```
/*
 * advanced programming
 * Alireza Akhavan Pour
 * akhavan@alirezaWeb.com
 * date: 1395/02/06
 */
int main()
{
    cout<< "In the name of God";
    Lecture_20 ();
    return 0;
}

string Lecture_20()
{
    cout << "1.Robofight-ISISRobot!" <<endl;
    cout << "2.Static " <<endl;
    cout << "1.Singleton" <<endl;
    return ":)"
}
```

1

ربات های داعشی!!!

در این مثال، رقابت روبات‌ها مانند قبل است با این تفاوت که برخی از روبات‌ها امکان عملیات انتحاری دارند:

به این ترتیب که در هر نوبت یا رفتار عادی خود را دارند یا به طور اتفاقی یک جهت را انتخاب کرده، به آن جهت حرکت می‌کنند تا وقتی یا به انتهای محوطه برسند یا به یک روبات دیگر که در این صورت آن روبات را از بین خواهند برد. در هر دو صورت، روباتی که عملیات انتحاری کرده است می‌میرد.

01_ISISRobot.cpp

2

```
class ISISRobot : public Robot {
public:
    ISISRobot(char i, int stm, int loc, Roboline* line)
        : Robot(i, stm, loc, line) {}

    void play_turn();
protected:
    void suicide();//entehari
};
```

ISIS زیرکلاسی از Robot است.

بازنویسی متد play_turn از کلاس Robot

اضافه کردن رفتار جدید عملیات انتحاری!

3

```
void ISISRobot::play_turn() {
    if (dead)
        return;
    if (rand() % 2 == 0)
        suicide();
    else
        Robot::play_turn();
}
```

به احتمال ۵۰٪ عملیات انتحاری صورت می‌گیرد

فراخوانی پیاده‌سازی play_turn از کلاس Robot

4

```
void ISISRobot::suicide() {
    int dir = rand() % 2;
    cout << id << (dir==DIR_LEFT ? '<' : '>') << ' ';
    int step;
    if (dir == DIR_LEFT)
        step = -1;
    else
        step = 1;

    int loc = location + step;
    while (!rl->invalid_loc(loc) && (rl->peek_at(loc) == NULL))
        loc += step;

    if (rl->peek_at(loc) != NULL)
        rl->peek_at(loc)->die();
    die();
}
```

5

تنها بخشی از مثال قبل که تغییر کرده است ساختن روبات جدید است که بعضی از روبات‌ها از نوع ISISRobot خواهند بود.

```
Roboline::Roboline(int size, int num_of_robots) : line(size) {
    for (int i = 0; i < num_of_robots; i++) {
        int loc = rand() % size;
        while (line[loc] != NULL)
            loc = rand() % size;

        Robot* new_robot;
        if (rand() % 2)
            new_robot = new ISISRobot('A'+i, rand() % MAX_POWER + 1, loc, this);
        else
            new_robot = new Robot('a'+i, rand() % MAX_POWER + 1, loc, this);

        robots.push_back(new_robot);
        line[loc] = new_robot;
    }
}
```

حدوداً نیمی از روبات‌ها با قابلیت عملیات انتحاری ایجاد می‌شوند.

دقت کنید در این جمله‌ی جایگزینی، سمت چپ از نوع اشاره‌گر به Robot است ولی سمت راست از نوع اشاره‌گر به ISISRobot. به خاطر این که ISISRobot زیرکلاس Robot است، این جایگزینی مشکل تایپ ندارد.

6

اعضای داده ای استاتیک کلاسها

هر شیءای از کلاس یک کپی از تمام اعضای داده ای آن کلاس را داراست. در بعضی از موارد فقط یک کپی از یک متغیر باید بین تمام اشیای یک کلاس مشترک باشد. کلمه کلیدی **استاتیک** برای این منظور و اهداف دیگر به کار میرود. وقتی کلمه کلیدی **static** را با اعضای داده ای کلاس به کار میبرید، به کامپایلر می گویید که فقط یک کپی از آن متغیر وجود خواهد داشت و تمام اشیای آن کلاس ، آن متغیر را به اشتراک می گذارند.برخلاف اعضای داده ای معمولی، فقط یک کپی از اعضای داده ای استاتیک وجود دارد. پس تمام اشیای آن کلاس از یک متغیر استفاده می کنند. حوزه متغیرهای استاتیک در یک کلاس است.

7

- **وقتی یک عضو داده ای static را در داخل کلاسی اعلان میکنید،** حافظه به آن اختصاص نمی یابد. در عوض باید در خارج از کلاس، آن را بصورت **عمومی** تعریف کنید. این کار با تعریف مجدد متغیر staticو با استفاده از عملگر تعیین کننده حوزه (::) برای شناسایی کلاسی که آن متغیر به آن کلاس تعلق دارد، انجام میشود. به این ترتیب حافظه به متغیر اختصاص می یابد.

8

- **متغیر عضو static** قبل از اینکه شیءای از کلاس آن ایجاد شود، وجود دارد. همچنین یکی از کاربردهای متغیر عضو static این است که دستیابی به منابع مشترکی را که توسط اشیای یک کلاس مورد استفاده قرار می گیرند، فراهم میسازد. کاربرد جالب دیگر این متغیر این است که تعداد اشیایی که از یک نوع کلاس خاص را که در حال استفاده اند نگهداری میکند. باید سعی کنید با استفاده از متغیرهای عضو static کمتر از متغیرهای عمومی استفاده کنید، زیرا متغیرهای عمومی ویژگی بسته بندی را در برنامه نویسی شیءگرا از بین میبرند.

9

توابع عضو استاتیک:

توابع عضو را نیز میتوان بصورت استاتیک تعریف کرد. ولی محدودیتهایی در این خصوص وجود دارد.

- آنها فقط میتوانند به سایر اعضای استاتیک کلاس مراجعه کنند. (به اعضای عادی دسترسی ندارند)
- این نوع توابع **نمیتوانند مجازی** باشند
- نمیتوانند حاوی اشاره گر **this** باشند.

زیرا اعضای داده ای استاتیک و توابع عضو استاتیک مستقل از هر شیء دیگری از کلاس وجود دارند.

10

```cpp
// static members in classes
#include <iostream>
using namespace std;

class Dummy {
  public:
    static int n;
    Dummy () { n++; };
};

int Dummy::n=0;

int main () {
  Dummy a;
  Dummy b[5];
  cout << a.n << '\n';
  Dummy * c = new Dummy;
  cout << Dummy::n << '\n';
  delete c;
  return 0;
}
```

```
6
7
```

02_static.cpp

11

مثالی از ایجاد خودکار ID برای هر شئ

```cpp
class User
{
  private:
  int id;
  static int next_id;

  public:
  static int next_user_id()
  {
    next_id++;
    return next_id;
  }
  int get_id(){return id;}
  /* More stuff for the class user */
  User()
  {
    id = User::next_id++; //or, id = User.next_user_id();
  }
};

int User::next_id = 0;
```

03_static_autoincrement.cpp

12

مثال کاربردی از استاتیک
Singleton Pattern

- میخواهیم تضمین کنیم که از یک کلاس تنها بتوان یک آبجکت (حالت ساده و متعارف سینگلتون) ایجاد کرد و همه ی درخواستها نیز تنها به همان یک آبجکت هدایت شوند و نیز بتوان یک دسترسی سراسری به آن داشت.

13

Ensuring a unique instance

شما می توانید عملیات کلاس را با تابع عضو استاتیک Instance از کلاس Singleton در C++ مشخص کنید.

کلاس Singleton متغیر عضو استاتیک _instance که شامل اشاره گر به نمونه واحد است را مشخص میکند.

کلاس Singleton اینگونه اعلام شده:

```
class Singleton {
  public:
    static Singleton* Instance();
  protected:
    Singleton();
  private:
    static Singleton* _instance;
};
```

14

Ensuring a unique instance

```
class Singleton {
  public:
    static Singleton* Instance();
  protected:
    Singleton();
  private:
    static Singleton* _instance;
};
```

```
Singleton* Singleton::_instance = 0;

Singleton* Singleton::Instance () {
    if (_instance == 0) {
        _instance = new Singleton;
    }
    return _instance;
}
```

پیاده سازی مربوطه:

04_static_singleton.cpp

15

```
Singleton* Singleton::_instance = 0;

Singleton* Singleton::Instance () {
    if (_instance == 0) {
        _instance = new Singleton;
    }
    return _instance;
}
```

```
int main()
{
    Singleton *sc1,*sc2;
    sc1 = Singleton::Instance();
    sc2 = Singleton::Instance();

    system("pause");
    return 0;
}
```

چون تا به حال آبجکتی ساخته نشده، یک آبجکت به صورت پویا یا داینامیک از کلاس میسازد
آن را در _instance گذاشته و در نهایت return میکند

16

```
Singleton* Singleton::_instance = 0;

Singleton* Singleton::Instance () {
    if (_instance == 0) {
        _instance = new Singleton;
    }
    return _instance;
}
```

```
int main()
{
    Singleton *sc1,*sc2;
    sc1 = Singleton::Instance();
    sc2 = Singleton::Instance();

    system("pause");
    return 0;
}
```

چون قبلا از این کلاس شئی ساخته شده
_instance دیگر ۰ نیست
پس شرط اجرا نمیشود و _instance که
اشاره گر به شئ قبلی است بر میگردد.

17

منابع:

- http://ramtung.ir/apnotes/html/12_Robofight.html
- http://www.sourcecodes.ir/post.php?id=211&title=اع
  ضای-داده-ای-استاتیک-کلاسها--در-سی-پلاس-پلاس-++C
- http://www.cplusplus.com/doc/tutorial/templates/
- http://www.cprogramming.com/tutorial/statickeyword.html

18