Métodos Computacionais

Tipos Estruturados

Tipos Estruturados

- C oferece tipos primitivos que servem para representar valores simples
  - Reais (float, double), inteiros (int), caracter (char)
- C oferece também mecanismos para estruturar dados complexos nos quais as informações são compostas por diversos campos

Tipos Estruturados !

Tipos Estruturados

- Exemplo de um tipo estruturado:
  - Considere um ponto representado por duas coordenadas: x e y
  - **Sem** mecanismos para agrupar as duas coordenadas, teríamos:

```
float    x ;

float    y ;
```

- Uma estrutura em C serve para agrupar diversas variáveis dentro de um único contexto

```
struct   ponto     {

                   float    x ;

                   float    y ;   }
```

Tipos Estruturado: Ponto

- A estrutura *ponto* passa a ser um tipo
- Então, podemos declarar uma variável deste tipo da seguinte forma:

```
struct   ponto   p ;
```

  - Elementos de uma estrutura são acessados via o operador de acesso (“.”)
    - Para acessar as coordenadas:

```
p.x  =  10.0;
p.y  =  5.0;
```

Utilizando oTipo Estruturado Ponto

```
*/ programa que captura e imprime coordenadas*/
#include <stdio.h>
struct   ponto  {
      float    x ;
      float    y ;  } ;

int  main ( void ){
   struct   ponto   p ;
   printf("\nDigite as coordenadas do ponto (x,y)") ;
   scanf ("%f %f", &p.x , &p.y ) ;
   printf("O ponto fornecido foi:(%f,%f)\n",p.x, p.y);
   return   0 ;
}
```

Na declaração de variável , o tipo é struct ponto

variável p não precisa de parênteses

Usando Ponteiro para Estruturas

- Ponteiro para estruturas
  - Declaração da variável

```
struct   ponto   *p ;
```

  - "p" armazena o endereço para uma estrutura
  - Para acessar os campos de uma estrutura por meio de um ponteiro, usa-se o operador (->):

```
p -> x  =  12.0 ;  ⇔ (*p).x = 12.0
p -> y  =  10.0 ;
```

Precisa de parênteses para variável p

Passagem de Estruturas para Funções

- Considere a função abaixo:

```
void   imprime  ( struct  ponto   p ){
   printf("O ponto fornecido foi:(%f,%f)\n",p.x,p.y);
}
```

- Valores da estrutura não podem ser modificados
- Cópia da estrutura na pilha não é eficiente.
- É mais conveniente passar apenas o ponteiro da estrutura

Passagem de Ponteiros para Estruturas para Funções

- Uma função para imprimir as coordenadas

```
void   imprime  ( struct  ponto*   p ){
 printf("O ponto fornecido foi:(%f,%f)\n",p->x,p->y);
}
```

- Uma função para ler as coordenadas

```
void   captura  ( struct  ponto   *p ) {
  printf( "Digite as coordenadas do ponto (x,y):" ) ;
  scanf ( "%f %f ", &p->x, &p->y )  ;
}
```

Permite modificar o valor da variável `p`

Usando Ponteiros para Estruturas

```
int  main  ( ) {
   struct   ponto   p ;
   captura ( &p ) ;
   imprime ( &p ) ;
    return 0 ;
}
```

- Alocação dinâmica de estruturas

```
struct   ponto   *p ;
p =(struct ponto *)malloc ( sizeof ( struct  ponto ));
```

Usando *typedef*

- Definição de novos tipos
  - C permite criar nomes de tipos
  - É útil para abreviar nomes de tipos ou tipos complexos
  - Exemplo:

typedef float Real ; ⇒ Define um novo tipo "Real "

  - Definição de nomes de tipos para estruturas

```
struct   ponto {
     float    x ;
     float    y ;
} ;
typedef  struct  ponto  PONTO ;
```

⇒ Representa a estrutura ponto

Usando *typedef*

- Após a definição de novos tipos, pode-se declarar variáveis destes tipos

```
PONTO    p ;
```

- Pode-se definir uma estrutura e associar um nome de tipo a ela em um **único** comando

```
typedef   struct  ponto  {
        float   x  ;     float   y ;
}   PONTO  ;
```

Tipos Estruturados Mais Complexos

- Aninhamento de estruturas
  - Campos de uma estrutura podem ser outras estruturas previamente definidas
  - Exemplo:
    - Considere a estrutura PONTO e a distância entre dois pontos (x1, y1) e (x2, y2)

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

```
float  distancia ( PONTO  *p ,  PONTO  *q ){
   float d = sqrt((q->x - p->x )*(q->x - p->x)+
                  (q->y - p->y)*(q->y - p->y)) ;
   return  d ;
}
```

Tipos Estruturados Mais Complexos

- Como temos o tipo PONTO definido, então ele pode ser usado na definição da estrutura circulo

```
struct   circulo {
   PONTO   p ;        /* centro do circulo*/
   float   r ;     } ;   /* raio do circulo*/
typedef   struct   circulo   CIRCULO ;
```

```
/* Determina se um dado ponto está no circulo
int   interior   ( CIRCULO   *c ,   PONTO   *p ){
   float   d  =  distancia ( &c->p ,  p ) ;
   return   ( d <= c->r ) ;
}
```

Endereço do centro do círculo

Vetores de Estruturas

- Considere o cálculo de um centro geométrico de um conjunto de pontos

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n} \quad \text{e} \quad \bar{y} = \frac{\sum_{i=1}^{n} y_i}{n}$$

```
PONTO  centrogeometrico  ( int n , PONTO  *v ){
       int   i ;
       PONTO  p = { 0.0 , 0.0 } ;
       for  ( i = 0 ;   i < n  ;   i++ ){
              p.x  +=  v [ i ].x ;
              p.y  +=  v [ i ].y ;
       }
       p.x /=  n ;p.y /=  n ;
       return  p ;
}
```

Endereço inicial do vetor de pontos

Vetores de Estruturas

- A função *centrogeometrico* retorna uma estrutura

```
PONTO centrogeometrico(int n,PONTO* v);
```

  - No entanto, não é conveniente retornar estruturas grandes
    - Deve-se, quando possivel, escrever funções que retornem ponteiros para as estruturas

Vetores de Ponteiros para Estruturas

- São úteis quando temos de tratar um conjunto de elementos complexos
  - Exemplo: Tabela de alunos

    Matricula: número inteiro ;
    Nome: cadeia com 80 caracteres ;
    Endereço: cadeia com 120 caracteres ;
    Telefone: cadeia com 20 caracteres ;

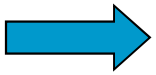
  - Em C podemos representar aluno:

```
struct aluno  {
       int  mat ;
       char  nome [81] ;
       char   end [121] ;
       char   tel [ 21 ] ;    } ;
typedef   struct  aluno   ALUNO ;
```

Vetores de Ponteiros para Estruturas

- Se montarmos a tabela de alunos usando um vetor com um número máximo de alunos:

```
#define MAX 100
ALUNO tab[max];
```



```
tab[i].mat =9912222;
```

  - Tipo ALUNO ocupa pelo menos 227(=4+81+121+21) bytes
  - Se for usado um número de alunos inferior ao máximo estimado, a declaração de um vetor dessa estrutura representa um desperdício significativo de memória
  - Uma solução é:

```
#define MAX 100
ALUNO* tab[max];
```

Usando Vetores de Ponteiros para Estruturas

```
#define   MAX   100
struct   aluno  {
      int   mat ;
      char   nome [81];
      char   end [121] ;
      char   tel [21] ;
  } ;
typedef  struct  aluno   ALUNO;
ALUNO   *tab [ MAX ] ;

void   inicializa (int n,ALUNO  **tab){
    int  i ;
    for(i=0 ;i< n;i++)
      tab [i] = NULL ;
}
```

Todas as posições do vetor de ponteiros guardam endereços nulos

Usando Vetores de Ponteiros para Estruturas

```
void preenche(int n,ALUNO **tab,int i ){
   if ((i < 0)||(i  >= n)){
      printf ("Indice fora do limite do vetor \n");
      exit(1);
   }
   if (tab[i] == NULL ){
      tab[i]= ( ALUNO*) malloc (sizeof(ALUNO));
      printf ("\nEntre com a matricula:");
      scanf ("%d", &tab[i]->mat) ;
      printf ("\nEntre com o nome:") ;
      scanf ("%80[^\n]", tab[i]->nome) ;
      printf ("\nEntre com o endereco:") ;
      scanf ("%120[^\n]", tab[i]->end) ;
      printf ("\nEntre com o telefone:") ;
      scanf ("%20[^\n]", tab[i]->tel) ;
   }
}
```

Espaço alocado para um novo aluno e endereço é armazenado no vetor

Usando Vetores de Ponteiros para Estruturas

```
void retira(int n,ALUNO** tab,int   i ) {
   if ((i < 0)||(i >= n)){
      printf("Indice fora do limite do vetor\n");
      exit (1); /* aborta o programa */
      }
   if (tab[i] != NULL) {
      free(tab[i]);
      tab[i] = NULL;
   }
}
```

Usando Vetores de Ponteiros para Estruturas

```
void imprime(int n,ALUNO** tab,int i ){
   if ((i < 0)||(i >= n)){
      printf ("Indice fora do limite do vetor \n" );
      exit (1) ;     /* aborta o programa */
   }
   if (tab[i]!= NULL){
      printf ("Matricula: %d\n",tab[i]->mat);
      printf ("Nome: %s\n",tab[i]->nome);
      printf ("Endereço: %s\n",tab[i]->end);
      printf ("Telefone: %s\n",tab[i]->tel);
   }
}
```

Usando Vetores de Ponteiros para Estruturas

```
void imprime_tudo(int n,ALUNO** tab){
   int i;
   for(i = 0; i < n; i++)
      imprime(n,tab,i);
 }
```

Usando Vetores de Ponteiros para Estruturas

```
#include <stdio.h>
 int main(){
   ALUNO *tab[10];
   inicializa(10,tab);
   preenche(10,tab,0);
   preenche(10,tab,1);
   preenche(10,tab,2);
   imprime_tudo(10,tab);
   retira(10,tab,0);
   retira(10,tab,1);
   retira(10,tab,2);
   return   0 ;
 }
```

Tipo Estruturado Union

- Armazena valores heterogêneos em um mesmo espaço de memória
- Apenas um único elemento de uma união pode estar armazenado em um determinado instante pois a atribuição a um campo da união sobrescreve o valor anteriormente atribuído a qualquer outro campo
  - A definição de um tipo *união* é dada por:

```
union   exemplo  {
 int i ;
 char c ;
}
```

  - Declaração de uma variável deste tipo

```
union exemplo v ;
```

Tipos Estruturado Union

- Outro exemplo de declaração de variável

```
union exemplo  {
   int i ;
   char  c ;
}

union exemplo v2;
```

Ocupa o espaço necessário para armazenar o maior de seus campos que neste caso é um inteiro.

- Acesso aos elementos de uma união
  - Diretamente (Operador " . "): `v2.i = 10 ;`
  - Via um ponteiro (Operador " -> "):

```
union exemplo *v3 ; v3->c = ' x ';
```

Tipos Estruturado Enumeração

- É uma forma mais elegante de organizar constantes
  - Considere a criação de um tipo *booleano*

```
#define  FALSE  0
 #define  TRUE  1
 typedef int BOOL;/*pode armazenar qualquer inteiro*/
```

  - Criação de um tipo *booleano* usando o tipo enumeração

```
enum    bool  {
    FALSE,/*O valor 0 é atribuído a FALSE */
    TRUE  /*O valor 1 é atribuído a TRUE */
} ;
typedef  enum  bool   BOOL ;
```

Tipos Estruturado Enumeração

- Criação de um tipo *booleano* usando o tipo enumeração (Cont.)
  - Os valores são atribuídos conforme a ordem em que os elementos são definidos em uma enumeração

```
enum    bool  {
  TRUE = 1, /*O valor 1 é atribuído a TRUE */
  FALSE = 0 /*O valor 0 é atribuído a FALSE */
} ;
 typedef  enum  bool   BOOL ;

 BOOL   result ;
```

Declara uma variável do tipo BOOL