Vector

Bài giảng môn Cấu trúc dữ liệu và giải thuật

Khoa Công nghệ thông tin
Trường Đại học Thủy Lợi

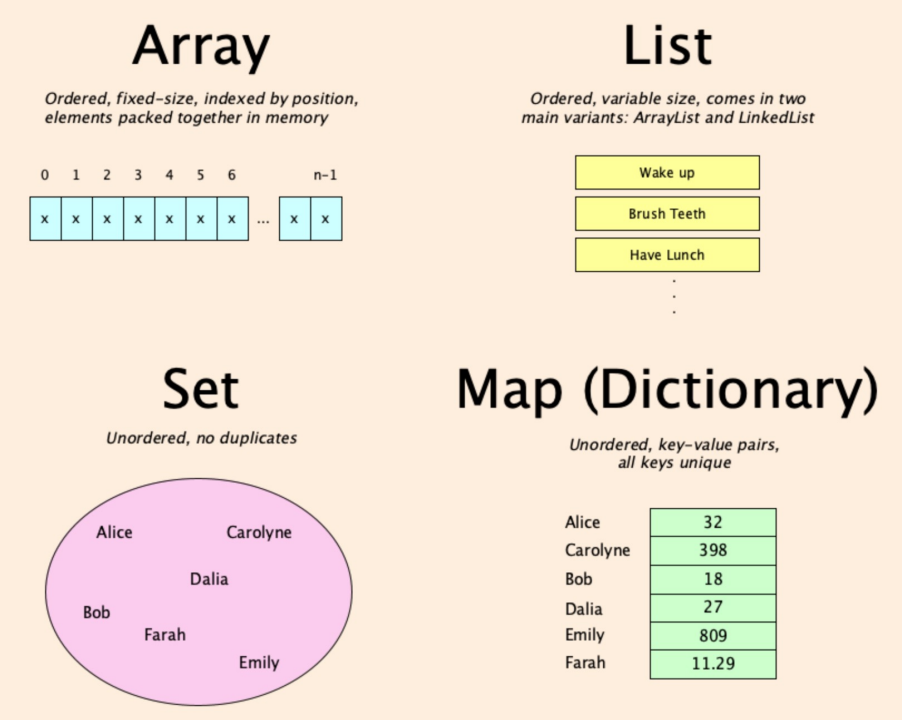
Nội dung

1. Cấu trúc dữ liệu là gì?
2. Vector
3. Chèn phần tử
4. Xóa phần tử
5. Thời gian chạy

1. Cấu trúc dữ liệu là gì?

Cấu trúc dữ liệu

- Là cách tổ chức dữ liệu trong bộ nhớ máy tính sao cho các thao tác xử lý dữ liệu (tìm, chèn, xóa…) hiệu quả hơn (nhanh hơn, tốn ít bộ nhớ hơn).
- Ví dụ cấu trúc dữ liệu:
  - Vector
  - Danh sách liên kết
  - Ngăn xếp/Hàng đợi
  - Cây
  - Bảng băm



Các bước cài đặt cấu trúc dữ liệu

```
// 1. Khai báo kiểu T của các phần tử trong cấu trúc dữ liệu.
//    Ở đây, T đang là int, nhưng có thể thay int bằng kiểu khác
//    tùy theo nhu cầu.
typedef int T;

// 2. Định nghĩa cấu trúc dữ liệu
struct <tên cấu trúc dữ liệu> { ... };

// 3. Khai báo các hàm khởi tạo, hủy, xử lý dữ liệu

// 4. Viết hàm main để chạy thử cấu trúc dữ liệu

// 5. Định nghĩa hàm khởi tạo cấu trúc dữ liệu

// 6. Định nghĩa hàm hủy cấu trúc dữ liệu

// 7. Định nghĩa các hàm xử lý dữ liệu như tìm, chèn, xóa
```

2. Vector

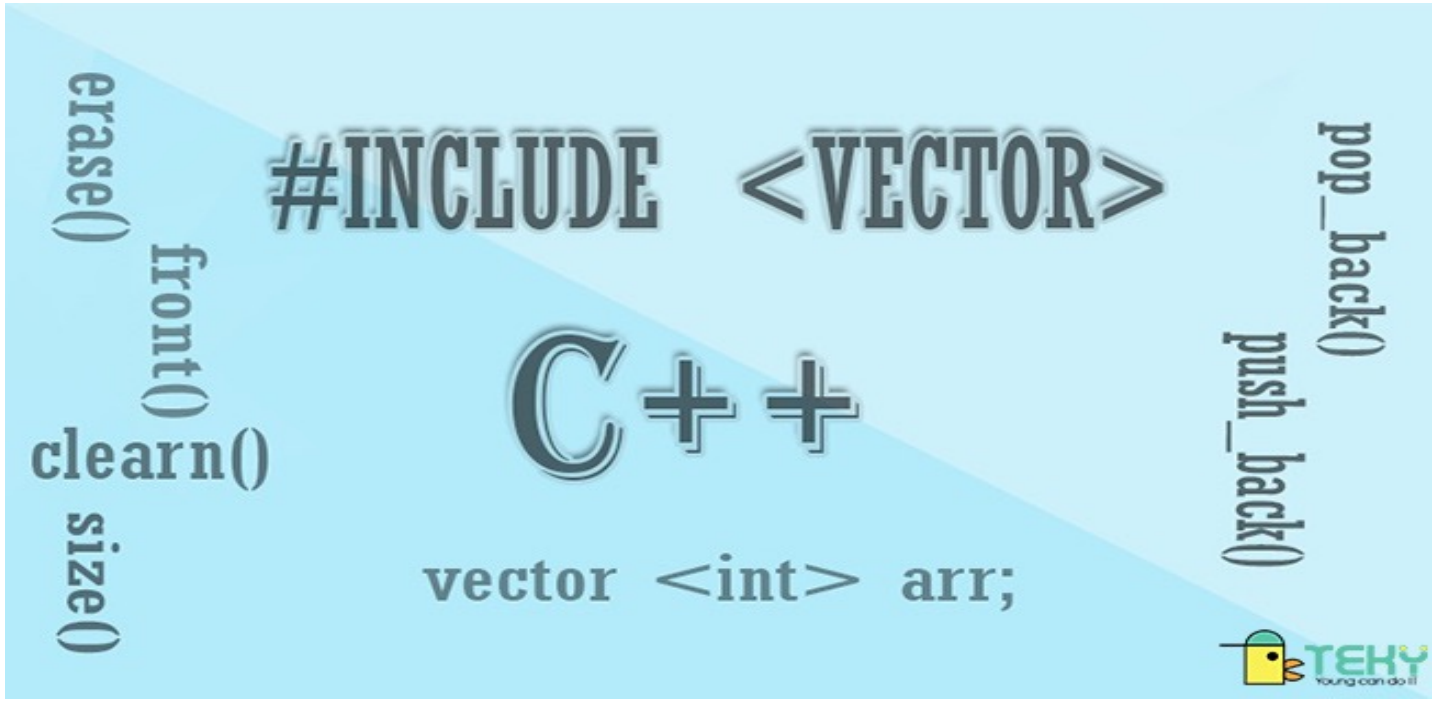
Vector

- Quản lý một dãy phần tử:
  - nằm liên tục trong bộ nhớ (như mảng một chiều);
  - *kích thước thay đổi được* (trong khi kích thước của mảng là cố định sau khi khai báo). Tức là, vector hoàn toàn có thể tự động nâng kích thước lên (phải resize).
  - Tự động giải phóng bộ nhớ khi vector không cần thiết.
  - Tự động ghi nhớ độ dài của mình

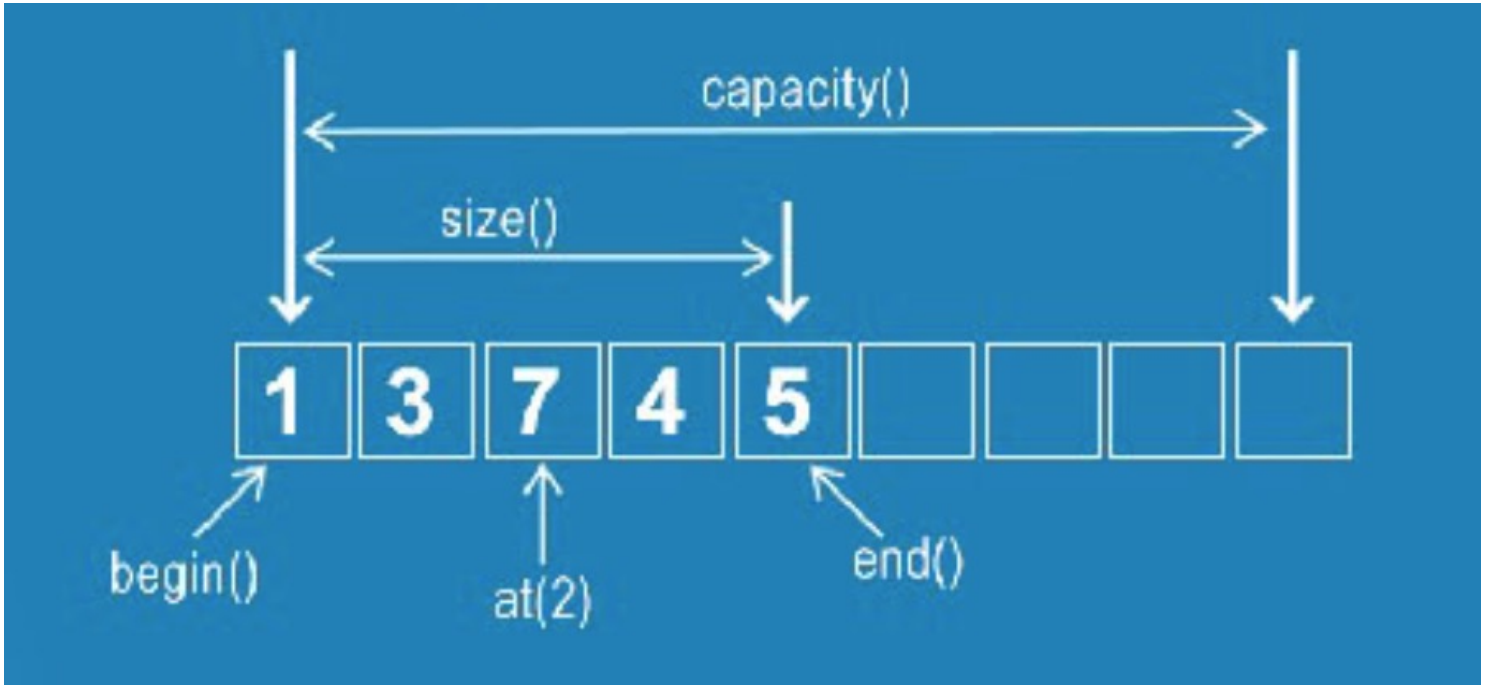
Vector

- Các thao tác chính:
  - Chèn và xóa phần tử ở cuối vector
  - Chèn và xóa phần tử ở giữa vector (bao gồm đầu vector)
  - Lấy kích thước vector (số phần tử hiện có)
  - Truy nhập phần tử dùng chỉ số
  - Dung lượng tổng các ô nhớ của vector (Hàm capacity)
  - Sắp xếp vector một cách nhanh nhất theo chiều tăng dần (hàm sort)
  - Kiểm tra xem vector có trống hay không (hàm empty)

Vector


#INCLUDE <VECTOR>
C++
vector <int> arr;
erase()
front()
clearn()
size()
pop_back()
push_back()
TEKY

Vector



Vector

- Thêm phần tử vào cuối

truyencontro.cpp | luongthang.cpp | vector.cpp | CTDL

vector.cpp > main(void)

```cpp
#include <iostream>
#include <vector>
using namespace std;
int main(void)
{
    vector <int> v;
    v.push_back(10);
    v.push_back(20);
    v.push_back(30);
    cout << "Size of vector is " << v.size() << endl;

}
```

Vector

- Duyệt phần tử vector thông qua phạm vi

```
for ( auto & x : v) {
    // Xử lý
}
```

Trong đó:

- `v` là tên vector
- `x` là tên một biến dùng để gán từng phần tử được lấy từ vector
- `auto` là kiểu suy luận giúp tự xác định kiểu dữ liệu của giá trị lấy từ vector

Vector

- Duyệt phần tử vector thông qua phạm vi

vector2.cpp > main(void)

```cpp
#include <iostream>
#include <vector>
using namespace std;
int main(void)
{
    vector <int> v;
    v.push_back(10);
    v.push_back(20);
    v.push_back(30);
    cout << "Size of vector is " << v.size() << endl;
    for(auto x:v)
    {
        cout << x << endl;
    }
    return 0;
}

```

Vector

- Duyệt phần tử vector thông qua phạm vi

vector2.cpp > main(void)

```
1   #include <iostream>
2   #include <vector>
3   using namespace std;
4   int main(void)
5   {
6       vector <int> v;
7       v.push_back(10);
8       v.push_back(20);
9       v.push_back(30);
10      cout << "Size of vector is " << v.size() << endl;
11      for(int x:v)
12      {
13          cout << x << endl;
14      }
15      return 0;
16  }
17
18
19
```

Vector

- Duyệt phần tử vector bằng iterator
  - Các kiểu dữ liệu như vector, list, map đều sử dụng iterator để biến chúng thành các trình lặp để dễ dàng xử lý
  - iterator như một con trỏ trỏ đến phần tử trong vector

```
for(auto itr = v.begin(); itr != v.end(); ++itr) {
    // Xử lý
}
```

Trong đó:

- `v` là tên vector
- `itr` là tên iterator

Vector

vector3.cpp > main(void)

```cpp
#include <iostream>
#include <vector>
using namespace std;
int main(void)
{
    vector <int> v;
    v.push_back(10);
    v.push_back(20);
    v.push_back(30);
    cout << "Size of vector is " << v.size() << endl;
    for(vector <int>::iterator it =v.begin();it != v.end();++it){
        cout << *it << endl;
    }
    return 0;
}

```

Vector

vector5.cpp > main(void)

```cpp
#include <iostream>
#include <vector>
using namespace std;
int main(void)
{
    int n;
    cin >> n;
    vector <int> v(n);
    for(int i=0;i<n;i++)
    {
        cout << "Nhap phan tu thu " << i <<" = ";
        cin >> v[i];
    }

    cout << "Size of vector is " << v.size() << endl;
    for(auto itr = v.begin(); itr != v.end(); ++itr) {
        cout << *itr << endl;
    }
    return 0;
}
```

Vector

vector5.cpp > main(void)

```cpp
#include <iostream>
#include <vector>
using namespace std;
int main(void)
{
    int n;
    cin >> n;
    vector <int> v(n);
    for(int i=0;i<n;i++)
    {
        cout << "Nhap phan tu thu " << i <<" = ";
        cin >> v[i];
    }

    cout << "Size of vector is " << v.size() << endl;
    for(auto itr = v.begin(); itr != v.end(); ++itr) {
        cout << *itr << endl;
    }
    return 0;
}
```

Vector

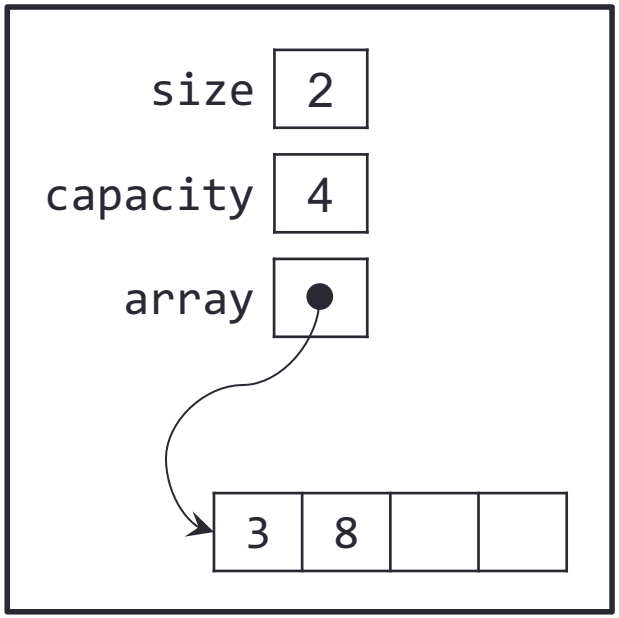
vector6.cpp > main(void)

```cpp
#include <iostream>
#include <vector>
using namespace std;
int main(void)
{
    int n;
    cin >> n;
    vector <int> v;
    int temp;
    for(int i=0;i<n;i++)
    {
        cout << "Nhap phan tu thu " << i <<" = ";
        cin >> temp;
        v.push_back(temp);
    }

    cout << "Size of vector is " << v.size() << endl;
    for(auto itr = v.begin(); itr != v.end(); ++itr) {
        cout << *itr << endl;
    }
    return 0;
}
```

Cài đặt vector

Chú ý: Lớp `vector` *trong thư viện chuẩn C++ dùng chữ "v" thường.*

```
// Khai báo kiểu phần tử
typedef int T;

// Định nghĩa cấu trúc Vector
struct Vector {
    // Kích thước vector (số phần tử
    // hiện có)
    int size;

    // Dung lượng vector (chứa được tối
    // đa bao nhiêu phần tử?)
    int capacity;

    // Con trỏ tới mảng chứa các phần tử
    T * array;
};
```



Hàm khởi tạo và hàm hủy

```
// initCapacity là dung lượng ban đầu của vector và
// có giá trị ngầm định bằng 16.
void vecInit(Vector & vec, int initCapacity = 16) {
    vec.size = 0; // Ban đầu chưa có phần tử nào
    vec.capacity = initCapacity; // Khởi tạo dung lượng
    vec.array = new T[vec.capacity]; // Tạo mảng chứa phần tử
}

void vecDestroy(Vector & vec) {
    delete[] vec.array; // Xóa mảng (giải phóng bộ nhớ)
}
```

Sao chép vector

```
// Sao chép nội dung từ vec2 sang vec.
void vecCopy(Vector & vec, Vector & vec2) {
  if (&vec != &vec2) {                  // Ngăn cản tự sao chép
    vec.size = vec2.size;               // Đặt kích thước mới
    vec.capacity = vec2.capacity;       // Đặt dung lượng mới
    delete[] vec.array;                 // Xóa mảng cũ
    vec.array = new T[vec.capacity]; // Tạo mảng mới

    // Sao chép các phần tử từ vec2 sang vec
    for (int i = 0; i < vec.size; i++)
      vec.array[i] = vec2.array[i];
  }
}
```

Kích thước vector và truy nhập phần tử

```
// Lấy kích thước vector (số phần tử hiện có).
int vecGetSize(Vector & vec) {
    return vec.size;
}

// Kiểm tra vector có đang rỗng hay không.
bool vecIsEmpty(Vector & vec) {
    return (vec.size == 0);
}

// Truy nhập một phần tử thông qua chỉ số (index) của nó.
T vecGetElem(Vector & vec, int index) {
    return vec.array[index];
}
// Cập nhật một phần tử.
void vecSetElem(Vector & vec, int index, T newValue) {
    vec.array[index] = newValue;
}
```

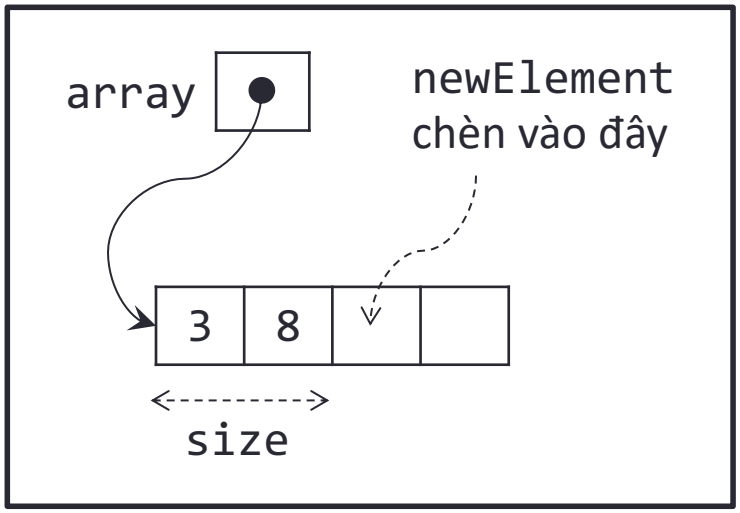
3. Chèn phần tử

Tăng dung lượng vector

```
// Đây là thao tác trợ giúp cho các thao tác chèn.
// newCapacity là dung lượng mới (phải lớn hơn kích thước).
void vecExpand(Vector & vec, int newCapacity) {
  if (newCapacity <= vec.size)
    return; // Thoát ra nếu dung lượng mới không đủ lớn

  T * old = vec.array; // Giữ lại địa chỉ mảng cũ
  vec.array = new T[newCapacity]; // Tạo mảng có chiều dài mới

  for (int i = 0; i < vec.size; i++)
    vec.array[i] = old[i]; // Sao chép mảng cũ sang mảng mới

  delete[] old;              // Xóa mảng cũ
  vec.capacity = newCapacity; // Đặt dung lượng mới
}
```

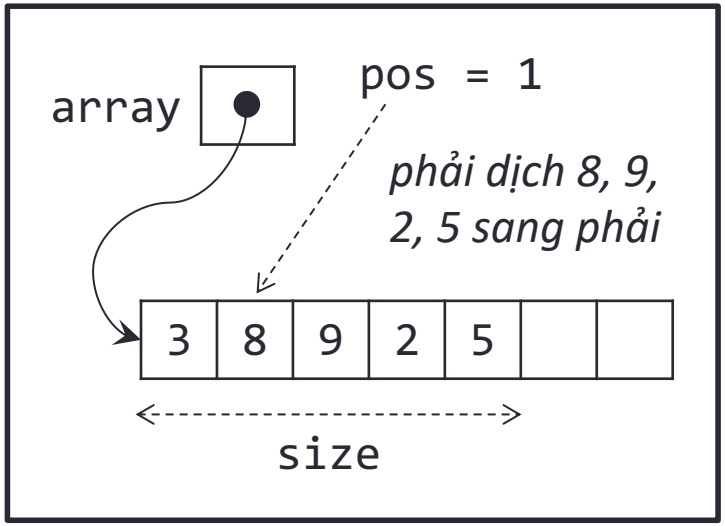
Chèn phần tử vào cuối vector

```
// newElement là phần tử mới cần chèn vào cuối vector.
void vecPushBack(Vector & vec, T newElement) {
    // Gấp đôi dung lượng nếu vector đã đầy
    if (vec.size == vec.capacity)
        vecExpand(vec, 2 * vec.capacity);

    // Chèn phần tử mới vào ngay sau phần tử cuối cùng
    vec.array[vec.size] =
        newElement;

    // Cập nhật kích thước
    vec.size++;
}
```



Chèn phần tử vào giữa vector

```
// pos (position) là vị trí chèn, có giá trị từ 0 đến size-1.
// newElement là phần tử mới cần chèn.
void vecInsert(Vector & vec, int pos, T newElement) {
    // Gấp đôi dung lượng nếu vector đã đầy
    if (vec.size == vec.capacity)
        vecExpand(vec, 2 * vec.capacity);

    // Dịch chuyển các phần tử ở pos và sau pos sang phải một vị trí;
    // phải quét ngược từ phải sang trái (for lùi) để tránh ghi đè.
    for (int i = vec.size; i > pos; i--)
        vec.array[i] = vec.array[i - 1];

    // Đặt phần tử mới vào vị trí chèn
    vec.array[pos] = newElement;

    // Cập nhật kích thước
    vec.size++;
}
```



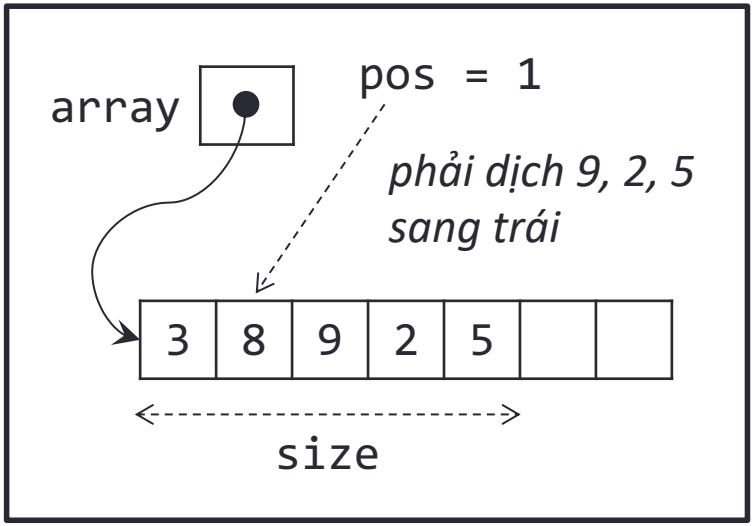
4. Xóa phần tử

Xóa phần tử ở cuối vector

```
// Xóa phần tử ở cuối vector.
void vecPopBack(Vector & vec) {
    vec.size--; // Giảm kích thước một đơn vị nghĩa
                // là “quên” phần tử cuối cùng.
}

// Xóa tất cả các phần tử.
void vecClear(Vector & vec) {
    vec.size = 0; // Đặt kích thước về 0 nghĩa là
                  // “quên” tất cả các phần tử.
}
```

Xóa phần tử ở giữa vector

```
// pos (position) là vị trí của phần tử cần xóa.
void vecErase(Vector & vec, int pos) {
    // Dịch các phần tử nằm sau vị trí xóa sang trái để
    // lấp đầy chỗ trống để lại do việc xóa.
    for (int i = pos; i < vec.size - 1; i++)
        vec.array[i] = vec.array[i + 1];

    // Cập nhật kích thước
    vec.size--;
}
```



5. Thời gian chạy

Phân tích thời gian chạy

- **Hàm khởi tạo**, **hàm hủy**: O(1)
- **Sao chép vector**: O(n) → vì phải sao chép n phần tử.
- **Lấy kích thước, kiểm tra rỗng, truy nhập phần tử**: O(1)
- **Tăng dung lượng**: O(n) → vì phải sao chép n phần tử.
- **Chèn vào cuối**: O(1)
- **Chèn vào giữa**: O(n) → vì phải dịch n phần tử sang phải trong trường hợp tồi nhất (chèn vào đầu vector).
- **Xóa ở cuối**: O(1)
- **Xóa tất cả**: O(1)
- **Xóa ở giữa**: O(n) → vì phải dịch n - 1 phần tử sang trái trong trường hợp tồi nhất (xóa phần tử đầu tiên).

Bài tập

1. Xét một vector đang có kích thước s_1 và dung lượng c_1, trong đó $s_1 \leq c_1$. Nêu các bước phải thực hiện để tăng dung lượng vector từ c_1 lên c_2, trong đó $c_1 < c_2$. Sau khi tăng dung lượng như vậy thì kích thước vector là bao nhiêu?
2. Xét một vector đang chứa các phần tử như sau:

   { 6, 5, 8, 2, 9, 7 }

   Giả thiết rằng vector chưa đầy và vị trí của các phần tử tính từ 0. Nêu các bước phải thực hiện để chèn giá trị X vào vị trí 3 trong vector.

Bài tập

3. Xét một vector đang chứa các phần tử như sau:

   { 8, 1, 9, 3, 4, 6 }

   Nêu các bước phải thực hiện để xóa phần tử ở vị trí 2 trong vector (vị trí tính từ 0).

4. Hỏi giữa chèn/xóa ở đầu vector và chèn/xóa ở cuối vector thì thao tác nào chạy nhanh hơn? Vì sao?

5. Giả sử ta phải bổ sung thao tác `truncate` vào vector nhằm cắt bỏ phần dung lượng dư thừa. Hãy đề xuất các bước cụ thể để thực hiện thao tác `truncate`.