Estrutura de Dados

Hamilton José Brumatto

Bacharelado em Ciências da Computação - UESC

11 de abril de 2016

Vetores

Sequência do mesmo tipo de tamanho finito

- Vetores é uma sequência de armazenamento de um mesmo tipo em memória.
- O tipo de dado armazenado pode ser primitivo ou abastrato.
- Para acessar os valores armazenados em um vetor são usados índices.
  - O tamanho definido do vetor representa quantos elementos ele possui
  - Cada elemento é representado por um índice no vetor.
- O tamanho do vetor pode (em algumas linguagens) ser definido na declaração do vetor ao se atribuir valores.

Exemplo básico de vetores

```
int main(int argc, char **args)
{
    int i;
    int vint1[10];
    int vint2[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

    for(i = 0; i < 10; i++)
        vint1[9-i] = vint2[i];
    for(i = 0; i < 10; i++)
        printf("%d..",vint1[i]);
    printf("\n");
    return 0;
}
```

Saída do exemplo acima

```
9..8..7..6..5..4..3..2..1..0..
```

Exemplo básico (será?) de vetores

```
typedef struct st_complexo
{
    double real;
    double imag;
} complexo;
int main(int argc, char **args)
{
    int i;
    complexo c1[3];
    complexo c2[] = {{2, 3}, {3, 4}, {1, 9}};
    for(i = 0; i < 3; i++)
    {
        c1[2-i].real = c2[i].imag;
        c1[2-i].imag = c2[i].real;
    }
    for(i = 0; i < 3; i++)
        printf("(%f + i%f)..",c1[i].real,c1[i].imag);
    printf("\n");
    return 0; }
```

Saída do exemplo acima

```
(9.000000 + i1.000000)..(4.000000 + i3.000000)..(3.000000 + i2.000000)..
```

Manipulando vetores a partir de ponteiros

- Ao declarar um vetor, como no exemplo: `int v[5];`
  - É alocado na memória um espaço para 5 inteiros, em sequência.
  - a variável `v` representa o endereço da primeira posição da memória.
  - este mesmo endereço é obtido diretamente pelo operador endereço & sobre o primeiro elemento do vetor: `&v[0]`.
  - `v` está armazenada exatamente no endereço que ela guarda.
- Podemos usar ponteiros para fazer referências a estes endereços.
- Uma vez declarado um ponteiro para o tipo inteiro (tipo do vetor): `int *p;`
  - Ele guarda o endereço para um elemento do vetor.
  - Incrementar o ponteiro de um, incrementa para o endereço do próximo elemento, automaticamente no tamanho do espaço ocupado por cada elemento.

Exemplo de endereçamento por ponteiros

```
int main(int argc, char **args) {
      int v[5] = {0, 1, 2, 3, 4};
      int *p;
      p = v;
      printf("Endereço do vetor: %p\n",p);
      p = &v[0];
      printf("Endereço do primeiro elemento: %p\n",p);
      printf("Valor do primeiro elemento: %d\n",*p);
      p++;
      printf("Endereço do segundo elemento: %p\n",p);
      printf("Valor do segundo elemento: %d\n",*p);
      p = (int *) &v;
      printf("Endereço da variável v: %p\n",p);
      return 0;
}
```

Saída do programa acima:

```
Endereço do vetor:  0x7fffe4336e30
Endereço do primeiro elemento:  0x7fffe4336e30
Valor do primeiro elemento:  0
Endereço do segundo elemento:  0x7fffe4336e34
Valor do segundo elemento:  1
Endereço da variável v:  0x7fffe4336e30
```

Ponteiro para vetores de tipos abstratos

```
typedef struct st_racional {
    int num, den;
} racional;
int main(int argc, char **args) {
    racional r[5] = {{1, 2}, {2, 3}, {3, 4}, {4, 5}, {5, 6}};
    racional *pr;
    pr = r;
    printf("Endereço do vetor: %p\n",pr);
    printf("Valor do primeiro elemento: %d/%d\n",pr->num,pr->den);
    pr++;
    printf("Endereço do segundo elemento: %p\n",pr);
    printf("Valor do segundo elemento: %d/%d\n",(*pr).num,(*pr).den);
    return 0;
}
```

Saída do programa acima

```
Endereço do vetor:  0x7fffaedb6460
Valor do primeiro elemento:  1/2
Endereço do segundo elemento:  0x7fffaedb6468
Valor o segundo elemento:  2/3
```

Um pouco de abstração

```
class racional {
  private:
    int num, den;

  public:
    racional() {
      num = 0;
      den = 1;
    }

    racional(int n, int d) {
      num = n;
      den = d;
    }

    int numerador() {
      return num;
    }

    int denominador() {
      return den;
    }

    racional operator=(racional rop) {
      num = rop.num;
      den = rop.den;
      return *this;
    }
};
```

Usando a classe racional

```
#include <iostream>
    using namespace std;

int main(int argc, char **args) {
    racional r[3];
    racional *pr;

    r[0] = racional(1,2);
    r[1] = racional(2,3);
    r[2] = racional(3,4);

    pr = r;
    cout << "Endereço do vetor: " << r << endl;
    pr = &r[0];
    cout << "Endereço do primeiro elemento: " << pr << endl;
    cout << "Valor do primeiro elemento: " << r[0].numerador() << "/" <<
r[0].denominador() << endl;
    pr++;
    cout << "Endereço do segundo elemento: " << pr << endl;
    cout << "Valor do segundo elemento: " << pr->numerador() << "/" <<
pr->denominador() << endl;
```

Resultado do programa

```
Endereço do vetor:  0x7fffa5dca050
Endereço do primeiro elemento:  0x7fffa5dca050
Valor do primeiro elemento:  1/2
Endereço do segundo elemento:  0x7fffa5dca058
Valor do segundo elemento:  2/3
```

- Somente foi possível usar o código: `r[0] = racional(1,2);`, pois:
  - Foi definido o operador $=$. Ele permite atribuir um racional completo.
  - Na ausência deste operador, teríamos de criar os métodos: `setNumerador(int)` e `setDenominador(int)`.
  - Ficaria assim: `r[0].setNumerador(1); r[0].setDenominador(2);`
- Na ausência do operador, poderíamos fazer: `racional *r[3];` e atribuir endereços, usando o criador new para atribuir um vetor.
- Como ficaria o ponteiro `pr` ????

Neste código, a classe Racional não possui definição `operador=()`

```
int main(int argc, char **args) {
    racional *r[3];
    racional **pr;

    r[0] = new racional(1,2);
    r[1] = new racional(2,3);
    r[2] = new racional(3,4);

    pr = r;
    cout << "Endereço do vetor: " << r << endl;
    pr = &r[0];
    cout << "Endereço do primeiro elemento: " << pr << endl;
    cout << "O que contém o primeiro elemento: " << r[0] << endl;
    cout << "Valor apontado pelo primeiro elemento: " << r[0]->numerador() <<
"/" << r[0]->denominador() << endl;
    pr++;
    cout << "Endereço do segundo elemento: " << pr << endl;
    cout << "O que contém o segundo elemento: " << *pr << endl;
    cout << "Valor apontado pelo segundo elemento: " << (*pr)->numerador()
<< "/" << (*pr)->denominador() << endl;
    return 0;
```

Resultado do programa

```
Endereço do vetor:  0x7fffac3d9870
Endereço do primeiro elemento:  0x7fffac3d9870
O que contém o primeiro elemento:  0x1ff9010
Valor apontado pelo primeiro elemento:  1/2
Endereço do segundo elemento:  0x7fffac3d9878
O que contém o segundo elemento:  0x1ff9030
Valor apontado pelo segundo elemento:  2/3
```

- Agora `r` é um vetor de ponteiros para `racional`
- O conteúdo do vetor são endereços que apontam para objetos `racional`
- Então `pr` agora é um ponteiro para (ponteiro para racional), por isso: **
- Aonde `pr` aponta `(*pr)` é um endereço de um racional.
- Logo, `(*pr)->denominador()` ou `(*(*pr)).denominador()`
- Esta é a abordagem usada na linguagem Java, só que de forma transparente para o programador.

Solução Java. Os ponteiros estão escondidos

```
public class Racional {
   private int num, den;
   public Racional(int n, int d) {
     num = n;
     den = d;
   }
   public int numerador() {
     return num;
   }
   public int denominador() {
     return den;
   }
   public static void main(String[] args) {
     Racional[] r;
     r = new Racional[3];
     r[0] = new Racional(1,2);
     r[1] = new Racional(2,3);
     r[2] = new Racional(3,4);
     System.out.println("Valor do Primeiro Elemento: "+r[0].numerador()+" /"+r[0].denominador());
     System.out.println("Valor do Segundo Elemento: "+r[1].numerador()+" /"+r[1].denominador());
     return;
   }
}
```

Resultado

```
Valor do Primeiro Elemento:  1/2
Valor do Segundo Elemento:  2/3
```

O básico é simples

```
void troca(int[3]);

int main(int argc, char **args) {
    int i;
    int vint[3] = {0, 1, 2};
    troca(vint);
    for(i = 0; i < 3; i++)
        printf("%d..",vint[i]);
    return 0;
}
void troca(int v[3]) {
    int aux;
    aux = v[2];
    v[2] = v[0];
    v[0] = aux;
    return;
}
```

Resultado:

```
2..1..0..
```

Podemos evitar especificar o tamanho do vetor como parâmetro

```
void troca(int*);

int main(int argc, char **args) {
    int i;
    int vint[3] = {0, 1, 2};
    troca(vint);
    for(i = 0; i < 3; i++)
        printf("%d..",vint[i]);
    return 0;
}
void troca(int *v) {
    int aux;
    aux = v[2];
    v[2] = v[0];
    v[0] = aux;
    return;
}
```

Resultado:

```
2..1..0..
```

Revendo o conceito de vetores e ponteiros

- A diferença entre ambos exemplos anteriores é que como parâmetro especificamos: `int *v`
- Ou seja, `v` é um ponteiro para inteiro que tem o endereço de `vint`
- Ainda assim usamos `v[0]`.
- Será que funciona somente como parâmetro? Ou sempre?
- É mais interessante que isto. Veja o próximo código.

A ordem das parcelas não altera a soma!

```
int main(int argc, char **args) {
    int vint[3] = {0, 1, 2};
    int *v;

    v = vint;

    printf("Segundo elemento: %d\n", vint[1]);
    printf("Segundo elemento: %d\n", v[1]);
    printf("Segundo elemento: %d\n", *(v+1));
    printf("Segundo elemento: %d\n", *(1+v));
    printf("Segundo elemento: %d\n", 1[v]);
    return 0;
}
```

Resultado

```
Segundo elemento:  1
Segundo elemento:  1
Segundo elemento:  1
Segundo elemento:  1
Segundo elemento:  1
```

Considerações finais

- A passagem de parâmetros de tipos abstratos é semelhante aos tipos primitivos
- Para isto é necessário definir um tipo (`typedef`), pois este será especificado como tipo do parâmetro.
- Há um texto anexo no material explorando mais o uso de ponteiros na passagem de parâmetros em uma função de parâmetros genéricos, vale a pena ler.

Tarefa:Considere o seguinte algoritmo

Entrada: : Vetor A de elementos em ordem qualquer
Saída: : Vetor de elementos em ordem crescente

```
Algoritmo OrdenacaoPorTroca(A)
    fim ← falso
    enquanto ¬fim faça
        fim ← verdade
        para j ← 2 até tamanho[A] faça
            se A[j] < A[j − 1] então
                aux ← A[j]
                A[j] ← A[j − 1]
                A[j − 1] ← aux
                fim ← falso
            fim do se
        fim do para
    fim do enquanto
retorne A
fim do Algoritmo
```

Para o algoritmo da página anterior:

- Faça um programa que leia da entrada um valor que indique o número de elementos e em seguida todos elementos.
- Armazene os elementos na forma de um vetor, aplique o algoritmo descrito e imprima o resultado.
- Considere:
  - Os elementos são números (inteiros ou reais) escolha sua.
  - Os elementos são números racionais
  - Os elementos são números complexos e a ordenação se dá pelo seu módulo: $\sqrt{\text{real}^2 + \text{imag}^2}$