Memory Management 2 : References

Pertemuan 7

Outline

- Memory Management 2 : References
  - Creating References
  - Using the Address of Operator on References
  - References Cannot be Reassigned
  - Passing Function Arguments by Reference
  - Making **swap()** Work with Pointers
  - Implementing **swap()** with References
  - Returning Multiple Values
  - Returning Values by Reference

Membuat *Reference*

- Reference adalah sebuah "alias", ketika Anda membuat reference, Anda artinya menginisialisasi reference tersebut dengan nama objek lainnya sebagai target. Dengan demikian, reference bertindak sebagai nama alternatif dan apa pun yang Anda lakukan pada reference tersebut benar-benar dilakukan untuk objek target. Reference dapat menggunakan nama variabel apapun, jika Anda memiliki sebuah variabel integer bernama **someInt**, Anda dapat membuat referensi ke variabel yang dibaca sebagai "**rSomeRef"** adalah reference ke integer yang diinisialisasi untuk merujuk **someInt**.

```
int & rSomeRef = someInt;
```

Alamat untuk Operator References

- Jika Anda meminta reference untuk alamatnya, ia akan mengembalikan alamat dari target. Itulah sifat reference. Misalnya, jika Anda memiliki class **President**, kemudian Anda mendeklarasikan sebuah instance dari kelas tersebut dan menyatakan referensi untuk **President** dan menginisialisasi dengan cara sebagai berikut :

```
President Barack_Obama;
President &Obama = Barack_Obama; // declare a reference
```

Apa saja yang dapat di-*Reference*

- Setiap objek dapat dirujuk (di-*reference*), termasuk objek yang ditetapkan pengguna. Perhatikan bahwa Anda membuat reference ke sebuah obyek, bukan ke class atau tipe data seperti int. Anda tidak menulis ini:

```
int &rIntRef = int; // wrong
```

- Anda harus menginisialisasi rIntRef untuk bilangan bulat tertentu, seperti ini:

```
int howBig = 200;
int &rIntRef = howBig;
```

- Dengan cara yang sama, Anda tidak dapat melakukan inisialisasi reference ke class Cat:

```
Cat &rCatRef = Cat; // wrong
```

- Anda harus menginisialisasi rCatRef ke objek Cat tertentu:

```
Cat Frisky;
Cat &rCatRef = Frisky;
```

- Referensi ke obyek yang digunakan seperti obyek itu sendiri. Data anggota dan fungsi dapat diakses menggunakan akses operator anggota kelas (.), tindakan referensi ini ebagai alias untuk objek.

1. Creating a Reference

Reference.cpp

```
1: #include <iostream>
2:
3: int main()
4: {
5:         int intOne;
6:         int &rSomeRef = intOne;
7:
8:         intOne = 5;
9:         std::cout << "intOne: " << intOne << "\n";
10:        std::cout << "rSomeRef: " << rSomeRef << "\n";
11:
12:        rSomeRef = 7;
13:        std::cout << "intOne: " << intOne << "\n";
14:        std::cout << "rSomeRef: " << rSomeRef << "\n";
15:        return 0;
16: }
```

2. Using the Address of Operator on References

Reference2.cpp

```
1: #include <iostream>
2:
3: int main()
4: {
5:         int intOne;
6:         int &rSomeRef = intOne;
7:
8:         intOne = 5;
9:         std::cout << "intOne: " << intOne << "\n";
10:        std::cout << "rSomeRef: " << rSomeRef << "\n";
11:
12:        std::cout << "&intOne: " << &intOne << "\n";
13:        std::cout << "&rSomeRef: " << &rSomeRef << "\n";
14:
15:        return 0;
16: }
```

3. References Cannot be Reassigned

Assignment.cpp

```
1: #include <iostream>
2:
3: int main()
4: {
5:         int intOne;
6:         int &rSomeRef = intOne;
7:
8:         intOne = 5;
9:         std::cout << "intOne:\t" << intOne << "\n";
10:        std::cout << "rSomeRef:\t" << rSomeRef << "\n";
11:        std::cout << "&intOne:\t" << &intOne << "\n";
12:        std::cout << "&rSomeRef:\t" << &rSomeRef << "\n";
13:
14:        int intTwo = 8;
15:        rSomeRef = intTwo; // not what you think!
16:        std::cout << "\nintOne:\t" << intOne << "\n";
17:        std::cout << "intTwo:\t" << intTwo << "\n";
18:        std::cout << "rSomeRef:\t" << rSomeRef << "\n";
19:        std::cout << "&intOne:\t" << &intOne << "\n";
20:        std::cout << "&intTwo:\t" << &intTwo << "\n";
21:        std::cout << "&rSomeRef:\t" << &rSomeRef << "\n";
22:        return 0;
23: }
```

4. Passing Function Arguments by Reference

ValuePasser.cpp

```
1: #include <iostream>
2:
3: void swap(int x, int y);
4:
5: int main()
6: {
7:         int x = 5, y = 10;
8:
9:         std::cout << "Main. Before swap, x: " << x
10:        << " y: " << y << "\n";
11:        swap(x,y);
12:        std::cout << "Main. After swap, x: " << x
13:        << " y: " << y << "\n";
14:        return 0;
15: }
16:
17: void swap (int x, int y)
18: {
19:        int temp;
20:
21:        std::cout << "Swap. Before swap, x: " << x
22:        << " y: " << y << "\n";
23:
24:        temp = x;
25:        x = y;
26:        y = temp;
27:
28:        std::cout << "Swap. After swap, x: " << x
29:        << " y: " << y << "\n";
30:
31: }
```

5. Making **swap()** Work with Pointers

PointerSwap.cpp

```
1: #include <iostream>
2:
3: void swap(int *x, int *y);
4:
5: int main()
6: {
7:         int x = 5, y = 10;
8:
9:         std::cout << "Main. Before swap, x: " << x
10:        << " y: " << y << "\n";
11:        swap(&x, &y);
12:        std::cout << "Main. After swap, x: " << x
13:        << " y: " << y << "\n";
14:        return 0;
15: }
16:
17: void swap(int *px, int *py)
18: {
19:        int temp;
20:
21:        std::cout << "Swap. Before swap, *px: " << *px
22:        << " *py: " << *py << "\n";
23:
24:        temp = *px;
25:        *px = *py;
26:        *py = temp;
27:
28:        std::cout << "Swap. After swap, *px: " << *px
29:        << " *py: " << *py << "\n";
30: }
```

6. Implementing **swap()** with References

ReferenceSwap.cpp

```
1: #include <iostream>
2:
3: void swap(int &x, int &y);
4:
5: int main()
6: {
7:         int x = 5, y = 10;
8:
9:         std::cout << "Main. Before swap, x: " << x
10:        << " y: " << y << "\n";
11:        swap(x, y);
12:        std::cout << "Main. After swap, x: " << x
13:        << " y: " << y << "\n";
14:        return 0;
15: }
16:
17: void swap(int &rx, int &ry)
18: {
19:        int temp;
20:
21:        std::cout << "Swap. Before swap, rx: " << rx
22:        << " ry: " << ry << "\n";
23:
24:        temp = rx;
25:        rx = ry;
26:        ry = temp;
27:
28:        std::cout << "Swap. After swap, rx: " << rx
29:        << " ry: " << ry << "\n";
30:}
```

7. Returning Multiple Values

ReturnPointer.cpp

```
1: #include <iostream>
2:
3: short factor(int, int*, int*);
4:
5: int main()
6: {
7:         int number, squared, cubed;
8:         short error;
9:
10:        std::cout << "Enter a number (0 - 20): ";
11:        std::cin >> number;
12:
13:        error = factor(number, &squared, &cubed);
14:
15:        if (!error)
16:        {
17:                  std::cout << "number: " << number << "\n";
18:                  std::cout << "square: " << squared << "\n";
19:                  std::cout << "cubed: " << cubed << "\n";
20:        }
21:        else
22:                  std::cout << "Error encountered!!\n";
23:        return 0;
24: }
25:
```

...

```
26: short factor(int n, int *pSquared, int *pCubed)
27: {
28:       short value = 0;
29:       if (n > 20)
30:       {
31:                 value = 1;
32:       }
33:       else
34:       {
35:                 *pSquared = n*n;
36:                 *pCubed = n*n*n;
37:                 value = 0;
38:       }
39:       return value;
40: }
```

8. Returning Values by Reference

ReturnReference.cpp

```
1: #include <iostream>
2:
3: enum ERR_CODE { SUCCESS, ERROR };
4:
5: ERR_CODE factor(int, int&, int&);
6:
7: int main()
8: {
9:         int number, squared, cubed;
10:        ERR_CODE result;
11:
12:        std::cout << "Enter a number (0 - 20): ";
13:        std::cin >> number;
14:
15:        result = factor(number, squared, cubed);
16:
17:        if (result == SUCCESS)
18:        {
19:                  std::cout << "number: " << number << "\n";
20:                  std::cout << "square: " << squared << "\n";
21:                  std::cout << "cubed: " << cubed << "\n";
22:        }
23:        else
24:        {
25:                  std::cout << "Error encountered!!\n";
26:        }
27:        return 0;
28: }
29:
```

...

```
30: ERR_CODE factor(int n, int &rSquared, int &rCubed)
31: {
32:       if (n > 20)
33:       {
34:                 return ERROR; // simple error code
35:       }
36:       else
37:       {
38:                 rSquared = n*n;
39:                 rCubed = n*n*n;
40:                 return SUCCESS;
41:       }
42: }
```

Tugas

- Modifikasi program ReturnPointer.cpp menggunakan reference dibandingkan sebelumnya menggunakan pointer.
- Modifikasi program ReferenceSwap.cpp agar dapat melakukan swap 3 angka.
- Modifikasi program Leak.cpp menggunakan pointer yang memanggil TheFuction() dan gunakan metode "proper deletion" untuk mencegah kekurangan memory.
- Modifikasi program ObjectRef.cpp dan RefPasser.cpp untuk menampilkan alamat dari variabel sebelum fungsi dipanggil dan setelah fungsi dipanggil. Hal ini akan memberikan penjelasan bagaimana mekanisme tersebut bekerja.