Copyright Warning

ViSLAB
High-Performance Visualization, Communication, Computing

Object Oriented Design

Week 2-2

Member Variable/Function

- In C++, you achieve OO with “class”.
- Class is an extension of struct in C.
- In C, struct can only take member variables.
- C++’s “class” can take:
  - member variables, and
  - member functions

Defining Class

- Since "class" is an extension of the struct, it is defined in the similar manner.

```
class ClassName {
        member-list
};
```

Struct ➔ Class

- struct (structure) is a "named" blob of data

```
class Monster {
   char name[64];  // the name of the monster
   int power;    // the current power
   int status;     // the current status such as live/dead/paused
};
```

- class

```
class Monster {
   char name[64]; // the name of the monster
   int power;     // the current power
   int status;    // the current status such as live/dead/paused

public:
   char* GetName();
   int GetPower();
   int GetStatus();
   int Walk();
};
```

“class” = “struct” + α ?

- according to C++ spec: the followings are “class”
  - class
  - struct
  - union
- by default, all members of struct/union are public

```
struct Monster {
   char name[64];   // the name of the monster
   int power;    // the current power
   int status;      // the current status such as live/dead/paused
};
```

- is equivalent to

```
class Monster {
public:
   char name[64];  // the name of the monster
   int power;    // the current power
   int status;      // the current status such as live/dead/paused
};
```

struct == class (in C++)

```
class Monster {
private:
   char name[64]; // the name of the monster
   int power;     // the current power
   int status;    // the current status such as live/dead/paused

public:
   void DisplayData();  // display the current info.
   int Walk();     // make the monster walk (change its position)
};
```

Implementation of member functions

- The class definition only lists prototypes of member functions.
- A function prototype:
  - name,
  - return type
  - storage class
  - arguments

Implementation of member functions

- You need to provide the actual implementation of member functions

```
void Monster::DisplayData() {
   cout << name << endl;
   cout << power << endl;
   cout << status << endl;

   return;
}
```

- You need to specify who owns this function.

Implementation of member functions

- In C++, you would separate a class definition and its member function's implementation. (like some OOPL...Objective-C)
- Class definition in a header file (.h)
- Implementations in another file (.cpp)

Inline Function

- If the member function is small.

```
class Monster {
private:
   char name[64];// the name of the monster
   int power;    // the current power
   int status;   // the current status such as live/dead/paused

public:
   void DisplayData() {
      cout << name << endl;
      cout << power << endl;
      cout << status << endl;
   }
   int Walk();
};
```

Using Class

- just like “struct”, defining “class” does not create its instance.
  - `ClassName variable_name;`
  - `Monster pikachu;`

- To access (accessible) members:
  - `class_variable_name.member_name`

```
strcopy(pikachu.name, "Pikachu");
pikachu.power = 100;
pikachu.status = 0;
pikachu.DisplayData()
```

Separating Sources

- in Monster.h

```
class Monster {
private:
    char name[64];  // the name of the monster
    int power;      // the current power
    int status;     // the current status such as live/dead/paused

public:
    void DisplayData();
    int Walk();
};
```

Separating Sources

- in Monster.cpp

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

void Monster::DisplayData() {
    cout << name << "\n";
    cout << power << "\n";
    cout << status << "\n";

    return;
}
```

Separating Sources

- in Game.cpp

```
#include <string.h>
#include "Monster.h"

int main() {
    Monster pikachu;

    strcpy(pikachu.name, "Pikachu");
    pikachu.power = 100;
    pikachu.status = 0;

    pikachu.DisplayData();

    return 0;
}
```

Use of Inline Function

- in Monster.h

```
class Monster {
    public:
        char name[64];      // the name of the monster
        int power;          // the current power
        int status;         // the current status such as live/dead/paused

        void DisplayData() {
            cout << name << "\n";
            cout << power << "\n";
            cout << status << "\n";

            return;
        };
};
```

All in one

- in Game.cpp

```
#include <string.h>
#include <iostream.h>

class Monster {
    public:
        char name[64];    // the name of the monster
        int power;        // the current power
        int status;       // the current status such as live/dead/paused

        void DisplayData();
};

void Monster::DisplayData() {
    cout << name << "\n";
    cout << power << "\n";
    cout << status << "\n";

    return;
}

int main() {
    Monster pikachu;

    strcpy(pikachu.name, "Pikachu");
    pikachu.power = 100;
    pikachu.status = 0;

    pikachu.DisplayData();

    return 0;
}
```

Three Sacred Treasures

- Many OO books say OO is about
  - Encapsulation ➛ Class
  - Inheritance
  - Polymorphism

- Although these are very useful, you don't have to use them...

Polymorphism with overloading

- In C++, you are allowed to define multiple functions with the same name.

```
class Monster {
    public:
        char name[64];     // the name of the monster
        int power;         // the current power
        int status;        // the current status such as live/dead/paused

        void DisplayData();
        void GiveDamage(int value);
        void GiveDamage(float ratio);
};

void Monster::GiveDamage(int vlaue) {
    int tmp = power - value;
    power = (tmp < 0) ? 0 : tmp;
}

void Monster::GiveDamage(float ratio) {
    power = (ratio < 0) ? 0 : (power * ratio);
}
```

Polymorphism with overloading

- In C++, you are allowed to define multiple functions with the same name.

```
Monster aMon;

strcpy(aMon.name, "Pikachu");
aMon.power = 100;
aMon.status = 0;

int damage = 10;
float damageR = 0.3;

aMon.GiveDamage(10);
aMon.GiveDamage(0.3);
```

Polymorphism with overloading

- If the same name functions are not allowed...

```
class Monster {
    public:
        char name[64];    // the name of the monster
        int power;        // the current power
        int status;       // the current status such as live/dead/paused

        void DisplayData();
        void GiveDamageInt(int value);
        void GiveDamageFlt(float ratio);
};

void Monster::GiveDamageInt(int vlaue) {
    int tmp = power - value;
    power = (tmp < 0) ? 0 : tmp;
}

void Monster::GiveDamageFlt(float ratio) {
    power = (ratio < 0) ? 0 : (power * ratio);
}

..
Monster aMon;

strcpy(aMon.name, "Pikachu");
aMon.power = 100;
aMon.status = 0;

int damage = 10;
float damageR = 0.3;

aMon.GiveDamageInt(10);
aMon.GiveDamageFlt(0.3);
..
```

Pattern of Overloading

- There seem to be two basic patterns in using overloading:
  - overload a member function with a different argument data type
  - overload a member function with different arguments.

Different Data Types

```
class Monster {
public:
    char name[64];    // the name of the monster
    int power;        // health of the monster
    int status;       // the current status such as live/dead/paused

    void DisplayData();
    void GiveDamage(int value) {
        int tmp = power - value;
        power = (tmp < 0) ? 0 : tmp;
        return;
    }
    void GiveDamage(long value) {
        int tmp = power - value;
        power = (tmp < 0) ? 0 : tmp;
        return;
    }
    void GiveDamage(float ratio) {
        power = (ratio < 0) ? 0 : (power * ratio);
        return;
    }
    void GiveDamage(double ratio) {
        power = (ratio < 0) ? 0 : (power * ratio);
        return;
    }
};
```

Different Arguments

```
class Monster {
public:
    char name[64];    // the name of the monster
    int power;        // health of the monster
    int attack_power;        // the current attack power
    int healing_power;        // the current healing power
    int defence_power;        // the current defence power
    int status;       // the current status such as live/dead/paused

    void DisplayData();
    void GiveDamage(int damage) {          // simply lower the current power
        int tmp = power - damage;
        power = (tmp < 0) ? 0 : tmp;
        return;
    }
    void GiveDamage(int damage, int damage_attack, int damage_healing, int damage_defence) {
        int tmp = power - damage;
        power = (tmp < 0) ? 0 : tmp;

        tmp = attack_power - damage_attack;
        attack_power = (tmp < 0) ? 0 : tmp;

        tmp = healing_power - damage_healing;
        healing_power = (tmp < 0) ? 0 : tmp;

        tmp = defence_power - damage_defence;
        defence_power = (tmp < 0) ? 0 : tmp;

        return;
    }
};
```

Different Arguments

```
class Monster {
public:
    char name[64];    // the name of the monster
    int power;        // health of the monster
    int attack_power;        // the current attack power
    int healing_power;        // the current healing power
    int defence_power;        // the current defence power
    int status;       // the current status such as live/dead/paused

    void DisplayData();
    void GiveDamage(int damage) {          // simply lower the current power
        int tmp = power - damage;
        power = (tmp < 0) ? 0 : tmp;
        return;
    }
    void GiveDamage(int damage, int damage_attack, int damage_healing, int damage_defence) {
        GiveDamage(damage);                // call previously defined method function.

        int tmp = attack_power - damage_attack;
        attack_power = (tmp < 0) ? 0 : tmp;

        tmp = healing_power - damage_healing;
        healing_power = (tmp < 0) ? 0 : tmp;

        tmp = defence_power - damage_defence;
        defence_power = (tmp < 0) ? 0 : tmp;

        return;
    }
};
```

Object Array

```
int main(void) {
    Monster myMons[3];

    strcpy(myMons[0].name, "Pikachu");
    myMons[0].power = 100;
    myMons[0].status = 0;

    strcpy(myMons[1].name, "Neko");
    myMons[1].power = 50;
    myMons[1].status = 1;

    strcpy(myMons[2].name, "Kame");
    myMons[2].power = 30;
    myMons[2].status = 2;

    int i;
    for (i = 0; i < 3; i++) {
        myMons[i].DisplayData();
    }
    return 0;
}
```

Object Pointer

```
int main(void) {
    Monster pikachu;
    Monster* aMonster;

    aMonster = &pikachu;

    strcpy(aMonster->name, "Pikachu");
    aMonster->power = 100;
    aMonster->status = 0;

    aMonster->DisplayData();

    return 0;
}
```

```
#include <iostream.h>

class Monster {
    public:
        char name[64];     // the name of the monster
        int power;         // the current power
        int status;        // the current status such as live/dead/paused

        void DisplayData() {
            cout << name << "\n";
            cout << power << "\n";
            cout << status << "\n";
            return;
        };
};

class MyMonster {
    public:
        void UseMonster(Monster* aMon) {    //a member function that takes an object pointer
            aMon->DisplayData();
            return;
        };

        Monster* GetMonster();              //a member function that return an object pointer
};

Monster* MyMonster::GetMonster() {
    Monster *aMon = new Monster();

    strcpy(aMon->name, "Pikachu");
    aMon->power = 100;
    aMon->status = 0;

    return aMon;
}

int main(void) {
    Monster *pMon;
    MyMonster myMon;

    pMon = myMon.GetMonster();
    myMon.UseMonster(pMon);

    return 0;
}
```

Next week

- OOP and Encapsulation