(Standard Template Library)

(S T L)

هي مكتبة قياسية في لغة C++ تتوفر محتوية جاهزة من Data structure, Algorithms على شكل تساعدك في كتابة كود أسرع وأسهل.

ليه استخدم STL ؟؟؟

① توفر الوقت والمجهود

لبدل ما اعمل كود ~~كبير~~ من الصفر فالمكتبة دي بتوفر كود جاهز بستخدمه كالطول

② أداء عالي

③ مرونة ⇐ تدعم محتوية من أنواع البيانات المختلفة

④ إعادة استخدام الكود مرة أخرى

لبدلاً من اعادة اختراع العجلة من الصفر

مكونات STL

١ حاويات (Containers) هي كلاسات جاهزة

- Sequential containers حاويات متسلسلة — مثل: vector, List, deque
- Container Adaptors حاويات متسلسلة خاصة — مثل: stack, queue, priority queue
- Associative containers حاويات ترابطية — مثل: set, map, Multiset, MultiMap

---

٢ خوارزميات جاهزة ← مثل

الفرز Sorting ← مثل sort

البحث Searching ← مثل Find, binary_search

التعديل Modifing ← مثل unique, reverse

---

٣ المكررات ← iterators

مثل Pointers لكنها أكثر تطوراً

وتستخدم للتنقل بين عناصر الـ Containers

المصفوفة (Vector)

هو حاوية ديناميكية [Dynamic Container]
له يمكن تكبير حجمه أثناء التشغيل
عكس Array

العمليات على Vector

1. Push_back(Value) ⟹ إضافة عنصر في نهاية vector
2. PoP_back() ⟹ حذف آخر عنصر من vector
3. size() ⟹ عدد العناصر داخل vector
4. insert(Position, Value) ⟹ إدراج عنصر في موقع معين من vector
   (الموقع index ، القيمة)
5. erase(Position) ⟹ حذف عنصر من موقع معين في vector
6. clear() ⟹ حذف كل العناصر من vector
7. Front() ⟹ إرجاع أول عنصر في vector
8. back() ⟹ إرجاع آخر عنصر في vector

مثال على vector

```
#include <vector>
#include <iostream>
using namespace std;
int main()
{
    vector<int> nums = {10, 20, 30};
    nums.push_back(40); // يضيف 40 في الآخر
    nums.push_back(50); // يضيف 50 في الآخر

    for(int num : nums)
    {
      cout<<num<<" "; // 10 20 30 40 50
    }
      cout<<endl; // سطر جديد
    nums.pop_back() // 50 يحذف آخر عنصر

    for(int num: nums)
    {
    cout<<num<<" "; // 10 20 30 40
    }
  return 0;
}
```

Array

هي حاوية ثابتة الحجم [Static container] حيث لا يمكن تغيير حجمها

Array بعد التعريف

ولكنها تتمتع بالأداء من Vector + Array

العمليات على Array

1. size() ⟹ Array عدد عناصر
2. front() ⟹ Array ارجاع أول عنصر من
3. back() ⟹ Array ارجاع آخر عنصر من
4. at index() ⟹ ارجاع عنصر من الموقع المحدد

```
#include <iostream>
#include <array>
using namespace std;
int main()
{
array<int, 5> nums = {1,2,3,4,5};
cout << nums.front() << endl;   // 1   العنصر الأول
cout << nums.back() << endl;    // 5   العنصر الأخير
cout << nums.at(2) << endl;     // 3   العنصر في الموقع
cout << nums.size() << endl;    // 5   الحجم array
return 0;
}
```

De que

Double-ended Queue

شبيه vector ولكنه يسمح بالإضافة والحذف من البداية والنهاية

أهم العمليات على Deque

1. Push_back(value) ⟹ اضافة عنصر في النهاية
2. Push_front(value) ⟹ اضافة عنصر في البداية
3. Pop_back() ⟹ حذف آخر عنصر
4. Pop_front() ⟹ حذف أول عنصر
5. size() ⟹ عدد العناصر لل deque
6. front() ⟹ ارجاع اول عنصر
7. back() ⟹ ارجاع آخر عنصر

```
#include<iostream>
#include<deque>
using namespace std;
int main(){
    deque<int> dq;
    dq.push_back(10);
    dq.push_front(5);
    dq.push_back(15);
    dp.pop_front(); // (5) حذف العنصر الأول
    for(int num:dq)
    {cout<<num<<" ";
    }return 0;
```

Container Adaptor

stack queue priority_queue

تكون كلها من الحاويات الاساسية مثل vector, deque وتلغي
توفر طريقة وصول محددة لكل عنصر حيث

[last in first out] LIFO ⟸ stack

[first in first out] FIFO ⟸ queue

الأولوية الأكبر أو الأصغر ⟸ priority queue

stack

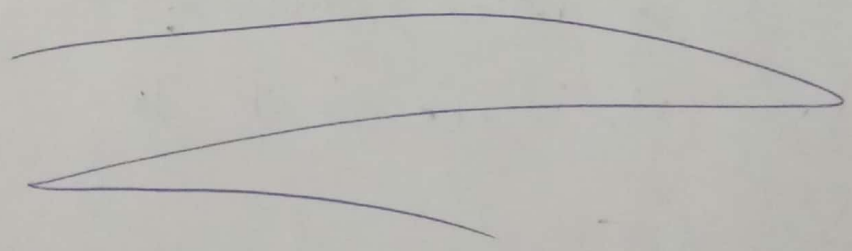
آخر واحد داخل ← أول واحد خارج
LIFO

العمليات على stack

① Push(value) ⟹ stack إضافة عنصر
② PoP() ⟹ stack إزالة العنصر العلوي من
③ toP() ⟹ stack أول عنصر في
④ size() ⟹ stack حجم (عدد العناصر الموجودة)
⑤ empty ⟹ هل stack فارغ أم لا

stack کی کلاس

```
#include <iostream>
#include <stack>
using namespace std;
int main(){
    stack <int> s;
    s.push(10);
    s.push(20);
    s.push(30);
    s.push(40);
    cout << s.top() << endl;   // 40
    s.pop();                   //      40 ہٹا دیا
    cout << s.top() << endl;   // 30
    s.pop();                   //      30 ہٹا دیا
    cout << s.top() << endl;   // 20
    cout << s.size() << endl;  // 2
    return 0;
}
```



Queue

طابور

يعمل بمبدأ [أول واحد يدخل أول واحد يخرج]

[FIFO]

طابونة العيش

العمليات على Queue

1. Push(value) ⟹ اضافة عنصر في Queue
2. POP() ⟹ حذف عنصر من Queue
3. front() ⟹ ارجاع أول عنصر في Queue
4. back() ⟹ ارجاع آخر عنصر في Queue
5. size() ⟹ ارجاع عدد عناصر الـ Queue
6. empty() ⟹ التحقق اذا كان الـ Queue فارغ أم لا

مثال على Queue

```
#include <iostream>
#include <queue>
using namespace std;
int main(){
  queue <int> q;
  q.push(5);
  q.push(20);
cout<<q.front()<<endl;   // 5
cout<<q.back()<<endl;    // 20
  q.push(30);
  q.push(40);
  cout<<q.size()<<endl;  // 4
  q.POP()
  cout<<q.size()<<endl;  // 3
```

Priority_queue

Priority ← مثل queue ولكنه يعتمد على الأولوية افتراضياً [يخرج العنصر ذو الأولوية الأكبر]

ولكن يمكن تغيير ذلك

أهم العمليات على Priority_queue

1. Push(value) ← اضافة عنصر مع الحفاظ على الترتيب
2. Pop() ← ازالة العنصر ذو الأولوية الأعلى
3. top() ← ارجاع العنصر ذو الأولوية الأعلى
4. size() ← ارجاع عدد العناصر
5. empty() ← التحقق اذا كانت فارغة أم لا

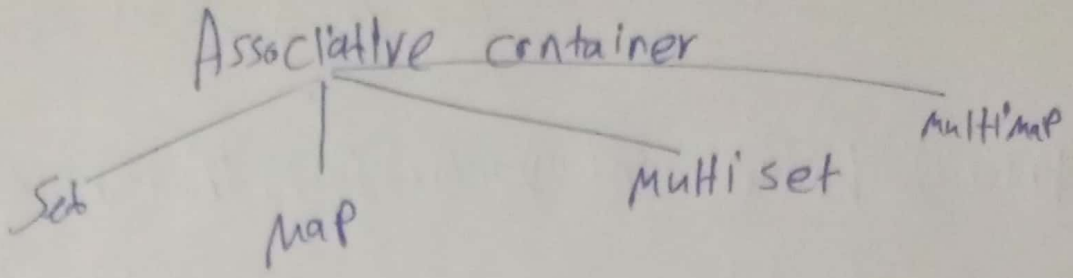
مثال [الأولوية الأكبر]

```
#include <iostream>
#include<queue>
using namespace std;
int main(){
    priority_queue<int> pq;
    pq.push(20);
    pq.push(30);
    pq.push(50);
    cout<<pq.top()<<endl;      // 50
    pq.pop();
    cout<<pq.top()<<endl;      // 30
    return 0;
}
```

تغير الأولوية من الأكبر إلى الأصغر

```
priority_queue <int, vector<int>, greater<int>> pq;
```

```
#include <iostream>
# include <queue>
using namespace std;
int main()
{
priority_queue <int, vector<int>, greater<int>> pq;
  pq.push(50);
  pq.push(40);
  pq.push(30);
cout<<pq.top()<<endl;      //30
  pq.pop();
cout<<pq.top<<endl;        //40
  return 0;
}
```



[1] set ⟸ هى عبارة عن مجموعة من القيم الفريدة المرتبة لا تكرر

* لا تسمح بالتكرار
* مرتبة [تصاعدياً]
* تسمح بعمليات الكشف والاسترجاع كبيرة O(log N)

العمليات على set

① Insert (value) ⟹ اضافة عنصر جديد
② erase (value) ⟹ حذف عنصر معين
③ Find (value) ⟹ البحث عن عنصر معين
④ Count (value) ⟹ التحقق اذا كان العنصر موجود أم لا
- لو العنصر موجود ⟸ يرجع 1
- لو العنصر غير موجود ⟸ يرجع 0

⑤ size() ⟹ عدد العناصر
⑥ empty() ⟹ التحقق اذا كانت set فارغة أم لا
⑦ clear() ⟹ حذف كل عناصر set

مثال على set

```
#include <iostream>
#include <set>
using namespace std;
int main(){

    set<int>s;
    s.insert(20);
    s.insert(30);
    s.insert(40);
    s.insert(20);   // مش هيكون جوا set لانه مكرر

    for(int num:s)
    {
     cout<<num<<" ";
    }
     cout<<endl;

    s.erase(s);
    if(s.count(10))
    {cout<<"yes";
    }else{cout<<"No";}

     for(int num:s){
     cout<<num<<" ";
    }return 0;
}
```

Map

نوع من انواع Datastructure يعتمد على Key, Value مفتاح والقيمة

يخزن الأزواج (Key, value) حيث كل مفتاح يرتبط قريناً لا يتكرر

العمليات على Map

① Insert({key, value}) ⟹ ادراج قيمة والمفتاح

② erase(key) ⟹ حذف مفتاح معين

③ find(key) ⟹ البحث عن مفتاح

④ count(key) ⟹ التحقق من وجود المفتاح

⑤ size() ⟹ ايجاد عدد العناصر

⑥ empty() ⟹ التحقق من Map اذا كان فارغاً أم لا

⑦ clear() ⟹ حذف كل عناصر Map

MAP لغة

```
#include <iostream>
#include <map>
using namespace std;
int main(){
map <string, int> students;
students["salah"]= 50;
students["sara"]= 90;
students['omar"] =80;
students['yara"] = 70;

for(auto student : students)
{ cout<< student.first << ":" << student.second<<endl;
}
student. erase('sara');    // سارة حذف

cout << endl;

for(auto student : students)
{
cout<< student.first<<":" <<student. second<<endl;
}
return 0;
}
```

Multiset

مجموعة تسمح بالتكرار

لها نفس وظائف set وطريقة تعريف البيانات

مثال

```
#include <iostream>
#include <set>
using namespace std;
int main(){
   Multiset <int> ms;

   ms.insert(10);
   ms.insert(20);
   ms.insert(30);
   ms.insert(10);          // تسمح بالتكرار

  for(int num: ms)
   {
     cout<< num<<" ";
   }
   cout<<"*******"<<endl;
   return 0;
}
```

Multi Map

شبه Map ولكنه يسمح بتكرار المفاتيح
لا يعني
key مشترك ومكرر عادي

مثال

```
#include <iostream>
#include <map>
using namespace std;
int main() {
    multimap <string, int> MM;
    MM.insert({"salah", 90});
    MM.insert({"salah", 80});
    MM.insert({"yara", 70});
    for (auto student : MM)
    {
    cout << student.first << " : " << student.second << endl;
    }
    return 0;
}
```

Iterators

هي طريقة للتنقل داخل الحاويات مثل vector, map, set, ...
وكأنها مؤشرات Pointers

نستخدم begin() و end() للوصول (لأول) آخر قيمة
نستخدم rbegin() و rend() لعكس القيم

مثال مع vector

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
 vector<int> nums = {1, 2, 3, 4, 5};
 vector<int>:: iterator it;   // تعريف iterator
For(it = nums.begin(); it != nums.end(); it++)
 { cout << *it << " ";
 }
 return 0;
}
```

ممكن استخدام كلمة (auto) بدل ما تكتب

```
vector<int>::iterator it;
```

مكان هذا الكلام

```
For(auto it = nums.begin(); it != nums.end(); it++)
 { cout << *it << " ";
 }
```

مثال على set

```
#include <iostream>
#include <set>
using namespace std;
int main()
{
  set<int> s{ 2, 4, 6, 2, 8};
  for(auto it=s.begin(); it!=s.end(); it++)
  {
    cout<<*it<<" ";
  }
  return 0;
}
```

مثال على map

```
#include <iostream>
#include <map>
using namespace std;
int main()
{
  map<string,int> M= {{"salah",90},{"sara",20},{"nour",70}};
  for(auto it=M.begin(); it!=M.end(); it++){
  cout<<it->first<<" : "<<it->second<<endl;
  }
  return 0;
}
```

map تستخدم مع key ← first
value ← second

لعكس الترتيب نستخدم

rbegin() ← يبدأ من آخر عنصر

rend() ← ينتهي قبل أول عنصر

begin() ← يبدأ من أول عنصر

end() ← يبدأ بعد آخر عنصر مباشرة

STL في Algorithms

#include <Algorithm> ← لازم يتعمل كتابة

① sort الترتيب

sort(begin(), end());

② reverse عكس الترتيب

reverse(begin(), end());

③ unique ← ازالة القيم المكررة

unique(begin, end)

لو لكن قبل unique سيتم sort

ثم حذف القيم مقابلياً باستخدام erase

④ find ← البحث بالعنصر

find(begin, end, value)

⑤ binary_search ← البحث بالعنصر

O(Log n) التعقيد ← find أسرع من

لابد من ترتيب البيانات لكي تعمل binary_search