C++ Class and Object

Dr. Md. Humayun Kabir
CSE Department, BUET

C++ Class

- C++ Classes are the logical abstraction or model of C++ Objects
- A Class declaration defines a new type
- It determines what an object of that type will look like
- It determines the nature of the data and functions of that type
- Classes must be defined before creating the objects, i.e., objects cannot be created without the classes
- Definition of a class does not create any physical objects rather a logical abstraction

C++ Class (cont…)

Classes are generally declared using the keyword **class**

```
class class_name {
  access_specifier_1:
        member1;
        member2;
  access_specifier_2:
        member3;
        member4; ...
} object_names;
```

C++ Class (cont…)

```
class CRectangle {
    int x, y;
    public:
          void set_values (int,int);
          int area () {return (x*y);}
};
void CRectangle::set_values (int a, int b) {
     x = a; y = b;
}
```

C++ Objects

- C++ Classes are used as the type specifier to create C++ Objects

CRectangle recta, rectb;

- An object declaration creates a physical entity of its class type, i.e., occupies memory space
- Each object has its own copy of data and functions
- Data and functions of two objects are independent even if they are created from the same class

Objects Example

```
#include <iostream>
using namespace std;
class CRectangle {
      int width, height;
      public:
        void set_values (int,int);
        int area () {return (width*height);}
};
void CRectangle::set_values (int a, int b) {
      width = a; height = b;
}
int main () {
      CRectangle recta, rectb;
      recta.set_values (3,4);
      rectb.set_values (5,6);
      cout << "recta area: " << recta.area() << endl;
      cout << "rectb area: " << rectb.area() << endl;
      return 0;
}
```

Pointers to classes

```
int main () {
      CRectangle a, *b, *c;
      CRectangle d[2];
      b= new CRectangle;
      c= &a;
      a.set_values (1,2);
      b->set_values (3,4);
      d[0]-set_values (5,6);
      d[1].set_values (7,8);
      cout << "a area: " << a.area() << endl;
      cout << "*b area: " << b->area() << endl;
      cout << "*c area: " << c->area() << endl;
      cout << "d[0] area: " << d[0].area() << endl;
      cout << "d[1] area: " << d[1].area() << endl;
      delete b;
      return 0;
}
```

Constructor Function

```
#include <iostream>
using namespace std;
class CRectangle {
      int width, height;
      public:
         CRectangle (int,int);
         int area () {return (width*height);}
};
CRectangle::CRectangle (int a, int b) {
      width = a; height = b;
}
int main () {
      CRectangle recta (3,4);
      CRectangle rectb (5,6);
      cout << "recta area: " << recta.area() << endl;
      cout << "rectb area: " << rectb.area() << endl;
      return 0;
}
```

Destructor Function

```
#include <iostream>
using namespace std;
class CRectangle {
    int *width, *height;
    public:
        CRectangle (int,int);
        ~CRectangle ();
        int area () {return (*width * *height);}
};

CRectangle::CRectangle (int a, int b) {
     width = new int;
    height = new int;
    *width = a;
    *height = b;
}
```

Destructor Function (cont…)

```
CRectangle::~CRectangle () {
    delete width;
    delete height;
}

int main () {
    CRectangle recta (3,4), rectb (5,6);
    cout << "recta area: " << recta.area() << endl;
    cout << "rectb area: " << rectb.area() << endl;
    return 0;
}
```

Assigning Objects

```
#include <iostream>
using namespace std;
class CRectangle {
      int width, height;
      public:
         CRectangle (int,int);
         int area () {return (width*height);}
};
CRectangle::CRectangle (int a, int b) {
      width = a; height = b;
}
int main () {
      CRectangle recta (3,4);
      CRectangle rectb (5,6);
      rectb=recta;
      cout << "recta area: " << recta.area() << endl;
      cout << "rectb area: " << rectb.area() << endl;
      return 0;
}
```

Assigning Objects (cont…)

```
#include <iostream>
using namespace std;
class CRectangle {
    int *width, *height;
    public:
        CRectangle (int,int);
        ~CRectangle ();
        int area () {return (*width * *height);}
};

CRectangle::CRectangle (int a, int b) {
     width = new int;
    height = new int;
    *width = a;
    *height = b;
}
```

Assigning Objects (cont…)

```
CRectangle::~CRectangle () {
    delete width;
    delete height;
}

int main () {
    CRectangle recta (3,4);
    CRectangle rectb (5,6);
    rectb=recta;
    cout << "recta area: " << recta.area() << endl;
    cout << "rectb area: " << rectb.area() << endl;
    return 0;
}
```

Objects passing to and returned from Function

```
#include <iostream>
using namespace std;
class CRectangle {
      int width, height;
      public:
         CRectangle (int,int);
         int area () {return (width*height);}
};
CRectangle::CRectangle (int a, int b) {
      width = a; height = b;
}
CRectangle larger(CRectangle recta, CRectangle rectb){
      if(recta.area()>rectb.area())
               return recta;
      else
               return rectb;
}
```

Objects passing to and returned from Function (cont…)

```
int main () {
      CRectangle recta (3,4);
      CRectangle rectb (5,6);
      CRectangle rect_larger(0,0);
      rect_larger=larger(recta, rectb);
      cout << "recta area: " << recta.area() << endl;
      cout << "rectb area: " << rectb.area() << endl;
      cout << "rect_larger area: " << rect_larger.area() << endl;

      return 0;
}
```

Objects passing to and returned from Function (cont…)

```
#include <iostream>
using namespace std;
class CRectangle {
    int *width, *height;
    public:
        CRectangle (int,int);
        ~CRectangle ();
        int area () {return (*width * *height);}
};

CRectangle::CRectangle (int a, int b) {
     width = new int;
    height = new int;
    *width = a;
    *height = b;
}
```

Objects passing to and returned from Function (cont…)

```
CRectangle::~CRectangle () {
    delete width;
    delete height;
}
CRectangle larger(CRectangle ra, CRectangle rb){
    if(ra.area()>rb.area())
        return ra;
    else
        return rb;
}
int main () {
    CRectangle recta (3,4), rectb (5,6);
    CRectangle rect_larger(0,0);
    rect_larger=larger(recta, rectb); //this will cause the program to crash
    cout << "recta area: " << recta.area() << endl;
    cout << "rectb area: " << rectb.area() << endl;
    cout << "rect_larger area: " << rect_larger.area() << endl;
    return 0;
}
```

Friend Function

```
#include <iostream>
using namespace std;
class CRectangle {
      int width, height;
      public:
         CRectangle (int,int);
         int area () {return (width*height);}
         friend bool isSquare(CRectangle rect);
};

CRectangle::CRectangle (int a, int b) {
      width = a; height = b;
}

bool isSquare(CRectangle rect){
      retrun rect.width==rect.height? true:false;
}
```

Friend Function

```
int main () {
    CRectangle recta (3,4);
    CRectangle rectb (5,5);
    if(isSquare(recta))
        cout<<“recta is a square”<<endl;
    if(isSquare(rectb))
        cout<<“rectb is a square”<<endl;
    cout << "recta area: " << recta.area() << endl;
    cout << "rectb area: " << rectb.area() << endl;
    return 0;
}
```

Friend Function

```
class Node {
private: int data;
int key;
// ...
friend int BinaryTree::find();
};
```

Friend Function

```
class BinaryTree {
    private:
    Node *root;
    int find(int key);
};
int BinaryTree::find(int key) {
    if(root->key == key) {
            return root->data;
    }
    ……
    ……
}
```

Friend Class

```
class Node {
    private:
    int data;
    int key;
    // ...
    friend class BinaryTree;
};
```

Friend Class

```
class BinaryTree {
      private:
      Node *root;
      int find(int key);
      bool isNull();
};
int BinaryTree::find(int key) {
      if(root->key == key) {
            return root->data;
      }
      ……….
}
bool isNull(){
      return root->key==NULL? true : false;
}
```

Special Pointer *this*

```
#include <iostream>
using namespace std;
class CRectangle {
    int width, height;
    public:
        CRectangle (int,int);
        int area () {return (width*height);}
};

CRectangle::CRectangle (int width, int height) {
    this->width = width; this->height = height;
}
```

Special Pointer *this*

```
int main () {
    CRectangle recta (3,4);
    CRectangle rectb (5,5);
        cout << "recta area: " << recta.area() << endl;
    cout << "rectb area: " << rectb.area() << endl;
    return 0;
}
```

Special operator *new* and *delete*

p-var=new type;

delete p-var;

p-var=new type[10];

delete [] p-var;

- *Advantage of new operator over **malloc**() function call*
  - Automatically allocates enough memory to hold an object of the specified type, do not need to use **sizeof** operator
  - Automatically returns a pointer of the specified type, do not to use an explicit type cast
- On failure new returns either NULL or an exception

Special operator *new* and *delete*

```
#include <iostream>
using namespace std;
int main(){
    int *p= new int;
    cout<<“Eneter an integer number: “;
    cin>>*p;
    cout<<“Entered number is <<*p<<endl;
    delete p;
    return 0;
}
```

Special operator *new* and *delete*

```
#include <iostream>
using namespace std;
int main(){
    int *p= new int(100);
    cout<<“Value in the pointer is <<*p<<endl;
    delete p;
    return 0;
}
```

Special operator *new* and *delete*

```
#include <iostream>
using namespace std;
int main(){
	int *p= new int[100];
	int i;
	for(i=0;i<100;i++){
		p[i]=i;
	}
	cout<<“Values in the array are <<endl;
	for(i=0;i<100;i++){
		cout<<p[i]<<endl;
	}
	delete [] p;
	return 0;
}
```

Special operator *new* and *delete*

```
#include <iostream>
using namespace std;
class CRectangle {
    int width, height;
    public:
        CRectangle (int,int);
        int area () {return (width*height);}
};

CRectangle::CRectangle (int width, int height) {
    this->width = width; this->height = height;
}
```

Special operator *new* and *delete*

```
int main(){
    CRectangle *rp=new CRectangle(3,4);
    cout<<“The area of the rectangle is “<<rp->area()<<endl;
    delete rp;
    return 0;
}
```

Array of Objects

```
#include <iostream>
using namespace std;
class Circle{
        double rad;
        public:
                void setRad(double r){rad=r;}
               double area(){return 3.14*rad*rad;}
}


int main(){
        Circle crls[2];
        crls[0].setRad(7.44);
        crls[1].setRad(3.65);
        cout<<“The area of the first circle is “<<crls[0].area()<<endl;
        cout<<“The area of the second circle is “<<crls[1].area()<<endl;
        return 0;
}
```

Array of Objects

```
#include <iostream>
using namespace std;
class Circle{
      double rad;
      public:
            void setRad(double r){rad=r;}
           double area(){return 3.14*rad*rad;}
}

int main(){
      Circle *crls=new Circle[2];
      crls[0].setRad(7.44);
      crls[1].setRad(3.65);
      cout<<“The area of the first circle is “<<crls[0].area()<<endl;
      cout<<“The area of the second circle is “<<crls[1].area()<<endl;
      delete [] crls;
      return 0;
}
```

Array of Objects

```
#include <iostream>
using namespace std;
class Circle{
      double rad;
      public:
          Circle(double r){rad=r;}
          double area(){return 3.14*rad*rad;}
}


int main(){
      Circle crls[2]={7.44, 3.65};
      cout<<“The area of the first circle is “<<crls[0].area()<<endl;
      cout<<“The area of the second circle is “<<crls[1].area()<<endl;
      return 0;
}
```

Array of Objects

```
#include <iostream>
using namespace std;
class Circle{
      double rad;
      double x;
      double y;
      public:
         Circle(double r, double x, double y){rad=r; this->x=x; this->y=y;}
         double area(){return 3.14*rad*rad;}
}

int main(){
      Circle crls[2]={Circle(7.44, 0.0, 0.0), Circle(3.65, 0.5, 2.5)};
      cout<<"The area of the first circle is "<<crls[0].area()<<endl;
      cout<<"The area of the second circle is "<<crls[1].area()<<endl;
      return 0;
}
```

Reference

```
#include <iostream>
using namespace std;
void swapargs (int x, int y){
    int t;
    t=x;x=y;y=t;
}
int main(){
    int i, j;
    i=20; j=40;
    cout<<"i="<<i<<", "<<j<<endl;
    swapargs(i,j);          //will not swap the values
    cout<<"i="<<i<<", "<<j<<endl;
    return 0;
}
```

Reference

```
#include <iostream>
using namespace std;
void swapargs (int *x, int *y){
    int t;
    t=*x;*x=*y;*y=t;
}
int main(){
    int i, j;
    i=20; j=40;
    cout<<“i=”<<i<<“, “<<j<<endl;
    swapargs(&i,&j);
    cout<<“i=”<<i<<“, “<<j<<endl;
    return 0;
}
```

Reference

```
#include <iostream>
using namespace std;
void swapargs (int &x, int &y){
    int t;
    t=x;x=y;y=t;
}
int main(){
    int i, j;
    i=20; j=40;
    cout<<"i="<<i<<", "<<j<<endl;
    swapargs(i,j);
    cout<<"i="<<i<<", "<<j<<endl;
    return 0;
}
```

Passing Reference to Function

```
#include <iostream>
using namespace std;
class Circle{
        double rad;
        public:
          void setRad(double r){rad=r;}
          double getRad()(return rad;}
          double area(){return 3.14*rad*rad;}
 };

void expand(Circle c, double e){
        c.setRad(c.getRad()*e);
}
int main(){
        Circle c;
        c.setRad(2.7);
        cout<<“The area of the circle is “<<c.area()<<endl;
        expand(c,1.5); //will not change the circle
        cout<<“The area of the circle is “<<c.area()<<endl;
        return 0;
}
```

Passing Reference to Function

```
#include <iostream>
using namespace std;
class Circle{
        double rad;
        public:
           void setRad(double r){rad=r;}
           double getRad()(return rad;}
           double area(){return 3.14*rad*rad;}
 };

void expand(Circle *c, double e){
        c->setRad(c->getRad()*e);
}
int main(){
        Circle c;
        c.setRad(2.7);
        cout<<“The area of the circle is “<<c.area()<<endl;
        expand(&c,1.5);
        cout<<“The area of the circle is “<<c.area()<<endl;
        return 0;
}
```

Passing Reference to Function

```
#include <iostream>
using namespace std;
class Circle{
        double rad;
        public:
           void setRad(double r){rad=r;}
           double getRad()(return rad;}
           double area(){return 3.14*rad*rad;}
 };

void expand(Circle &c, double e){
        c.setRad(c.getRad()*e);
}
int main(){
        Circle c;
        c.setRad(2.7);
        cout<<“The area of the circle is “<<c.area()<<endl;
        expand(c,1.5);
        cout<<“The area of the circle is “<<c.area()<<endl;
        return 0;
}
```

Passing Reference to Function

```
#include <iostream>
using namespace std;
class Circle{
      double *rad;
      public:
        Circle(double r){rad=new double; *rad=r; cout<<“Constructing…..”<<endl;}
        ~Circle(){delete rad; cout<<“Destructing…..”<<endl;}
        void setRad(double r){*rad=r;}
        double getRad()(return *rad;}
        double area(){return 3.14*(*rad)*(*rad);}
 };
void expand(Circle c, double e){
      c.setRad(c.getRad()*e);
}
int main(){
      Circle c(2.7);
      cout<<“The area of the circle is “<<c.area()<<endl;
      expand(c,1.5);
      cout<<“The area of the circle is “<<c.area()<<endl;
      return 0;
}
```

Passing Reference to Function

```
#include <iostream>
using namespace std;
class Circle{
      double *rad;
      public:
        Circle(double r){rad=new double; *rad=r; cout<<“Constructing…..”<<endl;}
        ~Circle(){delete rad; cout<<“Destructing…..”<<endl;}
        void setRad(double r){*rad=r;}
        double getRad()(return *rad;}
        double area(){return 3.14*(*rad)*(*rad);}
 };
void expand(Circle &c, double e){
      c.setRad(c.getRad()*e);
}
int main(){
      Circle c(2.7);
      cout<<“The area of the circle is “<<c.area()<<endl;
      expand(c,1.5);
      cout<<“The area of the circle is “<<c.area()<<endl;
      return 0;
}
```

Returning Reference from Function

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

int &xref(){
    return x;
}

int main(){
    xref()=100;
    cout<<“x= “<<x<<endl;
    return 0;
}
```

cout<<“Thank You”<<endl;

cout<<“**Have a Good Day**”<<endl;