An Introduction to Programming

through C++

with

Manoj Prabhakaran

Lecture 8

Functions

References

Based on material developed by Prof. Abhiram G. Ranade

Functions: So far

- Declaring (just specifying the *signature*) and defining a function
- Calling a function: how the stack works
- int main() (and implicit return 0)
- Functions returning void (and implicit return)

Today

- Passing arguments by *reference*
- Returning by reference
- Example use cases

Reference: Chapter 9

Example: Swapping

```
//  Swapping logic:
//  int tmp; tmp=x; x=y; y=tmp;
```

```
int main() {
  int x, xp, y, yp, deg, degp;
  ...
  int tmp;
  tmp=x; x=xp; xp=tmp; //swap x, xp
  tmp=y; y=yp; yp=tmp; //swap y, yp
  tmp=deg; x=degp; xp=tmp; //swap deg, degp
}
```

Repeated thrice.
Encapsulate in a function?

Example: Swapping (Attempt 1)

```
void swp1(int x, int y) {
  int tmp; tmp=x; x=y; y=tmp;
}
```

Only the local variables are swapped. They are destroyed when swp1 returns.

```
int main() {
  int x, xp, y, yp, deg, degp;
  ...
  swp1(x,xp);
  swp1(y,yp);
  swp1(deg,degp);
}
```

Example: Swapping using References

```
void swp(int& x, int& y) {
  int tmp; tmp=x; x=y; y=tmp;
}
```

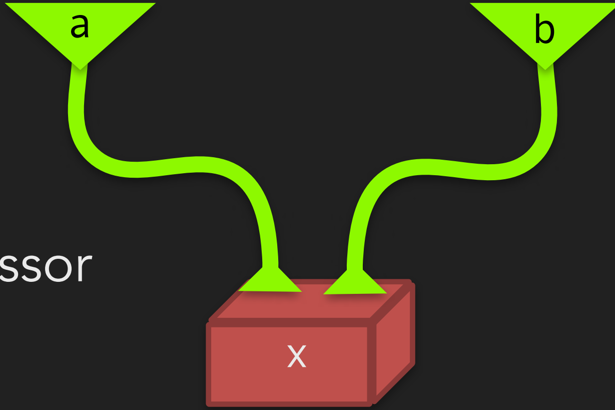
Now x, y are references to the arguments

```
int main() {
  int x, xp, y, yp, deg, degp;
  ...
  swp(x,xp);
  swp(y,yp);
  swp(deg,degp);
}
```

Works as desired!

References

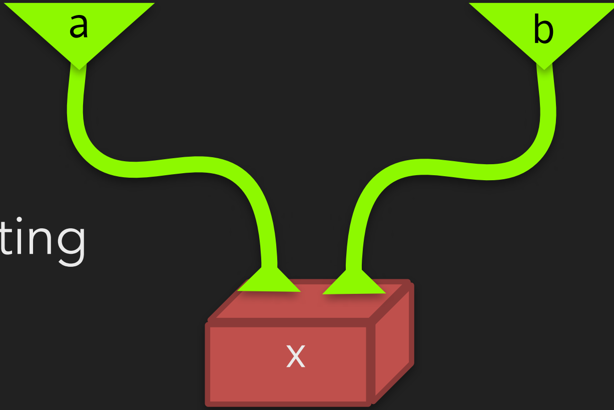
- A variable has a space in memory that your program can refer to
  - Think of a box containing a slate to write on
  - The program can point to a box and ask the processor to read from/write to that box. (e.g. `x = x + 1`)
- References: Tubes that are attached to boxes!
- You can point to these tubes too , and read from/write to the tubes. (e.g. `a = b + 1`)
  - Uses/affects the value in the box!
    (e.g,. here `a = b + 1` same as `x = x + 1`).



```
int x;
int& a = x;
int& b = x;
a = 10;
x++;
b == 11; // true
```

References

- Intermediate values can't be referred to
  - E.g., `int& a = x+1;   // won't compile`
  - Imagine that the processor uses an internal whiteboard for intermediate calculations. Not sitting in any box in the memory.
- Just like intermediate values can't be assigned to
  - E.g., `x+1 = 3;   // won't compile`
- Expressions which have boxes are called *lvalues* ("left" values which can appear on the left hand side of an assignment operation; for now just variables and references)
- Expressions without boxes are called *rvalues*



```
int x;

int& a = x;
int& b = x;

a = 10;

x++;

b == 11; // true
```

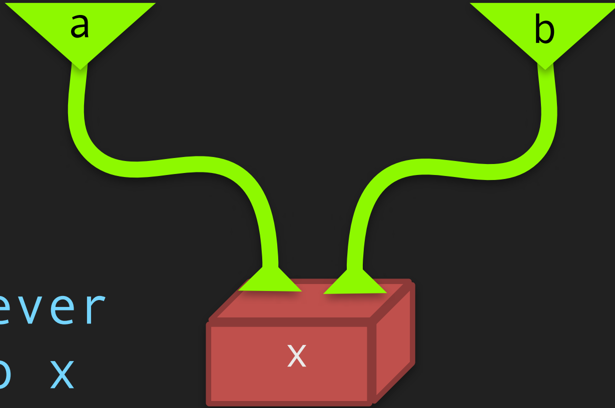
References

- A reference needs to be attached to a box when it is declared, and cannot be reattached later.
- Example:

```
int x, y;
int& a = x; // a is attached to x forever
a = y;      // will copy value of y to x
```

- While declaring a reference, instead of specifying a box to attach to, can specify a reference to it.
  For example:

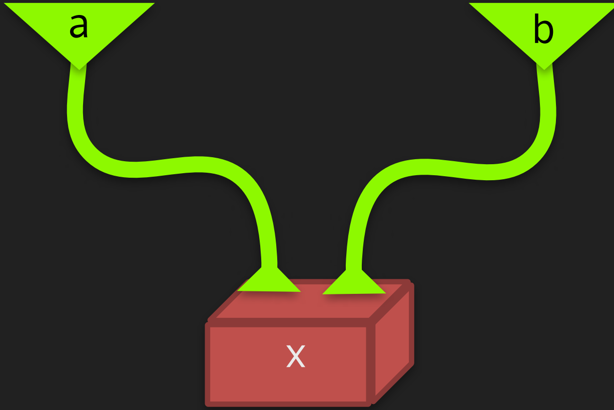
```
int x; int& a = x;
int& b = a; // b and a are attached to x
```



```
int x;

int& a = x;
int& b = x;

a = 10;

x++;

b == 11; // true
```

References

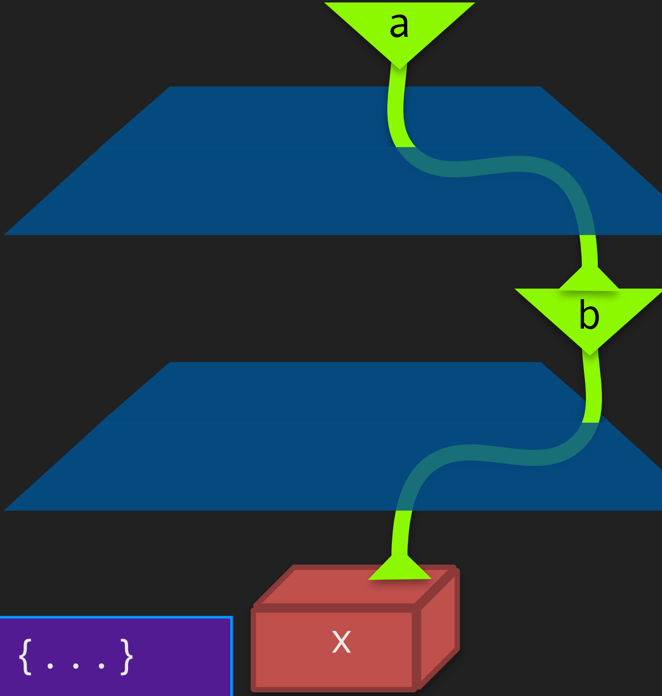
- When declaring multiple reference variables, each one should be marked using the & sign
  - E.g., `int & a = x, & b = y;`
- It is possible to mix reference and non-reference variables in a declaration
  - E.g., `int x, & a = x;`
- Spaces around & are optional
  - `int&x;` , `int& x;` and `int &x;` all mean the same



```
int x;

int& a = x;
int& b = x;

a = 10;

x++;

b == 11; // true
```

Passing Arguments by Reference

- If a function's parameter is a reference (a tube), it will be attached to a memory location (a box), when the function is called
- The box is (typically) in the frame of the calling function
  - Note: the called function gets access to variables not in its frame!
  - The box can be further down in the stack too!



```
void h(int &a) {...}
```

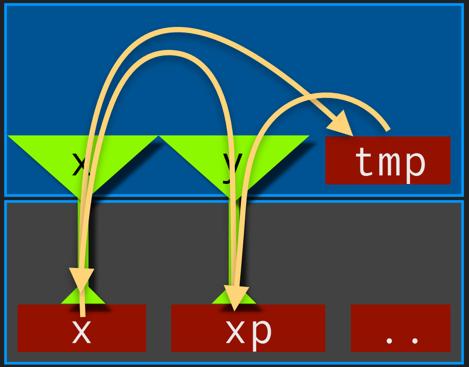
```
void f() {
   int x; ... g(x); ...
}
```

```
void g(int &b) {
   ... h(b); ...
}
```

Example: Swapping

```
void swp(int& x, int& y) {
  int tmp; tmp=x; x=y; y=tmp;
}
```

```
int main() {
  int x, xp, y, yp, deg, degp;
  ...
  swp(x,xp); swp(y,yp); swp(deg,degp);
}
```



Example: Incrementing

```
int postinc(int& x) { // implements x++
    int oldx = x;
    x = x+1;
    return oldx;
}
```

- Note that without the reference, the function will become equivalent to just the identity function (`return x;`)

Returning Many Values

- Passing by reference can be used as a way to let a function "return" many values
- Can pass "placeholders" for as many return values as are desired; the function will populate them before returning
- E.g., it is more efficient to compute sine and cosine together.

```
void SinCos(double theta, double& sin, double& cos) {..}

int main() {
  double x, sinx, cosx; cin >> x;
  SinCos(x,sinx,cosx);
  cout << "sin & cos: " << sinx << ", " << cosx << endl;
}
```

Returning Many Values

Demo

- Passing by reference can be used as a way to let a function "return" many values

```
int main() {
  ...
  string msg;
  bool valid;
  valid = isValid(...,msg);
  if (valid) {
    ...
  } else
    cerr << msg << endl;
  ...
}
```

```
bool isValid(int x, int y, int xp, int yp,
             int dx, int dy,
             int limit, string& msg) {
    if (abs(x+dx) >= limit || abs(y+dy) >= limit) {
        msg = "Can't hit the box!"; return false;
    }
    if (x+dx==xp && y+dy==yp) {
        msg = "Can't collide!"; return false;
    }
    msg = ""; return true;
}
```

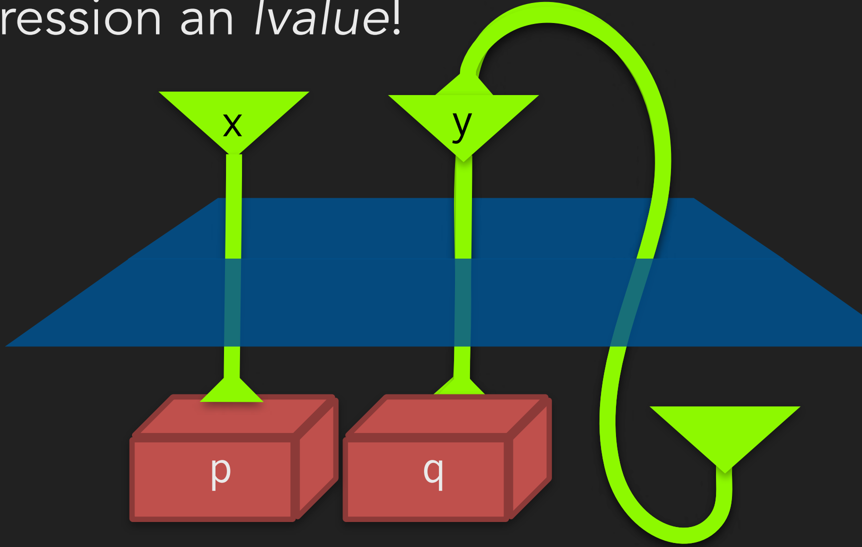
Returning a Reference

- A function can be declared to return a reference too!
- This makes the function evaluation expression an *lvalue*!

```
int& maximum(int& x, int& y) {
  if(x>=y) return x; else return y;
}
```

```
int main() {
  int p, q;
  cin >> p >> q;
  maximum(p,q) = 0;
}
```

Valid because LHS is an lvalue: a reference to a "box"



Returning a Reference

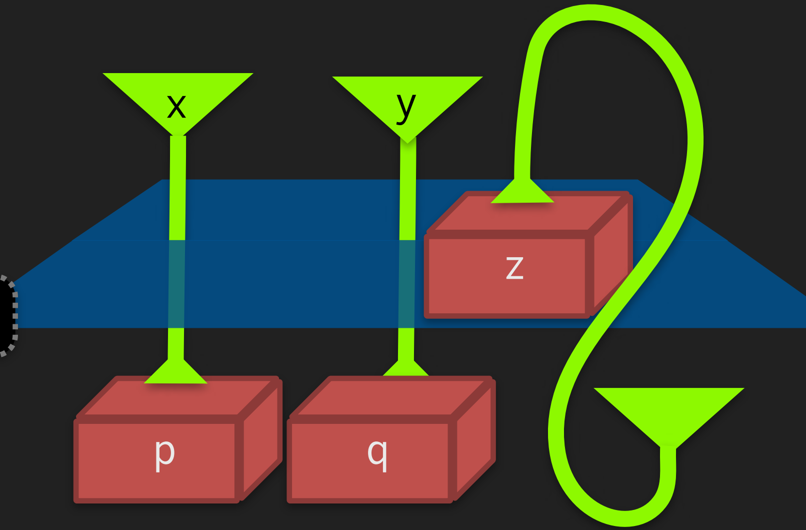
- Be careful not to return a reference to a local variable
  - Compiler can try to warn you

```
int& badMaximum(int& x, int& y) {
  int z = (x>=y)? x: y;
  return z;
}
```

Box for z will be destroyed when the function returns!

```
int main() {
  int p, q;
  cin >> p >> q;
  int& r = badMaximum(p,q);
}
```

Initialising a reference: valid because RHS is an lvalue (box or a tube)



Example: Incrementing

```
int& preinc(int& x) { // implements ++x
    x = x+1;
    return x;
}
```

```
...
int x;
int& a = preinc(x); // a attached to x now
preinc(preinc(x));
// But the following are illegal since postinc returns by value
// int& b = postinc(x);
// postinc(postinc(x));
```

Const Reference

- When passing "big" data, like strings, it can be more efficient to pass by reference, because copying data around memory can slow things down
    - E.g., `int getInput(string& prompt)`
    - But risky: without checking the internals of the function, can't tell if it modifies the argument
- If a function's parameter is a reference, its argument needs an lvalue
    - E.g., `bool isEmpty(string& s) { return s==""; }`
      `isEmpty("hello"); // will give compiler error`
- Using a const reference parameter (`const string&`) solves both these
    - For the second issue: const references can be initialised with rvalues
- Use a const reference instead of a reference whenever possible

Exercise

- Inspect the sample programs ref.cpp and increment.cpp accompanying this lecture.

  Understand why it works the way it works.

- Study the sample program 2turt-fun.cpp accompanying this lecture.

  Add more features and/or reorganise the code to use more functions.