

## CIS 3700 Tutorial

Teaching Assistants: Robert Collier and Wei Wang

Email: [ta3700@cis.uoguelph.ca](mailto:ta3700@cis.uoguelph.ca)

Office Hours: Monday 11:30 - 12:30 (Robert)  
Friday 11:30 - 12:30 (Wei)  
... or by appointment

Tutorials: MacKinnon 223  
Thursday 11:30 - 12:30  
(Optional)

## Reference/Dereference Operator Review

alpha = 3014

**alpha**

		<b>3014</b>		
	1592	1593	1594	

beta = alpha  
gamma = &(alpha)  
delta = \*(alpha)

**beta**

		<b>3014</b>		
	6535	6536	6537	

**gamma**

		<b>1593</b>		
	8979	8980	8981	

**delta**

		<b>4626</b>		
	3238	3239	3240	

		<b>4626</b>		
	3013	3014	3015	

# Inheritance

"creation of classes which are derived from other classes"

"automatically include some members from the 'parent' "

```
class CPoint {
    int x;
    int y;
public:
    CPoint() {
        x = 0;
        y = 0;
    }
    CPoint(int a, int b) {
        x = a;
        y = b;
    }
    void setX(int a) { x = a; }
    void setY(int b) { y = b; }
    int getX() { return x; }
    int getY() { return y; }
};

class CPoint3D : public CPoint {
    int z;
public:
    CPoint3D() {
        setX(0);
        setY(0);
        z = 0;
    }
    CPoint3D(int a, int b, int c) {
        setX(a);
        setY(b);
        z = c;
    }
    void setZ(int c) { z = c; }
    int getZ() { return z; }
};
```

## Virtual Members

"a class member that can be redefined in its derived classes"

"member with same name as base class can be called from pointer"

```
class CPolygon {
    int width;
    int height;
public:
    void set(int a, int b) {
        width = a;
        height = b;
    }
    virtual int area() {
        return 0;
    }
};

class CRectangle: public CPolygon {
public:
    int area () {
        return (width * height);
    }
};

class CTriangle: public CPolygon {
public:
    int area () {
        return (width * height / 2);
    }
};

int main () {
    CRectangle A_Rectangle;
    CPolygon A_Polygon;
    CPolygon * First_Polygon = &A_Rectangle;
    CPolygon * Second_Polygon = &A_Polygon;
    First_Polygon->set(4,5);
    Second_Polygon->set(4,5);
    printf("Area 1 = %d\n", First_Polygon->area());
    printf("Area 2 = %d\n", Second_Polygon->area());
    return 0;
}
```

## C++ - Abstraction

"classes containing a pure virtual function are abstract base classes"

" the class not fully implemented so instances can't be created"

```
class CPolygon {
    int width;
    int height;
public:
    void set(int a, int b){
        width = a;
        height = b;
    }
    virtual int area() = 0;
};

class CRectangle: public CPolygon {
public:
    int area () {
        return (width * height);
    }
};

class CTriangle: public CPolygon {
public:
    int area () {
        return (width * height / 2);
    }
};

int main () {
    CRectangle A_Rectangle;
    CTriangle A_Triangle;
    CPolygon * First_Polygon = &A_Rectangle;
    CPolygon * Second_Polygon = &A_Triangle;
    First_Polygon->set(4,5);
    Second_Polygon->set(4,5);
    printf("Area 1 = %d\n", First_Polygon->area());
    printf("Area 2 = %d\n", Second_Polygon->area());
    return 0;
}
```

## STL - Iterators

" a generalization of pointers - objects that point to other objects"

" the class not fully implemented so instances can't be created"

```
using namespace std;
```

```
vector<int> v;
```

```
...
```

```
v.push_back(1);
```

```
v.push_back(2);
```

```
v.push_back(3);
```

```
for(int y=0; y < v.size(); y++){
```

```
    cout<<v[y]<<" ";
```

```
}
```

```
using namespace std;
```

```
vector<int> v;
```

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

```
...
```

```
v.push_back(1);
```

```
v.push_back(2);
```

```
v.push_back(3);
```

```
for(i = v.begin(); i != v.end(); i++){
```

```
    cout<<*i<<" ";
```

```
}
```

## STL - Lists

```
#include <list>
...
using namespace std;

int main(){

    list<int> list1;
    int value1 = 10;
    int value2 = -3;
    list1.push_back (value1);
    list1.push_back (value2);
    list1.push_back (5);
    list1.push_back (1);
    cout << endl << "List values:" << endl;
    while (list1.size() > 0){
        int value = list1.front();
        cout << value << endl;
        list1.pop_front();
    }
    return 0;
}
```

... or you could also make a new class for the type (instead of int)

if you go this route, you **MUST** overload the =, ==, and < operators

you **SHOULD** also ensure your class is **NICE**, that is, supports:

1. a copy constructor
2. an assignment operator
3. an equality operator
4. an inequality operator

## STL - Lists

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

class AAA {
public:
    int x;
    int y;
    float z;
    AAA() { x = y = z = 0; }
    AAA(const AAA &);
    ~AAA(){};
    AAA &operator=(const AAA &rhs);
    int operator==(const AAA &rhs) const;
    int operator<(const AAA &rhs) const;
};

AAA::AAA(const AAA &copyin) {
    x = copyin.x;
    y = copyin.y;
    z = copyin.z;
}

AAA& AAA::operator=(const AAA &rhs) {
    this->x = rhs.x;
    this->y = rhs.y;
    this->z = rhs.z;
    return *this;
}

int AAA::operator==(const AAA &rhs) const
{
    if( this->x != rhs.x) return 0;
    if( this->y != rhs.y) return 0;
    if( this->z != rhs.z) return 0;
    return 1;
}

int AAA::operator<(const AAA &rhs) const {
    if(this->x == rhs.x && this->y == rhs.y && this->z < rhs.z){
        return 1;
    }
    if( this->x == rhs.x && this->y < rhs.y) return 1;
    if( this->x < rhs.x ) return 1;
    return 0;
}
```

## STL - Lists

```
int main() {
    list<AAA> L;
    AAA Ablob ;
    Ablob.x=7;
    Ablob.y=2;
    Ablob.z=4.2355;
    L.push_back(Ablob);
    Ablob.x=5;
    L.push_back(Ablob);
    Ablob.z=3.2355;
    L.push_back(Ablob);
    Ablob.x=3;
    Ablob.y=7;
    Ablob.z=7.2355;
    L.push_back(Ablob);
    list<AAA>::iterator i;

    cout << "Before Sorting: " << endl;
    for(i=L.begin(); i != L.end(); ++i){
        cout << (*i).x << " " << (*i).y << " " << (*i).z << " ";
        cout << endl;
    }
    cout << "After Sorting: " << endl;
    L.sort();
    for(i=L.begin(); i != L.end(); ++i){
        cout << (*i).x << " " << (*i).y << " " << (*i).z << " ";
        cout << endl;
    }
    return 0;
}
```

----- OUTPUT -----

Before Sorting:

```
7 2 4.2355
5 2 4.2355
5 2 3.2355
3 7 7.2355
```

After Sorting:

```
3 7 7.2355
5 2 3.2355
5 2 4.2355
7 2 4.2355
```

Press any key to continue

## STL - Stacks

```
#include <iostream>
#include <stack>

int main(){

    stack<int> s;
    int i;

    for (i = 1; i <= 5; i++) {
        s.push(i);
    }

    cout<<"Stack size = "<<s.size()<<endl;

    cout<<"Popping the stack = "<<endl;

    while (!s.empty()){
        cout<<s.top()<<" ";
        s.pop();
    }

    cout<<endl;
    return 0;
}
```

----- OUTPUT -----

```
Stack size = 5
Popping the stack =
5 4 3 2 1
Press any key to continue
```

**the STL Queue is, understandably, very similar to the Stack**

**there is also an STL Map that you may find useful**

## Sites Worth Visiting

### C++

[www.cplusplus.com/doc/tutorial/](http://www.cplusplus.com/doc/tutorial/)

### C++ and STL

[www.cppreference.com/index.html](http://www.cppreference.com/index.html)

[www.fredosaurus.com/notes-cpp/](http://www.fredosaurus.com/notes-cpp/)

### STL

[www.infosys.tuwien.ac.at/Research/Component/tutorial/prwmain.htm](http://www.infosys.tuwien.ac.at/Research/Component/tutorial/prwmain.htm)

### ... and finally

1. Something you should know about the 8-puzzle ...
2. To ensure correct submission ...
3. Any questions about the assignment so far ...