

Why Functions?

A function is a block of code that has a name, a return type and a list of arguments. Every C++ program has at least one function: main. Execution of a program always starts with main. The function main usually makes calls to other functions. In this chapter you will learn to write your own functions, and call them. One use for functions is to break a program into separate functions to improve the readability of a program and to create functions that can be used in other programs. Each function should have a clearly defined purpose. Each function should also have a comment to explain its purpose.

Example: The program below has a function that receives the length and width of a rectangle and returns the area. This has limited usefulness, but it is short and easy to follow.

```
0 // Area program: using area function that returns nothing
1 #include <iostream.h>
2 void area(int,int); // receives length & width, prints area
3 main()
4 { int length, width;
5   cout<<"Enter length: ";
6   cin>>length;
7   cout<<"Enter width: ";
8   cin>>width;
9   area(length, width); // calls function area
10  return 0;
11 } // main
12
13 void area(int l, int w) // print the area given l and w
14 { int a;
15   a=l*w;
16   cout<<"The area is "<<a<<"\n";
17 }
```

Explanation:

2. The prototype must appear before the function can be called. The prototype shows the return type and the arguments. When the function is used, the compiler can determine if the function has been used correctly. The function **area** must receive two integers.
9. The function is called, passing length and width. Main waits until the function area finishes, then continues with the next statement.
13. The function area receives the values that were stored in the variables length and width in main. These values are stored in variables l and w in function area.
14. The variable **a** is local to area. The space is allocated when area starts, it is de-allocated when area ends.
- 15-16. The area is calculated and printed. The variable **a** is not necessary, one statement `cout<<"The area is "<<l*w<<"\n";` would be sufficient if we had no interest in discussing local variables.
- 17: When the function area ends, any space that was allocated is de-allocated and main resumes where it left off.

Example: The program below has a function to test whether a number is odd or even. This has limited usefulness, but it is short and easy to follow. This time the function returns a value:

```
0 // Inputs numbers and tells if odd or even
1 #include <iostream.h>
2 int even(int); // the prototype for function even
3 int main()
4 { int num;
5   cout<<"Enter a number:";
6   cin>>num;
7   if (even(num)) // call function even passing num
8     cout<<num<<" is even.\n";
9   else
10    cout<<num<<" is odd.\n";
11   return 0;
12 } // main
13
14 int even(int n)
15 // returns 1 if n is divisible by 2, 0 otherwise
16 { if (n%2==0) // even if n/2 has remainder of 0
17   return 1; // 1 is true
18   else
19   return 0; // 0 is false
20 } // even
```

Explanation:

- 2: The prototype must appear before the function can be called. The prototype shows the return type and the arguments. When the function is used, the compiler can determine if the function has been used correctly. The function even will receive one integer value and return an integer. The function even is referred to as an integer function.
- 3: Function main does not receive any arguments, it returns an integer.
- 7: An integer function can appear any place an integer would appear. Here, it is passed the value of num. It will return a 0 or a 1, that value will determine if the Boolean expression is true or false.
- 14: The first line of the function (the header) matches the prototype except that the variables are given names so that they can be used in calculations. There is no semi-colon at the end of the header. Notice that the value of the variable called num was passed to even, but in even it has a different memory location and a different name.
- 16: If n is evenly divisible by 2, it will have a remainder of zero. If the n is even a 1 is returned, otherwise a 0 is returned. The return statement is the last line executed by a function. In the code shown below, the line to print "Bye" does not execute because the function ends as soon as the return statement is executed.

```
int even(int n)
// returns 1 if n is divisible by 2, 0 otherwise
{ if (n%2==0) return 1; // even if n/2 has remainder of 0
  else return 0; // 0 is false
  cout<<"Bye\n";
} // even
```

Structured Programs

A program is referred to as a “structured program” if main has a minimum of code, primarily to loop and call other functions. Each function has one clearly defined purpose. Advantage of structured programs include:

- It is easier to write a program if it is broken into steps;
- The program can be written by several programmers;
- A function that works can be used in other programs;
- If the program has to be modified, the modification may be restricted to just one function.

Example: The Shipping program from Chapter 1 is rewritten to use functions. This program is slightly longer, but it is easier to read. Compare this program to the one in Chapter 1.

```
0 // Decide which company to ship with for several packages
1 #include <iostream.h>
2 double calcSpeedy(int, char); // names of variables optional
3 double calcReliable(int wt, char overNite);
4 void chooseShipper(double s, double r);
5
6 int main()
7 { double speedy, reliable; // local, available only to main
8   int pounds; // used for weight of package
9   char rush; // 'Y' for overnight, 'N' if no rush
10  cout<<"Send overnight? Y or N (Q to quit):";
11  cin>>rush;
12  while (rush != 'Q' && rush != 'q')
13  { cout<<"Enter weight of package in pounds:";
14    cin>>pounds;
15    speedy = calcSpeedy(pounds, rush);
16    reliable = calcReliable(pounds, rush);
17    chooseShipper(speedy, reliable);
18    cout<<"Send overnight? Y or N (Q to quit):";
19    cin.ignore(100, '\n');
20    cin>>rush;
21  } // while
22  return 0;
23 } // main
24
25 double calcSpeedy(int wt, char overNite)
26 { // calculate cost to ship with Speedy
27   if (overNite == 'Y' || overNite == 'y')
28   { if (wt <= 30) return 10.00;
29     else if (wt <= 70) return 20.00;
30     else return 0.00;
31   } // overNite
32   else return 0.50 * wt;
33 } // calcSpeedy
34
35 double calcReliable(int wt, char overNite)
36 { // calculate cost to ship with Reliable
37   if (overNite == 'Y' || overNite == 'y')
```

```
38     return 7.00 + 0.20 * wt;
39     else return 5.00 + 0.10 * wt;
40 } // calcReliable
41
42 void chooseShipper(double s, double r)
43 // Print which shipper and cost.
44 // Remember that 0 for Speedy is not accepted!
45 { if (s == 0 || r < s) //Speedy rejects or Reliable less
46     cout << "Reliable: " << r << "\n";
47     else if (s == r) // Speedy and Reliable same cost
48     cout << "Either Speedy or Reliable: " << r << "\n";
49     else
50     cout << "Speedy: " << s << "\n";
    } // chooseShipper
```

Explanation: When you write programs like this, write and test one piece at a time, that is one of the advantages of structured programs.

- 0: A comment to explain what the program does.
- 1: Include header file for standard input and output. This allows us to use cin and cout.
- 2: Prototypes or the function must appear before the function can be called. This function will receive an integer and a character and return a float. When the function is called it must be passed an int and char in that order!
- 3: Names are given here for the variables, different names can be used in the actual function. These names can also be the same names used for the variables in main. The only reason different names are used here is to emphasize the fact that the value passed is stored in a different variable in the function.
- 4: The function chooseShipper receives two floats and returns nothing. (It is a void function.) Names are given to the variables because it is important to remember which variable is first and which is second. The compiler won't object if they get mixed up, but the output would be wrong.
- 6: Main is an integer function.
- 7-9: All of the variables declared here are local to main. In order for the other functions to use these variables, they must be passed from main.
- 10: The user needs to know what is expected.
- 12: This program loops to do several packages. The loop ends when the user types 'Q' or 'q'. It continues while rush is not 'Q' and is not 'q'.
- 13,14: If the user types 'Q' the program does not ask the weight of the package.
- 15: Call function calcSpeedy, sending it the weight of the package and whether to rush or not. Store the value returned in the variable speedy.
- 16: Same as 15 to calculate reliable.
- 17: Call chooseShipper to print out which company gives the lowest price. The order of variables passed is important.
- 18: Print prompt for next package.
- 19,20: It is necessary to flush the standard input buffer before reading a character. The next statement after this is to check the value of rush in the while statement.
- 21: End of the while loop.

- 22: The program ends here for now, adding a count and total cost would be a good idea!
- 23: End of main.
- 25: The function calcSpeedy receives the weight of the package and whether to ship overnight or not. It returns the cost to ship with speedy.
- 27-32: None of the elses are necessary. The program returns a value and never continues with the next line. The else is here for clarity only.
- 35-40: This is basically the same as calcSpeedy. Using the same names in both functions makes it easier to write, but they are different variables.
- 42: This function depends on receiving Speedy cost first, then Reliable cost.
- 45-50: This is only slightly different from the code in Chapter 1.

Pass by Reference

The program above passed the weight of the package and the variable rush to the functions by value. If a function needs to change the value of a variable it must be passed by reference. The & indicates that the variable will receive an address.

Example: the program below reads in two values, swaps so that they are in order, then prints them. Swap receives the variables by reference.

```

0 //Read 3 numbers, sort with function, then print
1 #include <iostream.h>
2 void swap(int& first, int& second) //passed by reference
3 { int temp;
4   temp = first;    //temp is a local variable
5   first = second; //first and second are references
6   second = temp;  // to the variables that were passed
7 } //swap
8
9 main() //execution always starts in main
10 { int num1, num2, num3;
11   cout<<"Enter the first number:";
12   cin>>num1;
13   cout<<"Enter the second number:";
14   cin>>num2;
15   cout<<"Enter the third number:";
16   cin>>num3;
17   if (num1>num2) //swap the two numbers
18     swap(num1, num2); //pass num1 and num2 by reference
19   if (num2>num3) //swap the two numbers
20     swap(num2, num3); //pass num2 and num3 by reference
21   if (num1>num2) //swap the two numbers
22     swap(num1, num2); //pass num1 and num2 by reference
23   cout<<"In order: "<<num1<<" "<<num2<<" "<<num3<<"\n";
24   return 0;
25 } //main

```

Explanation:

- 10: The variables num1 and num2 are passed to swap by reference

- 15: The variables first and second are reference variables. Instead of being given their own locations in memory, they simply refer to the variables that are passed.
- 16: The variable temp is local to swap. Memory is allocated when swap starts. The memory is de-allocated when swap ends.

Example: The program below loops to calculate the letter grade for several students. A negative midterm signals the end of the loop.

```
0 // Student Grades
1 #include <iostream.h>
2 int getInfo(int&, int&);//use reference var to change values
3 char calcGrade(int, int); //values: not to be changed
4 int main()
5 // loop until getInfo returns false
6 { int midterm, final;
7   char grade;
8   while (getInfo(midterm,final))
9     { grade = calcGrade(midterm, final);
10      cout<<"Grade = "<<grade<<"\n";
11    } // while loop
12   return 0;
13 } // main
14
15 int getInfo(int &mid, int &fin)
16 // get midterm and final grade
17 { cout<<"Enter midterm grade: ";
18   cin>>mid;
19   if (mid < 0 )
20     return 0; // signal end of data by returning false
21   else // valid midterm get final & return true
22     { cout<<"Enter final grade: ";
23       cin>>fin;
24       return 1;
25     } // valid midterm
26 } // getInfo
27
28 char calcGrade(int mid, int fin)
29 // calculate average, return letter grade
30 { int avg;
31   avg = (int)((mid + fin+1.0)/2.0);// rounds to nearest int
32   if (avg < 65) return 'F';
33   if (avg < 70) return 'D';
34   if (avg < 80) return 'C';
35   if (avg < 90) return 'B';
36   return 'A';
37 } // calcGrade
38
```

Explanation:

0. Begin every program with a comment.

1. Include `iostreamo.h` for `cin` and `cout`.
2. Prototype for an integer function `getInfo` that receives two integers by reference.
3. Prototype for a character function `calcGrade` that receives two integers by value.
4. A space to separate functions is just to make it easier to read.
5. `Main` is an integer function that receives nothing.
6. Every function should have at least one comment to explain its purpose.
7. Integer variables `midterm` and `final` are local to `main`.
8. Character variable `grade` is also local to `main`.
9. The function `getInfo` returns `true` if the loop should continue. It receives addresses of `midterm` and `final` so that it can store values in those variables.
10. The function `clacGrade` receives the values of `midterm` and `final` and returns the letter grade.
11. The grade is printed. Statements 11 and 12 could be combined, try it!
16. The integer function `getInfo` receives `mid` and `fin` by reference. Both `mid` and `fin` have an address as their value.
22. This program ends when the user enters a negative value for the midterm. Does the user know that? Modify the prompt to tell the user how to end the program.
23. If a valid midterm was entered, read in the final grade and return `true`.
30. The function `calcGrade` receives the midterm and final but does not need to change them so they are passed by value.
33. This statement rounds to nearest integer. Test this by giving it the grades 89 and 90, the grade should be an 'A'.
34. As soon as the grade is known it is returned. All other statements in the function are skipped. A nested `if else` is not necessary because the function ends after a value is returned.

Example: The program below calculates gross pay for employees. Separate functions are used to get information, calculate pay is broken into separate functions for regular and overtime pay. A separate function prints the pay amounts. This program only computes gross pay, no withholding tax is withheld. When we are ready to deduct withholding, it will be a separate function. When we are ready to read information about the employee from a file, only the function `getInfo` has to be modified. If we want to print a pay check for an employee instead of just a single line, only function `printInfo` needs to be changed.

```
0 // Pay program
1 #include <iostream.h>
2 int getInfo(int &hrsWorked, double &hrlyRate);
3 void calcPay(int hours, double rate, double &regPay,
4   double &overPay);
5 void printInfo(double regPay, double overPay,
6   double grossPay);
7 const regHours = 40; //constant
8
9 int main()
10 { double totalReg=0.00, totalOver=0.00;
11   int hrsWorked;
12   double hrlyRate;
13   double regPay, overPay;
14   while (getInfo(hrsWorked, hrlyRate))
```

```
15 { calcPay(hrsWorked, hrlyRate,regPay, overPay);
16   printInfo(regPay,overPay,overPay+regPay);
17   totalReg+=regPay;
18   totalOver+=overPay;
19 } // while
20 cout<<"TOTAL PAYROLL\n";
21 printInfo(totalReg,totalOver,totalReg+totalOver);
22 return 0;
23 } // main
24
25 int getInfo(int &hrs, double &rate)
26 //Read hourly rate. If rate is greater than 0, read hours
27 // worked and return true, else return false to signal end
28 { cout<<"Enter hourly rate: ";
29   cin>>rate; // rate is already an address: NO & !!!
30   if (rate > 0.00)
31   { cout<<"Enter hours worked: ";
32     cin>>hrs; // hrs is already an address: NO & !!!
33     return 1;
34   } // person read in
35   else return 0; //rate of 0 signals end of data
36 } // getInfo
37
38 void overtime(int hours,double rate,double &regPay,
39   double &overPay)
40 // pay for emp worked more than reg hours.
41 { int overHours;
42   double overRate;
43   regPay = rate * regHours;
44   overHours = hours - regHours;
45   overRate = rate * 1.5; // time and a half
46   overPay = overHours * overRate;
47 } // overtime
48
49 void regular(int hours,double rate,double &regPay,
50   double &overPay)
51 // pay for emp did not work overtime
52 { regPay = rate * hours;
53   overPay = 0.00;
54 } // overtime
55
56 void calcPay(int hours,double rate,double &regPay,
57   double &overPay)
58 // call either overtime or regular to calculate Pay
59 { if (hours > regHours)
60   overtime(hours, rate, regPay, overPay);
61   else
62   regular(hours, rate, regPay, overPay);
63 } // calcPay
64
```

```
65 void printInfo(double regPay,double overPay,  
66 double grossPay)  
67 { cout.precision(2); //print 2 decimal places  
68 cout.setf(ios::fixed);  
69 cout<<"Regular pay= "<<regPay<<"\n";  
70 cout<<"Overtime pay= "<<overPay<<"\n";  
71 cout<<"Gross pay= "<<grossPay<<"\n";  
} //printInfo
```

Explanation: This is more complicated than previous programs. When you write a program like this write and test one piece at a time, that is one of the advantages of structured programs.

- 0: A comment to explain what the program does.
- 1: Include header file for input and output..
- 2: The integer function getInfo receives hours worked and hourly rate by reference. We can give names to the variables if we want, names are optional in the prototype.
- 3,4: The function calcPay receives hours and rate by value, but it must calculate regPay and overPay so these are received by reference. (This statement is on two lines.)
- 5: The function to print out the pay only **uses** the values, it does not **change** any values, so everything is passed by value. It does not return anything.
- 7: regHours is a constant: we want to avoid having a value 40 buried in the code.
- 7: Function main receives no arguments, returns an integer.
- 10: Totals are given initial values of 0.00. (Using 0.00 instead of 0.0 indicates money, little hints can be a big help in abig program!) No variable for total gross is necessary because it can be found by adding totalReg and totalOver.
- 11-13: Initial values are not given to hrsWorked and hrlyRate because these are read in.
- 14: While loop calls getInfo passing hrsWorked and hrlyRate by reference. After reading information getInfo returns true or false. The value returned by getInfo determines whether the loop executes.
- 15: the function calcPay receives hrsWorked and hrlyRate by value, these numbers will not be changed. It receives regPay and overPay by reference because these amounts *will* change.
- 16: The function printInfo receives values to be printed.
- 17-18: Pay amounts are added to the totals.
- 20-21: The function printInfo does double duty to print the totals as well as the pay for each employee.
- 24: Function main returns 0, if it got to this point it ran to completion.
- 25-36: The function getInfo receives hrs and rate by reference. The names do not have to be the same as the names used in main. These variables are references to the variables hrsWorked and hrlyRate in main. If an hourly rate greater than 0 is entered, the user is asked for the hours worked and 1 is returned. Notice that if the user enters 0 for the ratge, they are not asked to input the hours. If the hourly rate is not greater than 0, hours worked is not read in and a 0 is returned.
- 38: There is no prototype for this function. A function, or the functions prototype, must appear before the first call to the function. This function is not called by main, it is called by the function calcPay which appears after it. This was done here to illustrate a point: it is usually better to put prototypes for all function before main. This function receives hours and rate by

value: they will not be changed. It receives regPay and overPay by reference: they will be changed.

41,42: The variables overHours and overRate are local to function overtime.

43: The constant regHours was given a value of 40. This is much easier to change than a 40 buried in the code at this point.

56: This function receives hours and rate by value: they will not be changed. It receives regPay and overPay by reference: they will not be changed directly by this function, but they are passed to either function overtime or function regular and are changed by those functions.

65: All of the information to be printed is passed by value: they will not be changed.

67-68: These 2 statements specify that the numbers will be printed with 22 decimal places.

71: The end! Whew! A program with 71 lines may seem long at this points, but it isn't even finished. Later, it will read information about an employee from a file. It should print to the printer instead of mixing the prompts together with the report on the screen. Also, this program ignores any withholdings.

```

Enter hourly rate: 6.50
Enter hours worked: 12
Regular pay= 78.00
Overtime pay= 0.00
Gross pay= 78.00
Enter hourly rate: 7.00
Enter hours worked: 42
Regular pay= 280.00
Overtime pay= 21.00
Gross pay= 301.00
Enter hourly rate: 8.25
Enter hours worked: 40
Regular pay= 330.00
Overtime pay= 0.00
Gross pay= 330.00
Enter hourly rate: 0
TOTAL PAYROLL
Regular pay= 688.00
Overtime pay= 21.00
Gross pay= 709.00

```

Programming Exercises:

1: Triangle problem

Read in the three sides of a triangle, if it is a triangle, print what kind. A triangle is equilateral if all three sides are equal; it is isosceles if any two sides are equal but not three; it is scalene if no sides are equal. A triangle is right if $c^2 = a^2 + b^2$. It is obtuse if $c^2 > a^2 + b^2$. It is acute if $c^2 < a^2 + b^2$.

Method: Follow the steps below, write and test each step.

- Read in three side using a function getSides. Print a line (with no line feed) that says The sides 6, 4, and 5:
- Write a function to sort the three sides. Compare and swap so that A and B are in order. Then compare and swap so that B and C are in order. Last, compare and swap so that A and B are in order. This last step is necessary if the numbers are 3, 4 and 2, or a comparable

arrangement.. Put this function call before the print statement. Test the numbers should now print out in order: The sides 4, 5, and 6: *(If the sort gives you a problem, you can write the rest of the program and put in the sides in order until you get the sort to work.)*

- Write a function to determine if the three sides could make a triangle (a function triangle returns true or false). Note: take sticks 3", 5" and 10": can you form a triangle with them? Add the statement to call the function. Print "not a triangle.\n" if it is not a triangle. Add a block to add the next two calls if it is a triangle.
- Write a function to print "equilateral", "isosceles" or "scalene".
- Write a function to print "right\n", "obtuse\n", or "scalene".

The output for a triangle should appear as shown below:

```
The sides 3, 4 and 5: scalene right
The sides 3, 4 and 10: not a triangle
```

Adding a loop to the program will make it easier to test. How will the loop end?

2: GPA Program

Write a program that loops to ask the number of credits and grade for each course. It prints the students GPA at the end. The GPA is calculated as follows: A grade of 'A' is 4 points; a 'B' is 3 points, a 'C' is 2 points, a 'D' is 1 points, an 'F' is 0 points. Multiply the points by the credits to get creditPoints. Add total credits, and total creditPoints. The GPA is the total points divided by credits. Use functions for as much as possible.

Example:

	Credits	Grade	Points	CreditPoints
	3	A	4	12
	1	C	2	2
	3	B	3	9
Total	<u>7</u>			<u>23</u>

GPA = 3.2

Challenge: Add an outer loop to do more than 1 student. How will the outer loop end?