetor

char nome_cliente[] = "Fulano";

//para ler uma string, porem o espaço identifica o fim de leitura
scanf("%s",nome);
printf("O nome armazenado foi: %s", nome);
```

Strings

- Para conseguirmos usar o espaço, usamos a função gets

```
char nome[61];
printf("Digite seu nome: ");
gets(nome);
printf("O nome armazenado foi: %s", nome);
```

Strings com Ponteiros

- Seguindo o raciocínio de ponteiros, podemos criar constantes de strings usando ponteiros, conhecidos como cadeia de constantes, são do tipo char*.
  - `char *nome = "Um string de caracteres";`
- Precisamos, para isto, entender como o C trata as strings constantes.
- Toda string que o programador insere no programa é colocada num banco de strings que o compilador cria.
- No local onde está uma string no programa, o compilador coloca o endereço do início daquela string (que está no banco de strings).

Strings com Ponteiros

- Inicializando Ponteiros (continuação)
- É por isto que podemos usar strcpy() do seguinte modo:
  - `char *nome;`
  - `strcpy(nome,"String constante.");`
  - `printf("Nome = %s", nome);`

Ponteiro para funções

- Ponteiros para funções é algo extremamente interessante, eficiente e elegante. Pode ser usado para substituir switch-case, definir em tempo de execução qual função deve ser chamada ou implementar callbacks.
- Ponteiros para funções são na realidade ponteiros que apontam para o endereço de uma função. Uma função possui um endereço de memória. Quando o programa é executado todo seu conteúdo é colocado na memória, então uma função é como um variável do tipo inteira, nada mais que um endereço de memória.

Ponteiro para funções

```
float adicao(float a, float b) {return a+b;};
float subtracao(float a, float b) {return a-b;};
float multiplicacao(float a, float b) {return a*b;};
float divisao(float a, float b) {return a/b;};

int main()
{
   float(*pt2Func[4])(float, float) = {NULL};
   float(*Func)(float, float) = NULL;

   pt2Func['+'] = &adicao;
   pt2Func['-'] = &subtracao;
   pt2Func['/'] = &divisao;
   pt2Func['*'] = &multiplicacao;

   float a;
   float b;
   char operator;
```

Ponteiro para funções

```
do{
    printf("Informe a: ");
    scanf("%f",&a);
    printf("\nInforme operador: ");
    scanf("%c",&operator);
    printf("\nInforme b: ");
    scanf("%f",&b);

     Func = pt2Func[operator];
     printf("\nResultado: %f\n", Func(a,b));
     printf("\nDeseja Continuar (y/n):");
     scanf("%c",&operator);
}while(operator != N);
```

Ponteiro para funções

- A noção de ponteiros permite supor que eles podem se referir a qualquer objeto localizado na memória. Tais objetos incluem também as funções e, portanto, é possível criar um ponteiro para uma função. A sintaxe é semelhante, mas há algumas regras próprias que podem ser observadas no programa de exemplo a seguir.

```
#include <stdio.h>


main(){
        int (*ptrf) ();
        ptrf = printf;
        (*ptrf)("Teste de ponteiro");
}
```