An Introduction to Programming though C++

Abhiram G. Ranade

Lecture 5.1

Ch. 9: Functions

Some shortcomings

Using what we just learned, it is not possible to write functions to do the following:

- A function that exchanges the values of two variables.
- A function that produces several values as results:
  - Function to produce polar coordinates given Cartesian coordinates.

Exchanging the values of two variables, attempt 1

```
void exchange(int m, int n){
    int temp = m;
    m = n; n = temp;
    return;
}
main_program{
    int a=1, b=2;
    exchange(a,b);
    cout << a <<‘ ‘<<
        b << endl;
}
```

- Does not work. 1, 2 will get printed.
- When exchange is called, 1, 2 are placed into m, n.
- Execution of exchange exchanges values of m,n.
- Change in m,n does not affect the values of a,b of main_program.

Reference parameters

```
void exchange(int &m, int &n){
    int temp = m;
    m = n; n = temp;
    return;
}
main_program{
    int a=1, b=2;
    exchange(a,b);
    cout << a <<‘ ‘<<
        b << endl;
}
```

- & before the name of the parameter:
- Says: “Do not allocate space for this parameter, but instead just use the variable from the calling program.”
- When function changes m,n it is really changing a,b.
- Such parameters are called reference parameters.
- 2 1 will be printed.

Remark

- If a certain parameter is a reference parameter, then the corresponding argument is said to be “passed by reference”.
- Now we can write a program that computes polar coordinates given cartesian coordinates
  - We use two reference parameters.
  - Called function stores the polar coordinates in the reference parameters.
  - These changes can be seen my the main program.
- There are other ways of returning 2 values – study later.

Cartesian to polar

```
void CtoP(double x, double y,
          double &r, double &theta){
     r = sqrt(x*x + y*y);
     theta = atan2(y, x); //arctan
return;
}
main_program{
     double x=1, y=1, r, theta;
     CtoP(x,y,r,theta);
     cout << r <<‘ ‘<< theta << endl;
}
```

- r, theta in CtoP are reference parameters,
- changing them in CtoP changes the value of r, theta in the main program.
- Hence sqrt(2) and pi/4 (45 degrees) will be printed.

Exercises

Write a function which takes a length in inches and returns the length in yards, feet, and inches. Note that 12 inches make a foot, and 3 feet make a yard. As an example: 100 inches = 3 yards, 2 feet, 4 inches.

Hint: your function will have 4 parameters, one in which you will pass the given length, and the other 3 in which the function will put the the number of yards, number of feet and number of inches.

What we discussed

- If we want to return more than one result we can do so by using a reference parameter.
- If we use a reference parameter R in a function, and pass as argument a variable A, then any change that the function makes in R will be seen by the calling program as a change in A.
- Next: Pointers, which perform a similar function.

Pointers

- If the memory of a computer has N bytes, then the bytes are numbered 0..N-1.
- The number of a byte (different from what is stored in the byte) is said to be its address.
- A pointer is a variable that can store addresses.
  - Sometimes “pointer” might mean address.
- What we accomplished using reference variables can also be accomplished using pointers.
  - This will be seen soon.
- Pointers will also be useful elsewhere.

How to find the address of a variable

- The operator & can be used to get the address of a variable.
  - The same & is used to mark reference parameters; but the meaning will be clear from the context.

```
int t;
cout << &t << endl;
```

- This prints the address of the variable t, i.e. the address of the first byte that comprises the variable t.
- Customarily, addresses get printed in hexadecimal radix, i.e. they will consist of a sequence of hexadecimal digits prefixed by “0x”
- Note: hexadecimal digits: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F.

Variables that can store addresses

- How to create a variable for storing addresses of variables of type int :

```
int *v;   // read as “int star v”
```

- The * is not to be read as multiplication.
  - Think of it as (int*) v; where int* means the type: “address of int”.

```
int p;
v = &p;   // address of p stored in v
```

- Since p is of type int, &p has type address of int.
- Thus, it is OK to store &p in v, which is also of type address of int.

```
cout << v <<‘ ‘<< &p << endl; // both print same
v = p; // compile time error: type mismatch
```

Pointers in general

- In general, to create a variable w to store addresses of variables of type T, write:

T* w;

- Assignment statements: types of lhs and rhs must be same
  - Except when both sides are numeric types; then conversion rules used.
  - No conversion rule between pointers of one type and pointers of other types.
  - No conversion rule between pointers of one type and values of any type.

The dereferencing operator *

- If v contains the address of p, then we can get to p by writing *v.

```
int *v;
int p;
v = &p;
*v = 10; // as good as p = 10.
```

- Think of * as the inverse of &.
- &p : the address of the variable p
- *v : the variable whose address is in v
- int *v;
  - v is such that *v is an int
  - v is an address of an int

Pointers in functions

```
void CtoP(double x, double y,
  double *pr, double *ptheta){
  *pr = sqrt(x*x + y*y);
  *ptheta = atan2(x,y);
  return;
}
main_program{
  double r, theta;
  CtoP(1,1,&r,&theta);
  cout << r <<‘ ‘
       << theta << endl;
}
```

- main_program calls CtoP, supplying &r, &theta as third and fourth arguments.
- This is acceptable because corresponding parameters have type double*.
- The first step of the call copies the addresses of r,theta of the main_program into pr, ptheta of CtoP.
- *pr means the variable whose address is in pr, in other words, the variable r of main_program.
- Thus CtoP changes the variables of main_program.
- Thus r becomes √2 = 1.41 and theta becomes π/4 = 0.79 and are printed.

Remarks

- In variable definitions, * associates to the right. Example:

int *v, p;

- This means int *v; and int p; i.e. defines a variable v of type int*, and variable p of type int.
- For now, assume that the only operations you can perform on a variable of type T* are
  - dereference it,
  - store into it a value &v where v is of type T,
  - store it into another variable of type T*
  - pass it to a function as an argument, provided corresponding parameter is of type T*

Exercise

- Point out the errors in this code.

```
int *p, *q, w;
p = w;
q = 3;
```

- What is the result of executing the following:

```
int *p, *q, w, x;
p = &w;
w = 10;
q = &x;
*q = 20;
cout << *p + x <<endl;
```

What we discussed

- A pointer is an address or a variable containing an address.
- A pointer to a variable can be created in the main program and dereferenced during a function call.
- This way a function can be allowed to modify variables in the main program, or other functions.
- Pointers can do the same thing as references, but the notation is clumsier.
- But pointers can do other things too. (Later)

Functions and graphics objects

- You can pass graphics objects to functions.
- The parameter must have the same type, i.e. shape.
- If you pass by value, a copy of the variable is made for use in the called function.
  - The copy is destroyed when the function returns.
- If you pass by reference of pass a pointer, the function can operate on the original graphics object.
- If you imprint an object in a call, the image will survive after the call finishes.

- Incidentally, you can make a copy of an object even using assignment

Rectangle r(100,100,80,20); Rectangle s=r;

```
void Rev360(Rectangle &r){
  repeat(36){
    r.right(10);
    r.imprint();
    wait(0.01);
  }
}


main_program{
  initCanvas();
  Rectangle r(100,100,80,20);
  Rev360(r);
  getClick();
}
```

Exercise

Write a function which takes a rectangle and coordinates x, y as input and decides whether the point (x,y) lies inside the rectangle.

You will need to know the following useful operations that can be performed on any rectangle R.

- R.getX() : returns the x coordinate of the rectangle center
- R.getY() : returns the y coordinate of the rectangle center
- R.getWidth() : returns the width of the rectangle
- R.getHeight() : returns the height of the rectangle

Concluding remarks

- If you find that you are performing the same operation at several places in your program, consider making it into a function.
- Function = “packaged software component”.
  - The user of the function does not need to worry what happens inside the function.
  - The user only expects the specification of the function to be honoured.
- Arguments can be passed by value:
  - If corresponding parameter is modified in function, no direct effect in calling program.
- Arguments can be passed by reference:
  - If corresponding parameter is modified in function, variable in calling program changes.
- Argument is a pointer to a variable in the calling function:
  - Code in called function can access variable by dereferencing pointer.

Exercises

- Write a function that draws an n sided regular polygon such that each side has length s, and returns the perimeter (n*s) as the result.
- Write a function that returns the cube root of a number using Newton's method. Have an additional parameter to the function for specifying the number of iterations you want performed.
- Other exercises at the end of Chapter 9.