Memory Management 3 : Advance Pointers & References

Pertemuan 8

Outline

- Advance Pointers & References
  - Passing by References for Efficiency
  - Passing a **const** Pointer
  - References as an Alternative to Pointers
  - Don’t Return a Reference to an Object
  - Returning a Reference to an Object on the Heap

Passing Reference untuk Efisiensi

- Setiap kali melewatkan (pass) sebuah objek ke fungsi dengan nilai, salinan (copy) dari objek akan dibuat, setiap kali mengembalikan (return) objek dari fungsi dengan nilai, salinan lainnya akan dibuat. Semakin banyak objek dibuat oleh user, akan banyak memory yang dibutuhkan, dan program akan berjalan lebih lambat.
- Passing by reference akan menghindari pembuatan salinan dan memanggil copy constructor, hal ini lebih efisien. Di sisi lain, ia juga melewati obyek itu sendiri, dan dengan demikian memperlihatkan objek yang berubah pada fungsi yang dipanggil.

References adalah Alternatif Pointers

- Program dapat dibuat untuk memecahkan masalah dengan banyaknya membuat salinan, menyimpan panggilan ke copy constructor dan destructor. Menggunakan pointer konstan untuk benda konstan, sehingga pemecahan masalah yang disebut fungsi membuat perubahan objek dibolehkan untuk dilewatkan sebagai parameter.
- Metode ini masih agak rumit, namun, karena objek dilewatkan ke fungsi pointer. Karena Anda tahu parameter tidak akan pernah NULL, lebih mudah untuk bekerja dengan fungsi jika reference dapat dilewatkan daripada pointer.

Kapan Pointer, Kapan Reference ?

- Pada umumnya, programmer C++ sangat suka reference untuk pointer karena mereka lebih “bersih” dan lebih mudah digunakan. Reference tidak bisa dipindahkan, namun jika Anda perlu untuk menunjuk satu objek dan kemudian ke yang lain, Anda harus menggunakan pointer.
- Reference tidak bisa NULL, jadi jika ada perubahan pada objek tersebut apakah mungkin?, Anda harus menggunakan pointer daripada reference. Jika Anda ingin mengalokasikan memori dinamis dari Heap, Anda harus menggunakan pointer seperti yang dibahas pada pertemuan sebelumnya.

Mengembalikan Reference ke Object

- Setelah programmer C++ belajar tentang reference, mereka mungkin akan memiliki kecenderungan untuk tidak terkendali. Ingatlah bahwa reference merupakan “alias” yang mengacu ke beberapa objek lainnya. Jika Anda melewati sebuah reference ke dalam atau keluar dari fungsi, pastikan untuk bertanya "Apa objek saya aliasing, dan akan masih ada setiap kali itu digunakan?“ . Akan menjadi bahaya jika mengembalikan reference ke sebuah obyek yang tidak lagi ada.

1. Passing by References for Efficiency

ObjectRef.cpp

```
1: #include <iostream>
2:
3: class SimpleCat
4: {
5:         public:
6:                     SimpleCat(); // constructor
7:                     SimpleCat(SimpleCat&); // copy constructor
8:                     ~SimpleCat(); // destructor
9: };
10:
11: SimpleCat::SimpleCat()
12: {
13:        std::cout << "Simple Cat Constructor ...\n";
14: }
15:
16: SimpleCat::SimpleCat(SimpleCat&)
17: {
18:        std::cout << "Simple Cat Copy Constructor ...\n";
19: }
20:
21: SimpleCat::~SimpleCat()
22: {
23:        std::cout << "Simple Cat Destructor ...\n";
24: }
25:
26: SimpleCat FunctionOne(SimpleCat theCat);
27: SimpleCat* FunctionTwo(SimpleCat *theCat);
28:
```

...

```
29: int main()
30: {
31:       std::cout << “Making a cat ...\n”;
32:       SimpleCat Frisky;
33:       std::cout << “Calling FunctionOne ...\n”;
34:       FunctionOne(Frisky);
35:       std::cout << “Calling FunctionTwo ...\n”;
36:       FunctionTwo(&Frisky);
37:       return 0;
38: }
39:
40: // FunctionOne, passes by value
41: SimpleCat FunctionOne(SimpleCat theCat)
42: {
43:       std::cout << “Function One. Returning ...\n”;
44:       return theCat;
45: }
46:
47: // functionTwo, passes by reference
48: SimpleCat* FunctionTwo (SimpleCat *theCat)
49: {
50:       std::cout << “Function Two. Returning ...\n”;
51:       return theCat;
52: }
```

2. Passing a **const** Pointer

ConstPasser.cpp

```
1: #include <iostream>
2:
3: class SimpleCat
4: {
5:        public:
6:                  SimpleCat();
7:                  SimpleCat(SimpleCat&);
8:                  ~SimpleCat();
9:
10:                 int GetAge() const { return itsAge; }
11:                 void SetAge(int age) { itsAge = age; }
12:
13:       private:
14:                 int itsAge;
15: };
16:
17: SimpleCat::SimpleCat()
18: {
19:       std::cout << "Simple Cat Constructor ...\n";
20:       itsAge = 1;
21: }
22:
23: SimpleCat::SimpleCat(SimpleCat&)
24: {
25:       std::cout << "Simple Cat Copy Constructor ...\n";
26: }
27:
```

...

```
28: SimpleCat::~SimpleCat()
29: {
30:        std::cout << "Simple Cat Destructor ...\n";
31: }
32:
33: const SimpleCat * const
34: FunctionTwo (const SimpleCat *const theCat);
35:
36: int main()
37: {
38:        std::cout << "Making a cat ...\n";
39:        SimpleCat Frisky;
40:        std::cout << "Frisky is ";
41:        std::cout << Frisky.GetAge() << " years old\n";
42:        int age = 5;
43:        Frisky.SetAge(age);
44:        std::cout << "Frisky is ";
45:        std::cout << Frisky.GetAge() << " years old\n";
46:        std::cout << "Calling FunctionTwo ...\n";
47:        FunctionTwo(&Frisky);
48:        std::cout << "Frisky is ";
49:        std::cout << Frisky.GetAge() << " years old\n";
50:        return 0;
51: }
52:
53: // functionTwo, passes a const pointer
54: const SimpleCat * const
55: FunctionTwo (const SimpleCat * const theCat)
56: {
57:        std::cout << "Function Two. Returning ...\n";
58:        std::cout << "Frisky is now " << theCat->GetAge();
59:        std::cout << " years old \n";
60:        // theCat->SetAge(8); const!
61:        return theCat;
62: }
```

3. References as an Alternative to Pointers

RefPasser.cpp

```
1: #include <iostream>
2:
3: class SimpleCat
4: {
5:         public:
6:                     SimpleCat();
7:                     SimpleCat(SimpleCat&);
8:                     ~SimpleCat();
9:
10:                    int GetAge() const { return itsAge; }
11:                    void SetAge(int age) { itsAge = age; }
12:
13:        private:
14:                    int itsAge;
15: };
16:
17: SimpleCat::SimpleCat()
18: {
19:        std::cout << "Simple Cat Constructor...\n";
20:        itsAge = 1;
21: }
22:
23: SimpleCat::SimpleCat(SimpleCat&)
24: {
25:        std::cout << "Simple Cat Copy Constructor...\n";
26: }
27:
28: SimpleCat::~SimpleCat()
29: {
30:        std::cout << "Simple Cat Destructor...\n";
31: }
32:
```

...

```
33: const SimpleCat & FunctionTwo (const SimpleCat & theCat);
34:
35: int main()
36: {
37:       std::cout << "Making a cat...\n";
38:       SimpleCat Frisky;
39:       std::cout << "Frisky is " << Frisky.GetAge()
40:       << " years old\n";
41:
42:       int age = 5;
43:       Frisky.SetAge(age);
44:       std::cout << "Frisky is " << Frisky.GetAge()
45:       << " years old\n";
46:
47:       std::cout << "Calling FunctionTwo...\n";
48:       FunctionTwo(Frisky);
49:       std::cout << "Frisky is " << Frisky.GetAge()
50:       << " years old\n";
51:       return 0;
52: }
53:
54: // functionTwo passes a ref to a const object
55: const SimpleCat & FunctionTwo (const SimpleCat & theCat)
56: {
57:       std::cout << "Function Two. Returning...\n";
58:       std::cout << "Frisky is now " << theCat.GetAge()
59:       << " years old \n";
60:       // theCat.SetAge(8); const!
61:       return theCat;
62: }
```

4. Don’t Return a Reference to an Object

ReturnRef.cpp

```
1: #include <iostream>
2:
3: class SimpleCat
4: {
5:         public:
6:                   SimpleCat(int age, int weight);
7:                   ~SimpleCat() {}
8:                   int GetAge() { return itsAge; }
9:                   int GetWeight() { return itsWeight; }
10:        private:
11:                  int itsAge;
12:                  int itsWeight;
13: };
14:
15: SimpleCat::SimpleCat(int age, int weight):
16: itsAge(age), itsWeight(weight) {}
17:
18: SimpleCat &TheFunction();
19:
20: int main()
21: {
22:        SimpleCat &rCat = TheFunction();
23:        int age = rCat.GetAge();
24:        std::cout << “rCat is “ << age << “ years old!\n”;
25:        return 0;
26: }
27:
28: SimpleCat &TheFunction()
29: {
30:        SimpleCat Frisky(5,9);
31:        return Frisky;
32: }
```

5. Returning a Reference to an Object on the Heap

Leak.cpp

```
1: #include <iostream>
2:
3: class SimpleCat
4: {
5:        public:
6:                  SimpleCat (int age, int weight);
7:                  ~SimpleCat() {}
8:                  int GetAge() { return itsAge; }
9:                  int GetWeight() { return itsWeight; }
10:
11:       private:
12:                 int itsAge;
13:                 int itsWeight;
14: };
15:
16: SimpleCat::SimpleCat(int age, int weight):
17: itsAge(age), itsWeight(weight) {}
18:
19: SimpleCat & TheFunction();
20:
```

...

```
21: int main()
22: {
23:       SimpleCat &rCat = TheFunction();
24:       int age = rCat.GetAge();
25:       std::cout << "rCat is " << age << " years old!\n";
26:       std::cout << "&rCat: " << &rCat << "\n";
27:       // How do you get rid of that memory?
28:       SimpleCat *pCat = &rCat;
29:       delete pCat;
30:       // Uh oh, rCat now refers to ??
31:       return 0;
32: }
33:
34: SimpleCat &TheFunction()
35: {
36:       SimpleCat *pFrisky = new SimpleCat(5,9);
37:       std::cout << "pFrisky: " << pFrisky << "\n";
38:       return *pFrisky;
39: }
```

Tugas

- Modifikasi program Rectangle.cpp untuk membuat method drawnShape() lainnya dengan 2 buah parameter integer yang nilainya default.
- Modifikasi program DeepCopy untuk merubah kecepatan dallas setelah kecepatan wichita berubah. Apakah perubahan dallas berdampak pada wichita?