Comp151

Reference Variables

Creating a Reference Variable

A reference is an alternative name (alias) for an object. Reference variables are usually used in parameter passing to functions. Before looking at how to use them as parameters we first examine their stand-alone properties.

```
int j = 1;
int& r = j;       // Now r = j = 1
int x = r;        // Now x = 1
r = 2;            // Now r = j = 2
```

The notation "`int& r = j;`" means that r is a reference variable that is another name for j.

A reference allows indirect manipulation of an object, somewhat like a pointer, without requiring complicated pointer syntax.

A reference must always be *bound* to an object. It must therefore be initialized when it is created:

```
int j = 1;
int& r1 = j;          // Ok
int& r2;              // Error
```

Initialization & Assignment

Distinguish between initializing a reference and making an assignment to it:

```
int j = 10;
int& r = j;
++r;          // both r and j become 11
j += 8;       // both r and j become 19
r = 7;        // both r and j become 7

cout << j << "  ,  " << &j << endl;
cout << r << "  ,  " << &r << endl;
```

- A reference can not be "redirected"; It always refers to the object it initially was initialized to.
- Assignment only changes the "value" of its referenced object.
- Even the address of a reference is that of the referenced object!

The Different Uses Of &

Do not confuse the use of & in a reference assignment, e.g.,

```
int& i = j;
```

with the use of & as the *address of* operator, e.g.,

```
int j; int *pi = &j
```

The following is wrong. Why?

```
int j; int &i = &j
```

The following is correct. What does it mean?

```
int j; int *pi = &j; int *&ref = pi;
```

```
#include<iostream.h>
void main()
{
  int j=1;                  //j is an int
  int *pi = &j;             //pi is an int*
                            //initialized to address of j
  int *&ref = pi;           //ref is ref variable of type int*

  cout << "j = "   << j;
  cout << "*pi = " << *pi << " *ref =" << *ref << endl;

  int k =2;
  pi = &k;
  cout << "j = "   << j;
  cout << "*pi = " << *pi << " *ref =" << *ref << endl;
}
```

Call-By-Reference and Reference Arguments

Reference arguments are a special case of references:

int f(**int**& i) { ++i; **return** i; }

int main() { **int** j = 7; cout ≪ f(j) ≪ endl; cout ≪ j ≪ endl; }

- Variable `i` is a local variable in the function `f`.
- Its type is `int` reference and is created when `f` is called.
- In the call `f(j)`, `i` is created similarly to the construction:

```
int& i = j;
```

- So within the function `f`, `i` will be an alias of the variable `j`, and that cannot be changed while the variable `i` exists.
- But everytime the function `f` is called, a new variable `i` is created and it can be a reference to a different object.

Call-By-Reference and Call-By-Value

In C++, an argument to a function may be passed by 2 methods:

- call-by-reference (CBR)
- call-by-value (CBV)

- In the call `f(j)`, `i` is created similarly to the construction:

CBR	CBV
`int& i = j;`	`int i = j;`

- Note that in CBV, `i` is NOT an alias of `j`. Thus, any change to `i` will not result in any change to `j`.

2 Reason for Call-By-Reference?

1. When function caller wants the function to be able to change the value of passed arguments.
2. If you pass a function argument by value, the function gets a local copy of the argument. For large objects, copying is expensive; on the other hand, passing an object by reference does not require copying, only passing a memory address.

```
class Large_Obj
{
  public:
    // Lots of data members
    int height;
};

void print_height(const Large_Obj& LO){ cout << LO.height(); }

int main() { Large_Obj dinosaur(50); print_height(dinosaur); }
```

Pointer vs. Reference

Reference can be thought as a special kind of pointer, but there are 3 big differences to remember!

- A pointer can point to *nothing* (`NULL`), but a reference is always bound to an object.
- A pointer can point to *different* objects at different times (through assignments). A reference is always bound to the *same* object. Assignments to a reference do NOT change the object it refers to but only the value of the referenced object.
- The name of a pointer refers to the pointer object. The * or -> operators have to be used to access the object.
  The name of a reference always refers to the object. There are no special operators.

Example: Pointer vs. Reference


```
#include<iostream.h>
void func1(int *pi) {(*pi)++;}
void func2(int &ri) {ri++;}

void main()
{ int i=1; cout << "i = " << i << endl;

 // call using address of i
 func1(&i); cout << "i = " << i << endl;

 // call using i
 func2(i); cout << "i = " << i << endl;
}
```

Example: More Pointer vs. Reference

```
IQ w("Maxwell", 180);
IQ x("Newton", 200);
IQ* a = 0;                                   // Ok: 'a' bound to nothing

IQ& y = x;                                   // Ok: 'y' is an alias of 'x'
IQ& z;                                       // Error: uninitialized ref var!

a = new IQ("Einstein", 250);                 // Ok: 'a' points to "Einstein"
a = new IQ("Galileo", 190);                  // Ok: 'a' now points to "Galileo"
y = w;                                       // 'y' is STILL an alias of 'x' NOT 'w';
                        // the value of 'w' is copied to 'y' ('x')

a->smarter(10); (*a).print();
y.smarter(20); y.print();
```