C++ Classes and Object Oriented Programming

CPSC 1070, Donald House, Clemson University
10/21/19

Some C code for handling 2D points

```c
#include <stdio.h>

typedef struct point{
  int x, y;
} Point;

void setpoint(Point *p, int x0, int y0){
  p->x = x0;
  p->y = y0;
}

void movepoint(Point *p, int dx, int dy){
  p->x += dx;
  p->y += dy;
}

void printpoint(Point *p){
  printf("(%d, %d)", p->x, p->y);
}
```

```c
int main(){
  Point p1;
  Point p2;
  int nx, ny;
  int dx, dy;

  printf("Enter p1: ");
  scanf("%d %d", &nx, &ny);
  setpoint(&p1, nx, ny);

  printf("Enter p2: ");
  scanf("%d %d", &nx, &ny);
  setpoint(&p2, nx, ny);

  printpoint(&p1);
  printf(", ");
  printpoint(&p2);
  printf("\n");

  printf("Enter p2 changes: ");
  while(scanf("%d %d", &dx, &dy) == 2){
    movepoint(&p2, dx, dy);
    printpoint(&p2);
    printf("\n");
  }

  return 0;
}
```

Critique of Point Code

The data structure:

```
typedef struct point{
  int x, y;
} Point;
```

Is defined separately from the functions operating on the data:

```
void setpoint(Point *p, int x0, int y0);
void movepoint(Point *p, int dx, int dy);
void printpoint(Point *p);
```

Also, pointers to the Point need to be passed around, and the functions need to be named to explicitly work on Points:

```
setpoint(&p2, nx, ny);
```

C++ remedies this by providing for classes, which are designed to package data structures with methods that operate on the data.

Some Nomenclature

- Class is a type whose description consists of declarations and definitions for both a data structure and operations on that data structure
- Method is an operation (i.e. a function) defined for a class
- Class Declaration: declarations for a data structure, and a set of methods that operate on the data structure.
- Class Definition: definitions (i.e. the actual code) for the set of methods for the class.
- Object is a variable of a given class

Example 2D Point Class

Declaration

```
class Point{
private:
  int x, y;

public:

  Point();

  void set(int x0, int y0);

  int getx();
  int gety();

  void move(int dx, int dy);

  void print();
};
```

The private region hides data and methods from access outside of the class's methods

The public region is accessible from outside

Class constructor, called whenever a Point object is created

Accessor methods that return hidden data

Example 2D Point Class

Definition

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

Point::Point(){
  set(0, 0);
}

void Point::set(int x0, int y0){
  x = x0;
  y = y0;
}

int Point::getx(){;
  return x;
}

int Point::gety(){
  return y;
}

void Point::move(int dx, int dy){
  x += dx;
  y += dy;
}

void Point::print(){
  cout << "(" << x << ", " << y << ")";
}
```

Without this, `cout` and other `iostream` variables and methods would have to be accessed like this: `std::cout`

The constructor for Point makes sure that the Point's coordinates are both 0 by default

Within a method of the class, the local namespace consists of the subfields of the class, as well as the parameters and local variables of the method.

Note, that each method name is proceeded by Point:: This indicates that the method is a member of the class Point, and not just a regular function

Example 2D Point Class

Usage

```cpp
int main(){
  Point p1;
  Point p2;
  int nx, ny;
  int dx, dy;

  cout << "Enter p1: ";
  cin >> nx >> ny;
  p1.set(nx, ny);

  cout << "Enter p2: ";
  cin >> nx >> ny;
  p2.set(nx, ny);

  p1.print();
  cout << ", ";
  p2.print();
  cout << endl;

  cout << "Enter p2 changes: ";
  cin >> dx >> dy;
  while(!cin.eof()){
    p2.move(dx, dy);
    p2.print();
    cout << endl;
    cin >> dx >> dy;
  }

  return 0;
}
```

An object of type Point is created just like any other variable, except that the class constructor is automatically called when the variable is created.

This is how a method is called for an object. The notation is identical with how the subfields of a struct are accessed. What happens is that `set()` is called with its local namespace set to the subfields of the object `p1`.

To call a method on an object, the notation is:
objname.method()

Example 2D Point Class

Declaration

```
class Point{
private:
  int x, y;

public:
  Point();

  void set(int x0, int y0);
  int getx();
  int gety();

  void move(int dx, int dy);

  void print();
};
```

Definition

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

Point::Point(){
  set(0, 0);
}

void Point::set(int x0, int y0){
  x = x0;
  y = y0;
}

int Point::getx(){;
  return x;
}

int Point::gety(){
  return y;
}

void Point::move(int dx, int dy)
{
  x += dx;
  y += dy;
}

void Point::print(){
  cout << "(" << x << ", "
       << y << ")";
}
```

Usage

```
int main(){
  Point p1;
  Point p2;
  int nx, ny;
  int dx, dy;

  cout << "Enter p1: ";
  cin >> nx >> ny;
  p1.set(nx, ny);

  cout << "Enter p2: ";
  cin >> nx >> ny;
  p2.set(nx, ny);

  p1.print();
  cout << ", ";
  p2.print();
  cout << endl;

  cout << "Enter p2 changes: ";
  cin >> dx >> dy;
  while(!cin.eof()){
    p2.move(dx, dy);
    p2.print();
    cout << endl;
    cin >> dx >> dy;
  }

  return 0;
}
```

Code is Normally Distributed Over Several Files

Declaration

Point.h

```cpp
class Point{
private:
  int x, y;

public:
  Point();

  void set(int x0, int y0);
  int getx();
  int gety();

  void move(int dx, int dy);

  void print();
};
```

Definition

Point.cpp

```cpp
#include “Point.h”
#include <iostream>
using namespace std;

Point::Point(){
  set(0, 0);
}

void Point::set(int x0, int y0){
  x = x0;
  y = y0;
}

int Point::getx(){;
  return x;
}

int Point::gety(){
  return y;
}

void Point::move(int dx, int dy){
  x += dx;
  y += dy;
}

void Point::print(){
  cout << "(" << x << ", "
       << y << ")";
}
```

Usage

trypoint.cpp

```cpp
#include “Point.h”

int main(){
  Point p1;
  Point p2;
  int nx, ny;
  int dx, dy;

  cout << "Enter p1: ";
  cin >> nx >> ny;
  p1.set(nx, ny);

  cout << "Enter p2: ";
  cin >> nx >> ny;
  p2.set(nx, ny);

  p1.print();
  cout << ", ";
  p2.print();
  cout << endl;

  cout << "Enter p2 changes: ";
  cin >> dx >> dy;
  while(!cin.eof()){
    p2.move(dx, dy);
    p2.print();
    cout << endl;
    cin >> dx >> dy;
  }

  return 0;
}
```

With a Makefile Like This

Makefile

```
CC = g++
CFLAGS = -g
PROJECT = trypoint

DEPS = Point.h
OBJ = $(PROJECT).o Point.o

%.o: %.cpp $(DEPS)
	$(CC) $(CFLAGS) -c -o $@ $<

$(PROJECT): $(OBJ)
	$(CC) $(CFLAGS) -o $@ $^

clean:
	rm -f *.o
	rm -f *~
	rm -f $(PROJECT)
```

This forces both `Point.cpp` and `trypoint.cpp` to be compiled to `.o` files

Example C++ Program

Point Exercising Program

`Download from the schedule page`