

Vetores

Fundamentos

Prof. Edson Alves - UnB/FGA

2020

1. Vetores em C/C++
2. Tipos de dados abstratos

Vetores em C/C++

Características dos vetores

- Os vetores (*arrays*) tem suporte nativo na maioria das linguagens de programação
- O tamanho (número de elementos) do vetor tem que ser conhecido em tempo de compilação
- Os elementos do vetor são armazenados na memória de forma sequencial (linear)
- O acesso aleatório é imediato: qualquer posição do vetor pode ser lida ou escrita em $O(1)$
- Porém a inserção e remoção aleatória é lenta: os elementos tem que ser transpostos no momento da inserção/remoção (complexidade $O(N)$)

Declaração estática de vetores

Sintaxe para declaração de vetores

```
tipo_do_dado nome_do_vetor[N];
```

- Na sintaxe acima, N é o número de elementos do vetor, e deve ser uma constante
- Os vetores armazenam elementos do mesmo tipo, conforme o que foi especificado em sua declaração
- Vale recordar que o primeiro elemento do vetor tem índice 0 (zero), e o último tem índice $N - 1$
- Um erro comum é usar uma variável para representar o valor de N

Exemplo de uso de vetores

```
1 /* Compile com a flag -lncurses */
2 #include <ncurses.h>
3 #include <ctype.h>
4
5 int found[4096], i;
6 char word[] = "TESTE", uppercase[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
7 typedef enum { STARTING, RUNNING, WAITING, VICTORY, GAME_OVER } State;
8
9 void print(int tries, State state) {
10     const char header[] = "Descubra a palavra secreta:";
11
12     clear();
13     mvprintw(1, 2, header);
14
15     for (i = 0; word[i]; ++i) {
16         addch(' ');
17         found[i] ? addch(word[i] | A_BOLD) : addch('_');
18     }
19
20     mvprintw(3, 0, "Chances restantes: %d", tries);
21     mvprintw(4, 0, "Letras disponíveis:");
```

Exemplo de uso de vetores

```
23     for (i = 0; uppercase[i]; ++i) {
24         if (uppercase[i] != '*')
25             printf(" %c", uppercase[i]);
26     }
27
28     mvprintw(6, 0, "Digite uma das letras disponíveis: ");
29
30     if (state == VICTORY)
31         mvprintw(8, 10, "Você descobriu a palavra secreta!");
32     else if (state == GAME_OVER)
33         mvprintw(8, 10, "Game Over!");
34
35     refresh();
36 }
37
38 State update(State state, int *tries) {
39     int left = 0, hits = 0, c;
40
41     if (state == STARTING)
42         return RUNNING;
43 }
```

Exemplo de uso de vetores

```
44     c = getch();
45
46     if (c < 'a' || c > 'z' || uppercase[c - 'a'] == '*')
47         return WAITING;
48
49     uppercase[c - 'a'] = '*';
50
51     for (i = 0; word[i]; ++i) {
52         if (!found[i] && word[i] == toupper(c)) {
53             ++hits;
54             found[i] = 1;
55         }
56
57         left += (!found[i] ? 1 : 0);
58     }
59
60     if (left == 0)
61         return VICTORY;
62     else if (hits == 0)
63         *tries -= 1;
64
```


Exemplo de uso de vetores

```
65     if (*tries == 0)
66         return GAME_OVER;
67
68     return RUNNING;
69 }
70
71 void init() {
72     initscr();
73     noecho();
74 }
75
76 void close() {
77     getch();
78     endwin();
79 }
80
```

Exemplo de uso de vetores

```
81 int main()
82 {
83     int tries = 7;
84     State state = STARTING;
85
86     init();
87
88     while (state != VICTORY && state != GAME_OVER) {
89         state = update(state, &tries);
90         print(tries, state);
91     }
92
93     close();
94
95     return 0;
96 }
```

Declaração de vetores dinâmicos

- Também é possível alocar um vetor dinamicamente, postergando o conhecimento do seu tamanho para a execução
- Em linguagem C, um vetor pode ser alocado dinamicamente de duas maneiras:

```
tipo *nome = (tipo *) malloc(sizeof(tipo) * tamanho);  
tipo *nome = (tipo *) calloc(tamanho, sizeof(tipo));
```

- Em linguagem C++, um vetor pode ser alocado através do operador **new**:

```
tipo *nome = new tipo[tamanho];
```

- A memória alocada para o vetor em C deve ser liberada através da função `free()`
- Já em C++ deve ser usado o operador **delete** []

Exemplo de uso de vetores dinâmicos

```
1 #include <iostream>
2 #include <cstdlib>
3
4 int* novo_jogo(int N) {
5     int *ns = new int[N];
6
7     if (!ns) return nullptr;
8
9     for (int i = 0; i < N; i++) {
10         ns[i] = (rand() % 60) + 1;
11
12         for (int j = 0; j < i; j++) {
13             if (ns[j] == ns[i]) {
14                 i--;
15                 break;
16             }
17         }
18     }
19
20     return ns;
21 }
```

Exemplo de uso de vetores dinâmicos

```
23 int main() {
24     srand(time(NULL));
25     int N;
26
27     do {
28         printf("Quantos numeros serão sorteados (entre 6 e 15)? ");
29         std::cin >> N;
30     } while (N < 6 || N > 15);
31
32     auto ns = novo_jogo(N);
33
34     if (ns) {
35         std::cout << "Numeros sorteados: ";
36         for (int i = 0; i < N; i++)
37             printf("%d%c", ns[i], " \n"[i + 1 == N]);
38
39         delete [] ns;
40     }
41
42     return 0;
43 }
```

Tamanho de um vetor

- Um vetor, seja alocado estaticamente, seja alocado dinamicamente, não guarda em si informações sobre o seu tamanho
- Como consequência da afirmação anterior, ao passar um vetor como parâmetro de uma função é necessário ou informar o tamanho ou definir um elemento delimitador (sentinela)
- Uma forma de evitar este problema é criar um novo tipo de dado que armazene os elementos do vetor e seu tamanho
- Importante notar que, no caso de vetores (*arrays*), o tamanho físico (capacidade, número máximo de elementos que ele comporta) coincide com o tamanho lógico (tamanho, número de elementos efetivamente preenchidos)
- Esta superposição de conceitos pode gerar problemas de interpretação (por exemplo, ler um elemento que não deveria ser considerado) e alinhamento de memória (pois as alocações precisam indicar a quantidade exata a ser usada)

Exemplo de uma estrutura para vetores

```
1 #ifndef FLOAT_VECTOR_H
2 #define FLOAT_VECTOR_H
3
4 typedef struct _FloatVector {
5     float *data;
6     int size;
7 } FloatVector;
8
9 extern float max(const FloatVector *v);
10 extern float min(const FloatVector *v);
11
12 #endif
```

Exemplo de uma estrutura para vetores

```
1 #include <float.h>
2 #include <float_vector.h>
3
4 static float limits(const FloatVector *v, char limit) {
5     register int i;
6     float a = FLT_MAX, b = -FLT_MAX;
7
8     for (i = 0; i < v->size; i++) {
9         a = (v->data[i] < a ? v->data[i] : a);
10        b = (v->data[i] > b ? v->data[i] : b);
11    }
12    return (limit == 'a' ? a : b);
13 }
14
15 float min(const FloatVector *v) {
16     return limits(v, 'a');
17 }
18
19 float max(const FloatVector *v) {
20     return limits(v, 'b');
21 }
```


Exemplo de uma estrutura para vetores

```
1 #include <stdlib.h>
2 #include <stdio.h>
3
4 #include <float_vector.h>
5
6 int main()
7 {
8     FloatVector v;
9     int i;
10
11     printf("Informe a quantidade de valores a serem inseridos: ");
12     scanf("%d", &v.size);
13
14     if (v.size < 1) {
15         fprintf(stderr, "Nada a fazer!\n");
16         return -1;
17     }
18
19     v.data = (float *) malloc(sizeof(float)*v.size);
20
```

Exemplo de uma estrutura para vetores

```
21  if (!v.data) {
22      fprintf(stderr, "Sem memoria\n");
23      return -2;
24  }
25
26  for (i = 0; i < v.size; i++) {
27      printf("Informe o valor %d: ", i + 1);
28      scanf("%f", &v.data[i]);
29  }
30
31  printf("O maior valor informado foi %3.2f\n", max(&v));
32  printf("O menor valor informado foi %3.2f\n", min(&v));
33
34  free(v.data);
35
36  return 0;
37 }
```

Tipos de dados abstratos

Tipos de dados abstratos

- Os tipos de dados abstratos (*Abstract Data Type* – *ADT*) são definidos pelas operações que agem sobre estes dados, e não pela implementação destas operações
- Conhecidas as operações, é possível escolher a implementação mais eficiente para cada caso
- Em alguns casos também é preciso determinar as restrições que cada operação possui, sejam restrições de comportamento ou de complexidade

Operações típicas de tipos de dados abstratos

Operação	Descrição
<code>create()</code>	cria uma nova instância S
<code>initialize(S)</code>	prepara a instância S para o seu estado inicial, permitindo seu uso em operações subsequentes
<code>free(S)</code>	libera a memória utilizada pela instância S
<code>empty(S)</code>	retorna verdadeiro se a instância S está vazia
<code>size(S)</code>	lista o número de elementos contidos em S
<code>compare(S, T)</code>	retorna verdadeiro se as instâncias S e T tem os mesmos elementos, na mesma ordem
<code>print(S)</code>	produz uma representação visual da instância S
<code>copy(S, T)</code>	faz com que a instância S fique com o mesmo estado que a instância T
<code>clone(S)</code>	cria uma nova instância T com o mesmo estado de S

- Contêiners são tipos de dados abstratos que representam coleções de outros objetos
- Além das operações `create` e `size`, os contêiners devem fornecer operações para:
 - remover todos os elementos de uma só vez (`clear`)
 - inserir novos elementos (`push`)
 - remover elementos (`pop`)
 - acesso aos elementos armazenados (`element`)
- A classe `vector` de C++ é um contêiner
- É possível criar um contêiner semelhante em C, implementando as operações listadas anteriormente
- Para tal, é preciso definir uma estrutura que contenha as variáveis necessárias para a implementação das funções que agirão sobre esta estrutura

Exemplo de uso da classe vector

```
1 #include <iostream>
2 #include <vector>
3
4 int main() {
5     std::vector<int> ns { 1, 1, 2, 3, 5, 8, 13, 21 };
6
7     std::cout << "ns tem " << ns.size() << " elementos\n";
8     ns.clear();
9     std::cout << "ns tem " << ns.size() << " elementos\n";
10
11     for (int i = 1; i <= 10; ++i)
12         ns.push_back(i);
13
14     ns.pop_back();
15
16     std::cout << "ns: ";
17     for (size_t i = 0; i < ns.size(); ++i)
18         std::cout << ns[i] << (i + 1 == ns.size() ? "\n" : " ");
19
20     return 0;
21 }
```

1. **DROZDEK**, Adam. *Algoritmos e Estruturas de Dados em C++*, 2002.
2. **KERNIGHAN**, Bryan; **RITCHIE**, Dennis. *The C Programming Language*, 1978.
3. **STROUSTROUP**, Bjarne. *The C++ Programming Language*, 2013.
4. C++ Reference¹.

¹<https://en.cppreference.com/w/>