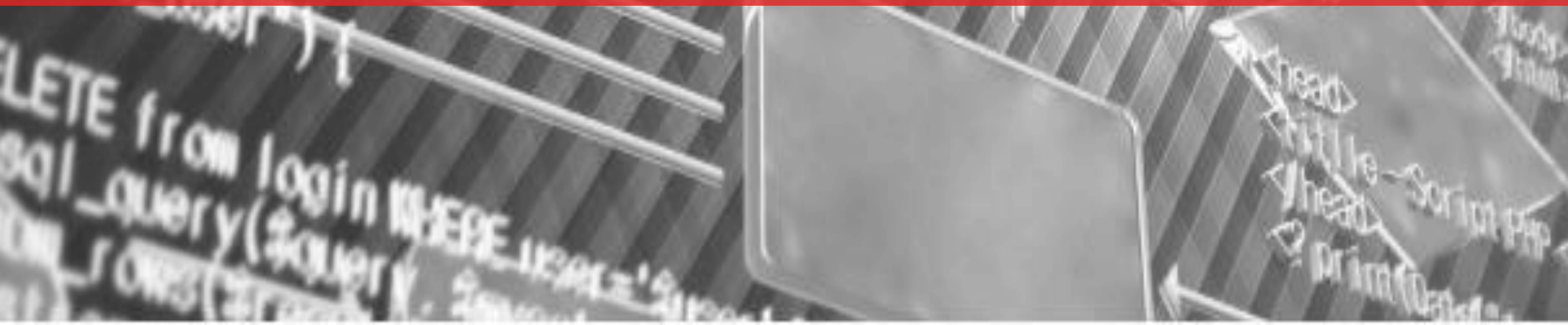
Bài 7_2
Thư viện STL
(Standard Template Library)



Khái niệm

- STL là thư viện chuẩn của C++, được xây dựng sẵn
- Cài đặt các cấu trúc dữ liệu và thuật toán thông dụng
- Bao gồm các lớp và hàm khuôn mẫu, cho phép làm việc với dữ liệu tổng quát
- Nằm trong một namespace có tên std
- Các phần chính:
  - Các lớp dữ liệu cơ bản: string, complex
  - Xuất nhập (IO)
  - Các lớp chứa (containers): list, vector, deque, stack, map, set,…
  - Duyệt phần tử của các lớp chứa (iterators)
  - Một số thuật toán thông dụng: tìm kiếm, so sánh, sắp xếp,…
  - Quản lý bộ nhớ, con trỏ
  - Xử lý ngoại lệ (exception handling)

Xử lý chuỗi

- `#include <string>`
- Lớp `string` cho chuỗi ASCII và `wstring` cho Unicode
- Các thao tác cơ bản: `+`, `+=` (nối chuỗi); `==`, `!=`, `>`, `<`, `>=`, `<=` (so sánh); `<<` (xuất), `>>` (nhập)
- Độ dài chuỗi: `int string::length() const`
- Chuỗi con: `string string::substr(int off, int count) const`
- Tìm chuỗi con: `int string::find(const char* str, int pos) const`
- Đổi sang chuỗi của C: `const char* string::c_str() const`
- Ví dụ:

```
string s1, s2("test123");
cin >> s1;
s1 += "123";
cout << (s2==s1 ? "same" : "different") << endl;

int pos = s2.find("est");
string s3 = s2.substr(pos, 4);
char s4[100];
strcpy(s4, s3.c_str());
cout << s4 << endl;
```

Mảng: vector

- Là mảng động
- Có thể chứa dữ liệu kiểu bất kỳ (template): `vector<type>`
- `#include <vector>`
- Ví dụ khai báo:

```
/* tạo vector rỗng kiểu dữ liệu int */
vector <int> first;
//tạo vector với 4 phần tử là 100
vector <int> second (4,100);
// lấy từ đầu đến cuối vector second
vector <int> third (second.begin(),second.end())
//copy từ vector third
vector <int> four (third)
/* Tạo vector 2 chiều rỗng */
vector < vector <int> > v;
/* khai báo vector 5×10 */
vector < vector <int> > v (5, 10) ;
/* khai báo 5 vector 1 chiều rỗng  */
vector < vector <int> > v (5) ;
```

Mảng: vector

- Ví dụ sử dụng:

```
int p[] = {4, 2, 6};
vector<int> a(p, p+3);       // khởi tạo từ mảng C
a.push_back(1);              // thêm vào cuối
a.insert(a.begin() + 2, 3); // thêm ở vị trí 2
a.insert(a.end() - 1, 5);   // thêm ở vị trí 1 từ cuối
a[3] = 10;                   // phần tử thứ 4

vector<int>::iterator i; // duyệt xuôi
for (i = a.begin(); i != a.end(); i++) *i += 5;

vector<int>::reverse_iterator j; // duyệt ngược
for (j = a.rbegin(); j != a.rend(); j++)
   cout << *j << ' ';
```

Mảng: vector - Các hàm thành viên:

- Capacity:
  - size : trả về số lượng phần tử của vector.
  - empty : trả về true(1) nếu vector rỗng, ngược lại là false(0).
- Truy cập tới phần tử:
  - operator [] : trả về giá trị phần tử thứ [].
  - at : tương tự như trên.
  - front: trả về giá trị phần tử đầu tiên.
  - back: trả về giá trị phần tử cuối cùng.
- Chỉnh sửa:
  - push_back : thêm vào ở cuối vector.
  - pop_back : loại bỏ phần tử ở cuối vector.
  - insert (iterator,x): chèn “x” vào trước vị trí “iterator” (x có thể là phần tử hay iterator của 1 đoạn phần tử…).
  - erase : xóa phần tử ở vị trí iterator.
  - swap : đổi 2 vector cho nhau (ví dụ: first.swap(second);).
  - clear: xóa vector.

Deque (Hàng đợi hai đầu)

- Deque là từ viết tắt của double-ended queue (hàng đợi hai đầu).
- Deque có các ưu điểm như:
  - Các phần tử có thể truy cập thông cua chỉ số vị trí của nó.
  - Chèn hoặc xóa phần tử ở cuối hoặc đầu của dãy.
- `#include <deque>`
- Ví dụ :
  - `push_back : thêm phần tử vào ở cuối deque.`
  - `push_front : thêm phần tử vào đầu deque.`
  - `pop_back : loại bỏ phần tử ở cuối deque.`
  - `pop_front : loại bỏ phần tử ở đầu deque.`
  - …

iterator

- Các lớp chứa của STL (vector, list,…) có định nghĩa kiểu `iterator` tương ứng để duyệt các phần tử (theo thứ tự xuôi)
  - Mỗi `iterator` chứa vị trí của một phần tử
  - Các hàm `begin()` và `end()` trả về một `iterator` tương ứng với các vị trí đầu và cuối
  - Các toán tử với iterator:

    `i++` phần tử kế tiếp

    `i--` phần tử liền trước

    `*i` giá trị của phần tử
- Tương tự, có `reverse_iterator` để duyệt theo thứ tự ngược
  - Các hàm `rbegin()` và `rend()`

Danh sách liên kết: list

- Có thể chứa dữ liệu kiểu bất kỳ (template): `list<type>`
- `#include <list>`
- Duyệt danh sách dùng `iterator` tương tự như với `vector`
- Ví dụ sử dụng:

```
double p[] = {1.2, 0.7, 2.2, 3.21, 6.4};
list<double> l(p, p+5); // khởi tạo từ mảng C
l.push_back(3.4);       // thêm vào cuối
l.pop_front();          // xoá phần tử đầu

list<double>::iterator i = l.begin();  // phần tử đầu
*i = 4.122;         // gán giá trị
i++;                // phần tử kế tiếp
l.insert(i, 5.0); // chèn phần tử
l.erase(i);         // xoá phần tử
l.sort();           // sắp xếp (tăng dần)
for (i = l.begin(); i != l.end(); i++) // duyệt xuôi
   cout << *i << ' ';
```

Stack (Ngăn xếp)

- Stack là một loại container adaptor, được thiết kế để hoạt động theo kiểu LIFO
- `#include <stack>`
- Các hàm:
  - `size : trả về kích thước hiện tại của stack.`
  - `empty : true stack nếu rỗng, và ngược lại.`
  - `push : đẩy phần tử vào stack.`
  - `pop : loại bỏ phần tử ở đỉnh của stack.`
  - `top : truy cập tới phần tử ở đỉnh stack.`

Stack (Ngăn xếp)

- Ví dụ:

```
#include <iostream>
#include <stack>
using namespace std;
stack <int> s;
int i;
main() {
        for (i=1;i<=5;i++) s.push(i); // s={1,2,3,4,5}
        s.push(100);                  // s={1,2,3,4,5,100}
        cout << s.top() << endl;      // In ra 100
        s.pop();                      // s={1,2,3,4,5}
        cout << s.empty() << endl;    // In ra 0
        cout << s.size() << endl;     // In ra 5
        system("pause");
}
```

Queue (Hàng đợi)

- Queue là một loại container adaptor, được thiết kế để hoạt động theo kiểu FIFO
- Trong queue, có hai vị trí quan trọng là vị trí đầu danh sách (front), nơi phần tử được lấy ra, và vị trí cuối danh sách (back), nơi phần tử cuối cùng được thêm vào.
- `#include <queue>`
- Các hàm:
  - `size : trả về kích thước hiện tại của queue.`
  - `empty : true nếu queue rỗng, và ngược lại.`
  - `push : đẩy vào cuối queue.`
  - `pop: loại bỏ phần tử (ở đầu).`
  - `front : trả về phần tử ở đầu.`
  - `back: trả về phần tử ở cuối.`

Queue (Hàng đợi)

- Ví dụ:

```
#include <iostream>
#include <queue>
using namespace std;
queue <int> q;
int i;
main() {
        for (i=1;i<=5;i++) q.push(i); // q={1,2,3,4,5}
        q.push(100);                  // q={1,2,3,4,5,100}
        cout << q.front() << endl;    // In ra 1
        q.pop();                      // q={2,3,4,5,100}
        cout << q.back() << endl;     // In ra 100
        cout << q.empty() << endl;    // In ra 0
        cout << q.size() << endl;     // In ra 5
        system("pause");
}
```

Priority Queue (Hàng đợi ưu tiên)

- Priority queue là một loại container adaptor, được thiết kế đặc biệt để phần tử ở đầu luôn luôn lớn nhất (theo một quy ước về độ ưu tiên nào đó) so với các phần tử khác.
- Nó giống như một heap, mà ở đây là heap max, tức là phần tử có độ ưu tiên lớn nhất có thể được lấy ra và các phần tử khác được chèn vào bất kì.
- `#include <queue>`
- Các hàm:
  - `size : trả về kích thước hiện tại của priority queue.`
  - `empty : true nếu priority queue rỗng, và ngược lại.`
  - `push : đẩy vào priority queue.`
  - `pop: loại bỏ phần tử ở đỉnh priority queue.`
  - `top : trả về phần tử ở đỉnh priority queue.`

Queue (Hàng đợi)

- Ví dụ:

```
priority_queue<int> gquiz;
gquiz.push(10);
gquiz.push(30);
gquiz.push(20);
gquiz.push(5);
gquiz.push(1);
```

```
30    20    10    5    1
```

Pair

- Pair: được định nghĩa trong tệp header <utility> bao gồm hai phần tử hoặc đối tượng dữ liệu.
- Phần tử đầu tiên được tham chiếu là ‘first’ và phần tử thứ hai là ‘second’ và thứ tự được cố định (first, second).
- Cặp được sử dụng để kết hợp với nhau hai giá trị có thể khác nhau về kiểu. Cặp cung cấp một cách để lưu trữ hai đối tượng không đồng nhất như một đơn vị duy nhất.
- `#include <utility>`
- Ví dụ:

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

int main()
{
    pair<int, char> PAIR1;

    PAIR1.first = 100;
    PAIR1.second = 'G';

    cout << PAIR1.first << " ";
    cout << PAIR1.second << endl;

    return 0;
}
```

Thuật toán: tìm kiếm

- Phần tử lớn nhất, bé nhất:
  - `vector<float>::iterator p = max_element(a.begin()+2, a.end()-3);`
  - `list<string>::iterator p = min_element(l.begin(), l.end());`
  - Dựa trên các toán tử so sánh → cần định nghĩa nếu chưa có
- Tìm đúng giá trị:
  - `list<float>::iterator p = find(p1, p2, 2.5f);`
- Tìm theo tiêu chuẩn: cần định nghĩa một hàm đánh giá
  - `bool isOdd(int i) { return i%2 == 1; }`

    `list<int>::iterator p = find_if(p1, p2, isOdd);`
- Tìm kiếm và thay thế, xoá:
  - `replace_if(p1, p2, isOdd, 10);`
  - `remove_if(p1, p2, isOdd);`

Thuật toán: sắp xếp

- Sắp xếp mảng:
  - Dùng toán tử so sánh:
    - `sort(a.begin(), a.end());`
    - Phải định nghĩa toán tử “<” cho kiểu dữ liệu được chứa
  - Dùng hàm so sánh tự định nghĩa:
    - ```
      bool compare(const table& a, const table& b) {
          return a.c1 < b.c1 ||
              (a.c1 == b.c1 && a.c2 < b.c2);
      }
      sort(a.begin(), a.end(), compare);
      ```
- Sắp xếp danh sách:
  - `l.sort();`
  - `l.sort(compare);`
      ```

Câu hỏi 1

- Cho một danh sách móc nối với các phần tử trong danh sách có kiểu S1 được định nghĩa như sau:

  struct S1{int info; struct S1 *next;} *head;

- Biết con trỏ head lưu địa chỉ của phần tử đầu tiên trong danh sách. Nhóm câu lệnh nào sau đây thêm một phần tử mới p vào đầu danh sách:

- A) head->next = p; p = head;
- B) Không có phương án nào đúng.
- C) p->next = head; head = p;
- D) p->next = head; head = p-> next;

Câu hỏi 2

- Với đoạn mã sau, nếu n=13, trong các phần tử được bổ sung vào ngăn xếp S theo thứ tự nào?

```
while (n!= 0){
        R = n % 2;
        S.push(R);
        n /= 2;
}
```

- A) 1 , 1 , 0 , 1
- B) 1, 0 , 1 , 1
- C) 6 , 3 , 1
- D) 1 , 3 , 6

Câu hỏi 3

- Hỏi đoạn chương trình sau in ra kết quả gì?

```
vector<int> v(50, 1);
int s = 0;
for(int i = 0; i < 50; i += 2){
    v[i] += v[i+1];
    s += v[i] + v[i+1];
}
cout << s;
```

Câu hỏi 4

- Hỏi đoạn chương trình sau in ra kết quả gì?

```
int a[] = {3, 5, 7, 8, 7, 4};
sort(a + 1, a + 5);
for(int i = 0; i < 6; i++)
    cout << a[i] << " ";
```

Câu hỏi 5

- Hỏi đoạn chương trình sau in ra kết quả gì?

```
vector< pair<int, int> > v;
v.push_back({12,5});
v.push_back({10,17});
v.push_back({3,2});
cout << v[1].second - v[2].first;
```

Câu hỏi 6

- Hỏi đoạn chương trình sau in ra kết quả gì?

```
priority_queue< pair<int, int> > Q;
Q.push({15, 2});
Q.push({3, 15});
Q.push({10, 7});
Q.push({5, 19});
cout << Q.top().first << " ";
Q.pop();
cout << Q.top().second;
```

Bài tập 3

- Viết chương trình nhập vào một mảng các số nguyên, hãy xác định xem liệu có thể chia mảng này thành hai mảng con có tổng các phần tử của mỗi mảng là bằng nhau.
  - Ví dụ: mảng {1, 5, 11, 5} có thể chia được (hai mảng con {1, 5, 5} và {11})
  - Ví dụ: mảng {1, 5, 3} không chia được.

Bài tập 3

- Input: mảng các số nguyên
- Output: có thể / không thể chia thành 2 mảng con tổng bằng nhau
- Phân tích:
  - Kiểm tra tổng các phần tử trong mảng
  - Nếu là số lẻ: không thể chia thành 2 mảng con → return false
  - Nếu là số chẵn: mỗi mảng con sẽ có tổng (**sum**) các phần tử = ½ tổng cả mảng
  - Mảng ban đầu có thể chia được nếu tồn tại một mảng con có tổng bằng **sum**
  - Xét phần tử cuối cùng trong mảng (last):
    - Nếu last > sum → bài toán trở thành: tìm mảng con trong n-1 phần tử có tổng bằng sum (bỏ phần tử cuối ra ngoài)
    - Nếu last <= sum: bài toán trở thành:
      - tìm mảng con trong n-1 phần tử có tổng bằng sum (mảng con này không bao gồm phần tử cuối) **hoặc**
      - tìm mảng con trong n-1 phần tử có tổng bằng sum - last (mảng con này bao gồm phần tử cuối)
- Các trường hợp đều giảm được kích thước bài toán từ n → n-1 phần tử

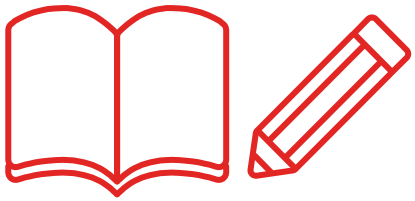
Bài tập 3

- Xây dựng bằng đệ quy:

```
bool isSubsetSum (int arr[], int n, int sum) {
    // trường hợp cơ sở
    if (sum == 0)     return true;
    if (n == 0 && sum != 0)         return false;

    // Nếu phần tử cuối > sum
    if (arr[n-1] > sum)
              return isSubsetSum (arr, n-1, sum);
    /* Ngược lại: (a) Bao gồm phần tử cuối hoặc
                  (b) Không bao gồm phần tử cuối */
    return isSubsetSum (arr, n-1, sum) ||
           isSubsetSum (arr, n-1, sum - arr[n-1]);
}
```

Bài tập 4



- Viết một chương trình:
  - *Nhập vào từ bàn phím một số nguyên dương có N chữ số.*
  - *Nhập vào một giá trị nguyên dương M.*
  - *Hãy thực hiện xoá đi M chữ số trong số N để thu được số còn lại sau khi xoá là lớn nhất có thể, xây dựng thuật toán sử dụng Stack.*

  - *Ví dụ: số 2019*
    - Xoá 1 chữ số: 219
    - Xoá 2 chữ số: 29
    - Xoá 3 chữ số: 9

Xoá để được số lớn nhất

- Input:
  - Số nguyên dương có N chữ số
  - M: số chữ số cần xoá (0<= M < N)
- Output:
  - Số lớn nhất sau khi xoá M chữ số
- Phân tích
  - Chuyển đổi số thành xâu ký tự để thuận tiện xử lý
  - Nhận thấy khi xóa đi mà số vẫn lớn nhất thì mỗi lần xóa một chữ phải tạo ra số lớn nhất
  - Các số sau khi xoá vẫn theo thứ tự ban đầu nên để thu được số lớn nhất có thể → bắt đầu xoá từ phía bên trái

Xoá để được số lớn nhất

- Phân tích: sử dụng ngăn xếp
  - Dãy đó luôn là dãy lớn nhất có thể tạo được khi xóa
  - Stack sẽ chứa các chữ số được chọn
- Ví dụ: số 3973811 gồm N=7 chữ số, cần xoá M=3 chữ số
  - Duyệt lần lượt các chữ số từ trái sang phải, stack ban đầu rỗng
  - Đọc chữ số đầu tiên: 3 và M=3 → push vào stack
    - Stack: 3 (cần xoá M=3)
  - Đọc chữ số tiếp theo: 9 và M=3 → so sánh với chữ số trong đỉnh stack: 9 > 3 → pop 3 ra khỏi stack và push 9 vào thay thế, 3 là chữ số sẽ bị xoá
    - Stack: 9 (cần xoá M=2)
  - Đọc chữ số tiếp theo: 7 và M=2 → so sánh với chữ số trong đỉnh stack: 7 < 9 → push 7 vào trong stack
    - Stack: 9 7 (cần xoá M=2)

Xoá để được số lớn nhất

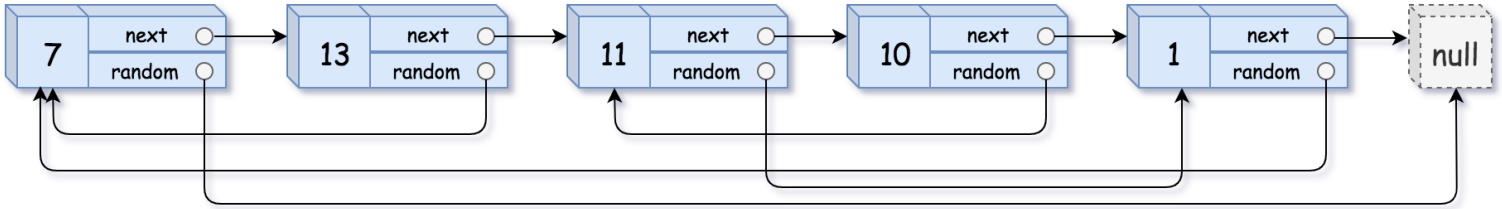
- Phân tích: sử dụng ngăn xếp
- Ví dụ: số 3973811 gồm N=7 chữ số, cần xoá M=3 chữ số
  - Đọc chữ số tiếp theo: 3 và M=2 → so sánh với chữ số trong đỉnh stack: 3 < 7 → push 3 vào trong stack
    - Stack: 9 7 3 (cần xoá M=2)
  - Đọc chữ số tiếp theo: 8 và M=2 → so sánh với chữ số trong đỉnh stack: 8 > 3 → pop 3 ra khỏi stack (M=1), 8 > 7 → pop 7 ra khỏi stack (M=0) và push 8 vào thay thế, 3 và 7 là chữ số sẽ bị xoá
    - Stack: 9 8 (cần xoá M=0)
  - Đọc chữ số tiếp theo: 1 và M=0 → push 1 vào stack
    - Stack: 9 8 1 (cần xoá M=0)
  - Đọc chữ số tiếp theo: 1 và M=0 → push 1 vào stack
    - Stack: 9 8 1 1 (cần xoá M=0)
  - Kết thúc → số thu được 9811

Xoá để được số lớn nhất

```
void Delete(char s[], int n, int m) {
    char stack[100];
    top = -1;
    Push(stack, s[0]);
    for (int i = 1; i < n; i++) {
        if ((s[i] > Top(stack)) && (m != 0)) {
            Pop(stack);
            m--;
        }
        Push(stack, s[i]);
    }
    char x[100];
    for (int i = 0; i < n - m; i++) {
        x[i] = Pop(stack);
    }
    for (int i = n - m - 1; i >= 0; i--)
        printf("%d", x[i]);
}
```

Bài tập 5

- Xây dựng một danh sách liên kết, mỗi nút trong danh sách có 2 con trỏ.
  - Một con trỏ đến nút đứng kế tiếp trong danh sách
  - Một con trỏ đến một nút ngẫu nhiên trong danh sách hoặc giá trị NULL
  - Ví dụ:



  - a. Hãy viết hàm tạo bản sao của một danh sách như trên
  - b. Hàm cập nhật con trỏ ngẫu nhiên của từng nút danh sách liên kết để trỏ đến nút có giá trị lớn nhất bên phải của chúng

Bài tập 5

- Định nghĩa cấu trúc danh sách trên:

```
struct Node {
    int data;
    Node* next, *random;
    // Constructor
    Node(int data) {
        this->data = data;
        this->next = this->random = NULL;
    }
};
```

Bài tập 5

- Hàm thêm một nút mới vào đầu danh sách:

```
void push(Node* &head, int data) {
    Node* newNode = new Node(data);
    newNode->next = head;
    head = newNode;
}
```

Thanks!

Any questions?