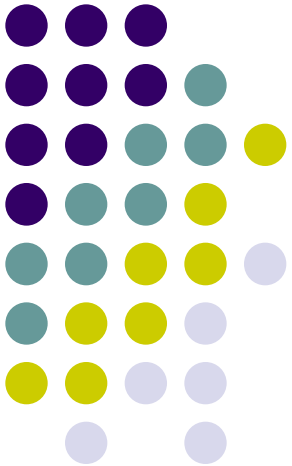
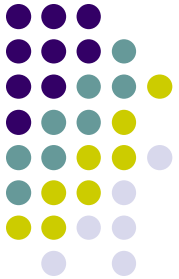
Chương 7

Khuôn mẫu(Template)

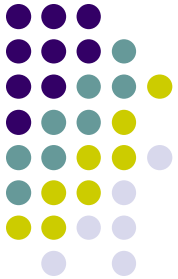


Nội dung



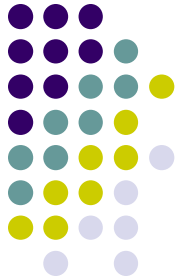
- Giới thiệu
- Khuôn mẫu hàm
- Khuôn mẫu lớp
- Friend của khuôn mẫu
- Standard Template Library - STL

Giới thiệu



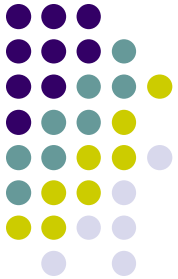
- Khuôn mẫu là một trong những tính năng phức tạp nhất và mạnh nhất của ngôn ngữ C++
- Là công cụ hỗ trợ tái sử dụng code
- Là cơ chế cho phép lập trình tổng quát
- C++ hỗ trợ 2 loại khuôn mẫu: khuôn mẫu hàm và khuôn mẫu lớp
- Khuôn mẫu hàm/lớp là hàm/lớp tổng quát, không phụ thuộc vào kiểu dữ liệu

Khuôn mẫu hàm(Function Template)



- Bài toán:
  - Viết hàm hoán vị 2 số nguyên
  - Viết hàm hoán vị 2 số thực
  - Viết hàm hoán vị 2 kí tự
- Cách giải quyết: Sử dụng chồng hàm swap

Khuôn mẫu hàm(Function Template)

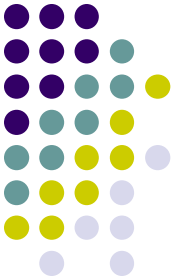


```
void hoanvi(int &a, int &b)
{
    int temp = a;
    a = b;
    b = temp;
}
```

```
void hoanvi(float &a, float &b)
{
    float temp = a;
    a = b;
    b = temp;
}
```

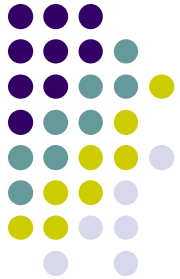
```
void hoanvi(char &a, char &b)
{
    char temp = a;
    a = b;
    b = temp;
}
```

Khuôn mẫu hàm(Function Template)



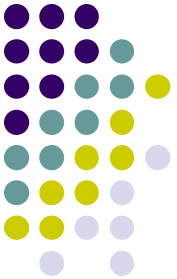
```
int main()
{
    int ia = 5, ib = 4;
    float fa = 1.5, fb = 5.5;
    char ca = 'A', cb = 'B';
    hoanvi(ia,ib);
    hoanvi(fa,fb);
    hoanvi(ca,cb);
    cout<<"ia = "<<ia<<",ib = "<<ib<<endl;
    cout<<"fa = "<<fa<<",fb = "<<fb<<endl;
    cout<<"ca = "<<ca<<",cb = "<<cb<<endl;
    return 0;
}
```

Khuôn mẫu hàm(Function Template)



- Chồng hàm gây ra tình trạng “lặp lại mã”
  → Sử dụng khuôn mẫu hàm
- Khuôn mẫu hàm cho phép viết 1 lần và sử dụng cho nhiều kiểu dữ liệu khác nhau
- Bộ dịch sẽ sinh ra nhiều phiên bản khác nhau của 1 khuôn mẫu hàm khi hàm được gọi(hàm thể hiện)

Khuôn mẫu hàm(Function Template)



- Khai báo

```
//Khuôn mẫu hàm với 1 tham số kiểu dữ liệu T
template <class T>
return-type funcname (arguments of type T)
{
        //thân khuôn mẫu hàm
}
```

```
//Khuôn mẫu hàm với nhiều tham số kiểu dữ liệu
T1, T2,…
template <class T1, class T2,…>
return-type funcname (arguments of type T1,T2)
{
        //thân khuôn mẫu hàm
}
```

Ví dụ 1

```
template <class T>
void hoanvi(T &a, T &b)
{
    T temp = a;
    a = b;
    b = temp;
}
```

Tự động phát sinh

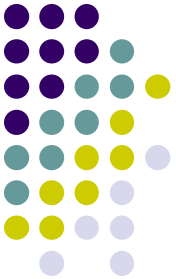
```
void hoanvi(int &a, int &b)
{
    int temp = a;
    a = b;
    b = temp;
}
```

```
void hoanvi(float &a, float &b)
{
    float temp = a;
    a = b;
    b = temp;
}
```

```
void hoanvi(char &a, char &b)
{
    char temp = a;
    a = b;
    b = temp;
}
```

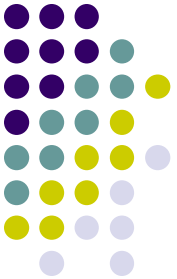
```
int main()
{
    int ia = 5, ib = 4;
    float fa = 1.5, fb = 5.5;
    char ca = 'A', cb = 'B';
    hoanvi(ia,ib);
    hoanvi(fa,fb);
    hoanvi(ca,cb);
    cout<<"ia = "<<ia<<",ib = "<<ib<<end
    cout<<"fa = "<<fa<<",fb = "<<fb<<end
    cout<<"ca = "<<ca<<",cb = "<<cb<<end
    return 0;
}
```

Ví dụ 2: Sử dụng khuôn mẫu hàm với kiểu dữ liệu lớp



```
template <class T>
void hoanvi(T &a, T &b)
{
    T temp = a;
    a = b;
    b = temp;
}
int main()
{
    Point p1(2,4), p2(1,5);
    hoanvi(p1,p2);
    cout<<"p1:"<<p1<<", p2:"<<p2<<endl;
    return 0;
}
```

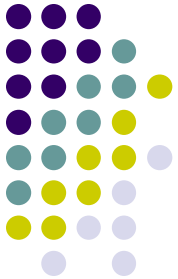
Ví dụ 3: Khuôn mẫu hàm với nhiều tham số kiểu dữ liệu



```
template<class T1, class T2>
T1 sum(T1 x, T2 y, T1 z)
{
    return x + y + z;
}
int main()
{
    int ia = 5, ib = 4, ic = 3;
    float fa = 1.5, fb = 5.0, fc = 4.0;
    cout<<sum(ia,fa,ib)<<endl;//(int,float,int)
    cout<<sum(fa,ia,fb)<<endl;//(float,int,float)
    cout<<sum(ia,ib,ic)<<endl;//(int, int, int)
    cout<<sum(fa,fb,fc)<<endl;//(float,float,float)
    //cout<<sum(ia,fa,fb)<<endl;//(int,float,float)
    return 0;
}
```

error C2782: 'T1 sum(T1,T2,T1)'
: template parameter 'T1' is ambiguous

Hạn chế của khuôn mẫu hàm



- Về nguyên tắc, một tham số kiểu có thể tương ứng với kiểu dữ liệu bất kì
- Tuy nhiên, trong một số trường hợp hàm thể hiện được sinh ra không thực hiện đúng hoặc báo lỗi dịch

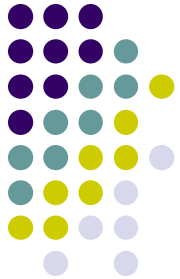
```
template<class T>
T find_min(T a, T b)
{
    return (a < b ? a : b);
}
int main()
{
    cout<<find_min(4,5)<<endl; //return 4
    cout<<find_min(1.5,2.0)<<endl;//return 1.5
    cout<<find_min("HELLO","ANNA")<<endl; // return HELLO. Incorrect Result
    return 0;
}
```

Hạn chế của khuôn mẫu hàm

```
template<class T>
T find_min(T a, T b)
{
    return (a < b ? a : b);
}
int main()
{
    Circle c1(2,3,5), c2(2,3,1);
    cout<<find_min(c1,c2)<<endl;

    return 0;
}
```

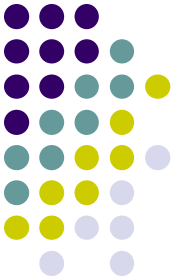
error C2676: binary '<' :
'Circle' does not define this operator

Khuôn mẫu lớp(Class Template)



- Khuôn mẫu lớp cho phép tạo ra lớp tổng quát
- Những lớp chỉ làm việc trên một kiểu dữ liệu thì không nên tổng quát hóa
- Những lớp chứa (*container class*) như **Stack**, **List**,… nên được tổng quát hóa.
- Bộ dịch sẽ sinh ra nhiều phiên bản khác nhau của lớp khi đối tượng được tạo

Khuôn mẫu lớp(Class Template)

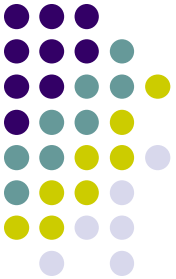


- Khai báo

```
//Khuôn mẫu lớp với 1 tham số kiểu dữ liệu T
template <class T>
class  class_name
{
        //thanh viên của lớp với kiểu dữ liệu T
}
```

```
//Khuôn mẫu lớp với nhiều tham số kiểu dữ liệu
T1, T2,…
template <class T1, class T2,…>
class  class_name
{
    //thành viên của lớp với kiểu dữ liệu T1, T2
}
```

Khuôn mẫu lớp (Class Template)



- Tạo đối tượng của lớp mẫu

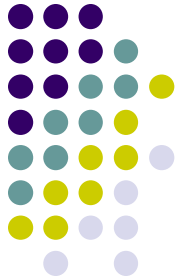
  class_name<T> obj_name;

  class_name<T1,T2> obj_name;

- Định nghĩa hàm thành viên của lớp mẫu

```
template <class T>
return-type class-name <T>:: function-name (argument list)
{
    //thân hàm thành viên của lớp mẫu với kiểu dữ liệu T
}
```

Ví dụ: My_List.h

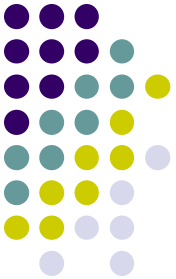


```
#pragma once
template<class T>
class My_List
{
private:
//so luong phan tu lon nhat danh sach co the chua
    int max_size;
//count:so luong phan tu hien co trong danh sach
    int count;
    T *data;
public:
     My_List(int size);
    ~My_List(void);
    bool IsEmpty() const;
    bool IsFull() const;
    int  getCount() const;
    void Insert(T newElem);
    void Delete(T elem);
    bool IsContain(T elem) const;
    void Print() const;

};
```

Ví dụ: My_List.cpp

```
#include "My_List.h"
#include <iostream>
using namespace std;

template<class T>
My_List<T>::My_List(int size)
{
     count = 0;
     max_size = size;
     data = new T[max_size];
}
template<class T>
void My_List<T>::Print() const
{
     for(int i = 0; i < count;i++)
          cout<<data[i]<<"\t";
     cout<<endl;
}
template<class T>
void My_List<T>::Insert(T newElem)
{
     data[count++] = newElem;
}
```

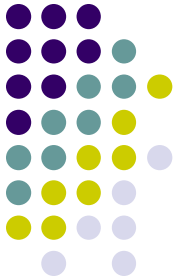
Ví dụ: main.cpp



```
#include <iostream>
#include "My_List.h"

using namespace std;
int main()
{
    My_List<int> ds1(5);
    My_List<float> ds2(5);
    My_List<char> ds3(5);
    cout<<"Danh sach 1 co: "<<ds1.getCount()<<" phan tu"<<endl;
    ds1.Insert(4);
    ds1.Insert(1);
    cout<<"Danh sach 1 co: "<<ds1.getCount()<<" phan tu"<<endl;
    ds2.Insert(4.5);
    ds2.Insert(1.5);
    ds3.Insert('A');
    ds1.Print();
    ds2.Print();
    ds3.Print();
    return 0;
}
```

error LNK2019: unresolved external symbol "public: __thiscall My_List<int>::~My_List<int>(void) ……

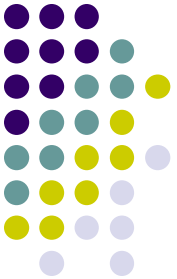
Ví dụ: main.cpp



```
#include "My_List.h"
#include "My_List.cpp"
using namespace std;
int main()
{
    My_List<int> ds1(5);
    My_List<float> ds2(5);
    My_List<char> ds3(5);
    cout<<"Danh sach 1 co: "<<ds1.getCount()<<" phan tu"<<endl;
    ds1.Insert(4);
    ds1.Insert(1);
    cout<<"Danh sach 1 co: "<<ds1.getCount()<<" phan tu"<<endl;
    ds2.Insert(4.5);
    ds2.Insert(1.5);
    ds3.Insert('A');
    ds1.Print();
    ds2.Print();
    ds3.Print();
    return 0;
}
```

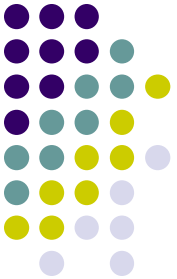
Thêm dòng #include “My_List.cpp”
để giải quyết vấn đề lỗi liên kết file .h và .cpp

Khai báo bạn bè của khuôn mẫu lớp



```
#include <iostream>
using namespace std;
template<class T>
class My_List
{
private:
//so luong phan tu lon nhat danh sach co the chua
    int max_size;
//count:so luong phan tu hien co trong danh sach
    int count;
    T *data;
public:
     My_List(int size);
    ~My_List(void);
    bool IsEmpty() const;
    bool IsFull() const;
    int  getCount() const;
    void Insert(T newElem);
    void Delete(T elem);
    bool IsContain(T elem) const;
   // void Print() const;
    template<class T>
    friend ostream& operator<<(ostream& out, const My_List<T> &ds);
};
```

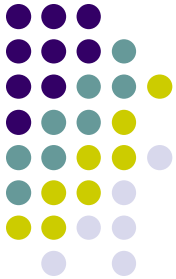
Khai báo bạn bè của khuôn mẫu lớp



```
#include "My_List.cpp"
#include <iostream>
#include "My_List.h"
using namespace std;
int main()
{
    My_List<int> ds1(5);
    My_List<float> ds2(5);
    My_List<char> ds3(5);
    cout<<"Danh sach 1 co: "<<ds1.getCount()<<" phan tu"<<endl;
    ds1.Insert(4);
    ds1.Insert(1);
    cout<<"Danh sach 1 co: "<<ds1.getCount()<<" phan tu"<<endl;
    ds2.Insert(4.5);
    ds2.Insert(1.5);
    ds3.Insert('A');
    cout<<ds1<<endl;
    cout<<ds2<<endl;
    cout<<ds3<<endl;
    return 0;
}
```

Sử dụng toán tử << để in danh sách

Tham số biểu thức trong khuôn mẫu lớp

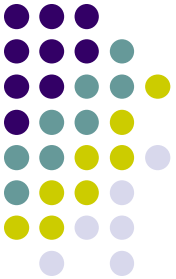


```
template <class T, int  size = 100>
class Stack
{
   private:
      int    stackptr;
      T      stackmem[size];
   public:
      Stack( ) : stackptr(0) { }
      void Push(T  data);
      T      Pop( );
};
```

```
template  <class T, int size>
void Stack<T, size>::Push(T data)
{
   if (stackptr < size)
      stack[stackptr++] = data;
   // assumes operator= is defined for T
}


template <class T, int  size>
T Stack<T, size>::Pop( )
{
   return  stack[--stackptr];
}
```

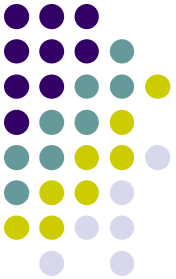
Tham số biểu thức kiểu int và chúng ta cần chỉ rõ giá trị của nó khi sử dụng khuôn mẫu lớp

Tham số biểu thức trong khuôn mẫu lớp



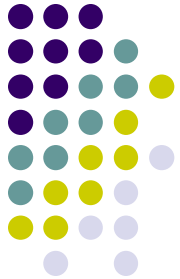
```
void main( )
{
    Stack<int, 10>  S;
    Stack<double> D;

    S.Push(10);
    S.Push(20);

    D.Push(2.7);
    D.Push(3.141495);

    cout << S.Pop( ) << endl;
    cout << D.Pop( ) << endl;
}
```

Phép gán giữa các đối tượng của các lớp thể hiện



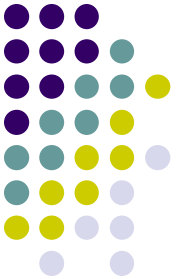
```
Stack<int, 10> S;
Stack<double> D;

D = S; // error

Stack<int, 10> S;
Stack<int, 20> D;

S = D;  //error
```

Standard Template Library (STL)



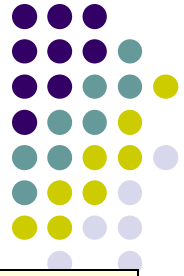
- STL là thư viện của C++ cung cấp nhiều cấu trúc dữ liệu, và thuật toán để hỗ trợ cho lập trình máy tính
- Mỗi lớp trong STL đều là khuôn mẫu
- STL gồm nhiều lớp chứa như: vector, list, set, map, stack, queue,…
- Mỗi lớp chứa trong STL có thể sử dụng với kiểu dữ liệu, đối tượng bất kì

Ví dụ: Lớp Vector



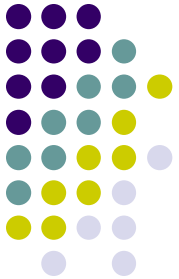
```
// Instantiate a vector
vector<int> V;

// Insert elements
V.push_back(2);           // v[0] == 2
V.insert(V.begin(), 3); // V[0] == 3, V[1] == 2

// Random access
V[0] = 5;                 // V[0] == 5

// Test the size
int size = V.size();      // size == 2
```

Ví dụ: Lớp map



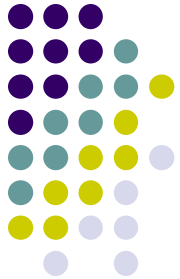
```
#include <iostream>
#include <map>
#include <string>
using namespace std;

int main() {
    map<string, int> freq; // map of words and their frequencies
    string word;           // input buffer for words.

    //--- Read words/tokens from input stream
    while (cin >> word) {
        freq[word]++;
    }

    //--- Write the count and the word.
    map<string, int>::const_iterator iter;
    for (iter=freq.begin(); iter != freq.end(); ++iter) {
        cout << iter->second << " " << iter->first << endl;
    }
    return 0;
}//end main
```

Bài tập



- Bài 1: Viết hàm template trả về giá trị trung bình của một mảng, các tham số hình thức của hàm này là tên mảng và kích thước mảng.
- Bài 2: Cài đặt hàng đợi template.
- Bài 3: Cài đặt lớp template dùng cho cây nhị phân tìm kiếm (BST).
- Bài 4: Cài đặt lớp template cho vector để quản lý vector các thành phần có kiểu bất kỳ.

Bài tập



- Bài 5: Viết hàm template để sắp xếp kiểu dữ liệu bất kỳ.
- Bài 6: Trong C++, phép toán new được dùng để cấp phát bộ nhớ, khi không cấp phát được con trỏ có giá trị NULL. Hãy cài đặt lại các lớp Matrix và Vector trong đó có bổ sung thêm thành viên là lớp exception với tên gọi là Memory để kiểm tra việc cấp phát này.