Copyright Warning

COMMONWEALTH OF AUSTRALIA

Copyright Regulations 1969

WARNING

This material has been reproduced and communicated to you by or on behalf of the University of Sydney pursuant to Part VB of the Copyright Act 1968 (**the Act**).

The material in this communication may be subject to copyright under the Act. Any further copying or communication of this material by you may be the subject of copyright protection under the Act.

Do not remove this notice.

Copyright © ViSLAB, The University of Sydney

Object Oriented Design

Week 9-2

Copyright © ViSLAB, The University of Sydney

Template and STL

- In Week 4 tutorial
- AutoFreePtr class:

```
class AutoFreePtr {
    char *ptr;   // a pointer to an allocated char array
public:
    // you need implement here
}
```

Copyright © ViSLAB, The University of Sydney

Template and STL

```
    class AutoFreePtr {
    char *ptr;
public:
    AutoFreePtr():ptr(0) {}
    ~AutoFreePtr() {
        delete[] ptr;
    }

    AutoFreePtr(char *ptr):ptr(0){
        operator=(ptr);
    }

    char *operator=(char *ptr) {
        if(this->ptr) delete[] this->ptr;
        this->ptr = ptr;
        return this->ptr;
    }

    operator char* () {
        return ptr;
    }

    char &operator[](int index) {
        return ptr[index];
    }
};
```

```
void removeSpace(char *str) {
    int n = strlen(str);
    AutoFreePtr tmp = new char[n+1];
    strcpy(tmp, str);
    int i, j;
    for (i = j = 0; i < n; i++)
        if (str[i] != ' ')
            tmp[j++] = str[i];
    tmp[j] = '\0';
    strcpy(str, tmp);
}
```

Copyright © ViSLAB, The University of Sydney

Template and STL

- In C++'s standard library (std::)
- auto_ptr : provides similar auto release mech.
- Template?

Invitation

Dear

Date

Place

auto_ptr<   >

Copyright © ViSLAB, The University of Sydney

auto_ptr

- include <memory>
- auto_ptr : provides similar auto release mech.
- Template?

Invitation

Dear

Date

Place

auto_ptr< >

Copyright © ViSLAB, The University of Sydney

```
#include <memory>
using namespace std;

void foo() {
   auto_ptr<int> p(new int);
   *p = 10;
}

int main(void) {
   foo();
   return 0;
}
```

- <int>
- auto_ptr has int*
- p : variable
- call constructor
- needs int *

- auto_ptr p;
  - compiler error...not arg for the template

Copyright © ViSLAB, The University of Sydney

- declare with a template
  - a customised class is instantly created.

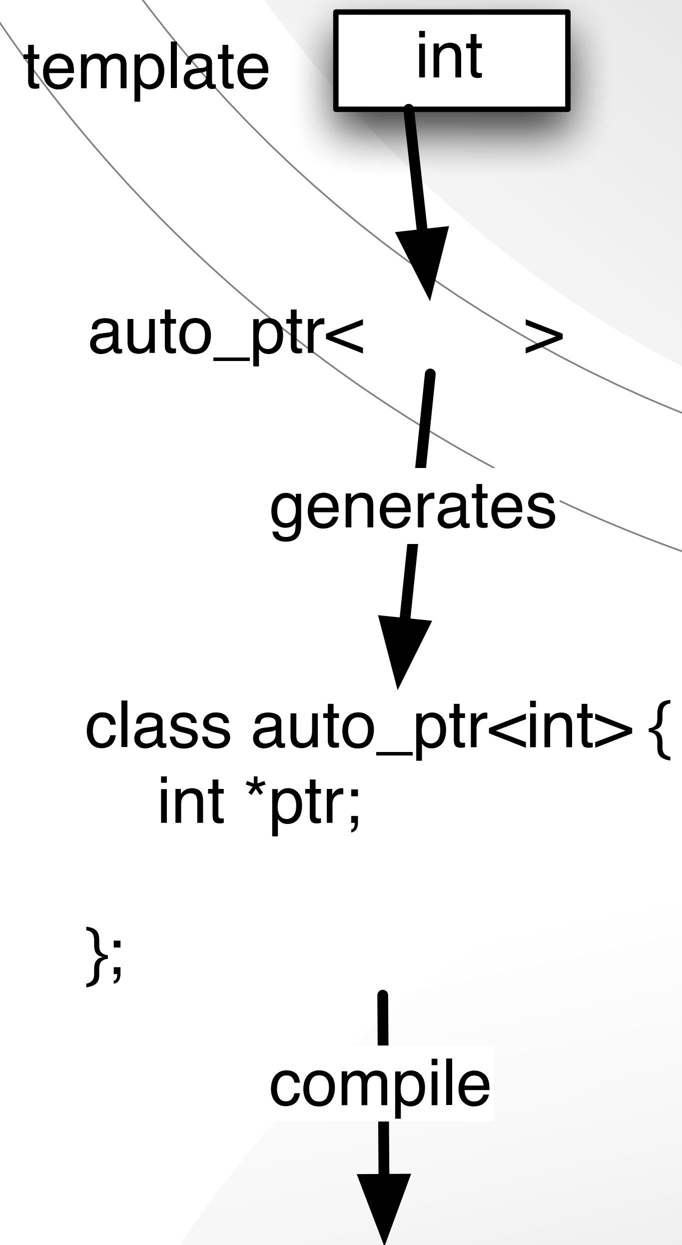
- operator*, operator-> ..predefined

Copyright © ViSLAB, The University of Sydney

```
   #include <memory>
using namespace std;

void foo() {
   auto_ptr<int> p(new int);
   *p = 10;
}

int main(void) {
   foo();
   return 0;
}
```



Copyright © ViSLAB, The University of Sydney

type of templates

- function template
- class template

Copyright © ViSLAB, The University of Sydney

how to make it

- “template” keyword

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

template <class type> void display(type arg) {
   cout << arg << endl;
}
```

- `template <class type>` specify template
- `void display(type arg)` function def.
- `type` : an arbitrary type.

Copyright © ViSLAB, The University of Sydney

how to make it

- "template" keyword

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

template <class type> void display(type arg) {
  cout << arg << endl;
}
```

- `void display(int arg)`
- `void display(double arg)`
- etc.

Copyright © ViSLAB, The University of Sydney

Template Parameter (T/type)

```
template <typename type> void display(type arg) {
   cout << arg << endl;
}
```

or

```
template <typename T> void display(T arg) {
   cout << arg << endl;
}
```

or

```
template <class T> void display(T arg) {
   cout << arg << endl;
}
```

Copyright © ViSLAB, The University of Sydney

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

template <typename T> void display(T arg) {
   cout << arg << endl;
}

int main() {
   display(30.0);
   string str = "hello";
   display(str);
   return 0;
}
```

- `void display(int arg)`
- `void display(string arg)`

Copyright © ViSLAB, The University of Sydney

explicit use

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

template <typename T> void display(T arg) {
   cout << arg << endl;
}

int main() {
   display<double>(30.0);
   string str = "hello";
   display<string>(str);
   return 0;
}
```

Copyright © ViSLAB, The University of Sydney

arbitrary type...

- statements in the template function
  - might not support certain types,
  - might give you compilation errors,
- `template <typename T> void display(T arg)`
  - `operator << (ostream)` must support `T`

Copyright © ViSLAB, The University of Sydney

```
template <typename T> T abs(T arg) {
    T temp;
    temp = (arg < 0) ? -arg : arg;
    return temp;
}
```

- imagine T as “int”, “double”, etc.
- T can be use anywhere in the function:
  - return type
  - arg type
  - local variable..

Copyright © ViSLAB, The University of Sydney

```
    template <typename T> T abs(T arg) {
    T temp;
    temp = (arg < 0) ? -arg : arg;
    return temp;
}

int main() {
    int i = -30;
    double d = -4343.88;
    i = abs(i);
    d = abs(d);

    return 0;
}
```

Copyright © ViSLAB, The University of Sydney

```
template <typename T> T abs(T arg) {
   T temp;
   temp = (arg < 0) ? -arg : arg;
   return temp;
}

int main() {
   char *p = “hello”;
   p = abs(p);

   return 0;
}
```

Copyright © ViSLAB, The University of Sydney

explicit specialisation

- create a special function for char *
- don’t specify <typename T>..just <>

Copyright © ViSLAB, The University of Sydney

```
   template <typename T> T abs(T arg) {
   T temp;
   temp = (arg < 0) ? -arg : arg;
   return temp;
}

template <> char *abs(char *arg) {
    // some implemenation to abs a string
    ...
}

int main() {
   char *p = "hello";
   p = abs(p);

   return 0;
}
```

Copyright © ViSLAB, The University of Sydney

explicit specialisation

- abs(“hello”)
  - not instantiation from a template
  - just call the specialised function.

- if there is a pre-defined function (like abs(int)),
- predefined function will becalled
  - abs(int) ..... std-lib’s abs function
  - abs(double) ..... function template’s abs()

Copyright © ViSLAB, The University of Sydney

how can I call abs(int)?

- want to call function template abs(int)

```
abs<int>(i);
```

- When you use a function template
  - make sure to check the desired f.t. is actually called.

Copyright © ViSLAB, The University of Sydney

how about macro?

- `#define abs(a) (((a)<0)?-(a):(a))`

- char literal replacement.
- side effects.
- explicit specialisation?

Copyright © ViSLAB, The University of Sydney

many parameters?

```
template<class T, class U, class V> void bar(T x, U y, V
z) {
  ...
}

string str = "hello";
int i = 20;
double d = 30.5;
bar(str, i, d);

bar<string, int, double>(str, i, d);
```

Copyright © ViSLAB, The University of Sydney

class template

- function template :
  - you specify data types the function can use.

- if you specify data type a class can use
  - class template

Copyright © ViSLAB, The University of Sydney

```
   class MultiString {
   string s1, s2;
public:
   MultiString(const string& a, const string& b) : s1(a), s2(b) {}
   string getS1() const {return s1;}
   string getS2() const {return s2;}
};
```

Copyright © ViSLAB, The University of Sydney

```
template <class T> class MultiData {
   T s1, s2;
public:
   MultiData(const T& a, const T& b) : s1(a), s2(b) {}
   T getS1() const {return s1;}
   T getS2() const {return s2;}
};
```

Copyright © ViSLAB, The University of Sydney

```
   template <class T> class MultiData {
   T s1, s2;
public:
   MultiData(const T& a, const T& b);
   T getS1() const {return s1;}
   T getS2() const {return s2;}
};

template <class T> MultiData<T>::MultiData(const T& a, const
T& b) : s1(a), s2(b) {}
```

Copyright © ViSLAB, The University of Sydney

```
template <class T, class U> class MultiData {
   T s1;
   U s2;
public:
   MultiData(const T& a, const U& b);
   T getS1() const {return s1;}
   U getS2() const {return s2;}
};

template <class T, class U> MultiData<T, U>::MultiData(const
T& a, const U& b) : s1(a), s2(b) {}
```

Copyright © ViSLAB, The University of Sydney

```
int main() {
string str1 = "hello";
string str2 = "world";
int i = 20, j = 40;
double d = 88.3, e = 2.0;

MultiData<string, string> ssd(str1, str2);
MultiData<string, double> sdd(str1, e);
MultiData<int, int> sdd(i, j);
}
```

Copyright © ViSLAB, The University of Sydney

```
int main() {
    string str1 = "hello";
    string str2 = "world";
    int i = 20, j = 40;
    double d = 88.3, e = 2.0;

    MultiData<string, string> *ssd;
    ssd = new MultiData<string, string>(str1, str2);

    MultiData<string, double> *sdd;
    sdd = MultiData<string, double>(str1, e);
}
```

•

Copyright © ViSLAB, The University of Sydney

```
int main() {
MultiData<MultiData<int, string>, MultiData<double, string> >
   isds(MultiData<int, string>(12, "hello"),
        MultiData<double, string>(3.4, "world"));
}
```

Copyright © ViSLAB, The University of Sydney