

Reference



C++ Object Oriented Programming

Pei-yih Ting
96/02 NTOU CS

..... 1

Contents

- ❖ What is reference in C++?
- ❖ Concept of an alias
- ❖ Initialization of a reference
- ❖ Reference can replace a pointer but is not a pointer
- ❖ Function that can be used as an l-value
- ❖ Reference can be used to increase efficiency
- ❖ Reference as a member variable
- ❖ Reference in copy constructor X(X&)

2

References

- ❖ C simulates “call by reference” through pointers

```
void func(int *ptrData) {  
    *ptrData = 10;  
}
```

```
void main() {  
    int data;  
    ...  
    func(&data);  
    ...  
}
```

- ❖ C++ has true references

```
void func(int &param) {  
    param = 10;  
}
```

```
void main() {  
    int data;  
    ...  
    func(data);  
    ...  
}
```

no dereference operator required

no address operator required

It is also the goal of C++ to reduce the usage of pointers.

- ❖ Some C++ programmers might do the following for saving time and memory of argument passing

```
static void Foo(const CBigData &data) {  
    ...  
}
```

3

References (cont'd)

- ❖ There are no promotions or type conversions with references

```
void func(double &data) {  
    data = 10;  
}
```

```
void main() {  
    int data;  
    ...  
    func(data);  
    ...  
}
```

error C2664: 'func' : cannot convert parameter 1 from 'int' to 'double &'

- ❖ A reference variable can not be bound to a temporary object

```
int getValue() {  
    int tmp;  
    return tmp;  
}  
int func(int &value);  
void main() {  
    func(getValue());  
}
```

error C2664: 'func' : cannot convert parameter 1 from 'int' to 'int &'

4

References as Aliases

- ❖ A reference can function as an **alias to another variable**.

```
void main {
    int x = 5;
    int &alias = x;
    cout << "The value of x is " << x << endl;
    cout << "The value of the alias is " << alias << endl;
    alias = 10;
    cout << "The value of x is " << x << endl;
    cout << "The value of the alias is " << alias << endl;
}
```

- ❖ Like a constant variable, the reference must be initialized in its declaration.

```
int x = 5;
int &alias;
alias = x;
```

Error: 'const' or '&' variable needs initializer

Note: Initialization and assignment are very different.

References are not Pointers

- ❖ Cannot be reassigned

```
int &alias = x;
int &alias = y;
```

Error: identifier 'alias' redeclared.

- ❖ Not related to concept of memory addresses any more

<pre>int x = 5; int y = 5; int &aliasX = x; int &aliasY = y; if (aliasX == aliasY) cout << "identical.\n"; else cout << "different\n";</pre>	<pre>int x = 5; int y = 5; int *ptrX = &x; int *ptrY = &y; if (ptrX == ptrY) cout << "identical.\n"; else cout << "different\n";</pre>
--	--

Output: identical

Output: different

References are not Pointers (cont'd)

- ❖ You cannot obtain the address of a reference

```
int x = 5;
int *ptr;
int &alias = x;
ptr = &alias;
```

There are only two variables in this code segment. ptr contains the address of x (not the address of alias, And indeed alias is not itself a variable)

- ❖ No similar thing as pointer arithmetic

```
int array[] = {3, 2, 1};
int &alias = array[0];
alias++;
```

Output: 4
4

- ❖ Can you alias a pointer variable? Yes

```
void main() {
    char *string = "hello";
    Foo(string);
    cout << string;
}

void Foo(char* &strPtrRef) {
    strPtrRef = "good day";
}
```

Output: good day

Function Returning a Reference

- ❖ Assuming that you want to emulate a Pascal-style 1-based array:

```
int &pArray(int cArray[], int index) {
    return cArray[index-1];
}

void main() {
    int array[] = {1, 2, 3};
    cout << pArray(array, 2) << '\n';
    pArray(array, 1) = 10;
    cout << pArray(array, 1) << '\n';
}
```

Output: 2
10

- ❖ You can use the function as an l-value.

Returning a Reference (cont'd)

❖ Why is the following code not working?

```
int &pArray(int index) {
    int cArray[] = {1, 2, 3};
    return cArray[index-1];
}
void main() {
    cout << pArray(2) << '\n';
    pArray(1) = 10;
    cout << pArray(1) << '\n';
}
```

Output:	
	2
	2

9

Reference Saves Computation

❖ Like the usage of pointer, reference used for function arguments can save computation time in copying data.

```
BigDataT x, y;
...
Foo(x, y)
...

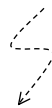
void Foo(const BigDataT &inputData, BigDataT &outputData)
{
    ...
    inputData.accessor(); // access the aliased variable by inputData, i.e. x, directly
    ... // without changing it
    outputData.mutator(); // access y directly and modify its value
    ...
}
```

10

References as Data Members

```
double gCustomerCreditLimit = 1000;
...
class Patron {
public:
    Patron(double &limit);
    void Charge(double amount);
private:
    const double &fCreditLimit;
};
...
Patron::Patron(double &limit): fCreditLimit(limit) {
}
...
Patron patron(gCustomerCreditLimit);
...
```

The only way to initialize
a reference member variable



11

The Hidden Perils of C++

```
class String {
public:
    String();
    String(const char *inputStr);
    ~String();
    const char *GetString() const;
private:
    char *fString;
};

String::String(char *inputStr) {
    fString = new char[strlen(inputStr)+1];
    strcpy(fString, inputStr);
}

String::~~String() {
    delete[] fString;
}

void main() {
    String string1("Hello");
    String string2 = string1;

    cout << string1.GetString() << endl;
    cout << string2.GetString() << endl;
}

This piece of code often make your  
program crash. The lack of explicit copy  
constructor creates two pointers for the  
same piece of memory.
```

12

Copy Ctor X(X&)

✧ Definition of a copy constructor

```
String(String &src) {  
    fString = new char[strlen(src.fString)+1];  
    strcpy(fString, src.fString);  
}
```

✧ It is necessary that the copy constructor use reference as parameter. Without reference parameter, it would be a recursive definition.

✧ Usage of a copy constructor

1. String string2 = string1;
2. String string2(string1);
3. Calling a function fun(string1);
void fun(String stringParam) {
 ...
}