COMSC-110 Chapter 8

Writing complicated programs: Subprograms

Valerie Colber, MBA, PMP, SCPM

Functions

- C++'s subprograms are called *functions*.
- But there are no functions in the C++ library to do all the things we would ever want to do.
- Sequential processing *jumps* to another set of instructions...and *returns* when that is completed.
- Subprogram identifiers are distinguished from variable identifiers by: parentheses.
- Subprograms use parameter (data) lists to share variables declared in "main".

Sources of Functions

1. Programmer defined
2. Library
3. Programming language built-in functions (for example, “dot” functions)
4. Main

Types of functions

1. "call" as **statement: void**
2. "call" as **expression: value-returning**

Why use functions?

- **modularization**: for organizing complicated code
- **code reuse**: for code blocks that get repeated
- Get user input
- Input validation
- Password protection validation
- Input processes

EXAMPLE:
passingGradeFun.cpp

Function definition

- *function definition* -- it is a container for the detailed code that got removed from "main" and replaced with a call.
- It is a program within a program (hence the word "subprogram").
- The first line of a function definition is the *function header*.
- It gets executed whenever a call is made that references it by its name.

How to create a value returning function

- move code block into a subprogram...
- ...replace with a "call" *expression*

Discontinuity of variables

- variables in "main" are *not available* to subprograms
- "parameter lists" are used to share data values

Beginning and end of a subprogram

- Note that the *first* statement in the data code block subprogram creates a variable whose data type matches the return type of the function.
- It also sets it to a default value.
- This is the value that the function will return unless it gets changed before the **return** statement, which is the *last* statement in the subprogram.

Return statement

- Any value-returning function *must* have a **return** statement as the last statement in the definition.
- Following the keyword "return", there needs to be a value, variable, or expression that matches the *return type* .
- Actually, "main" is a function -- note that its return type is "int", and that it ends with the statement **return 0;** , in which the "0" is an "int" value, matching the return type.

```
...
//programmer defined functions
// subprogram to get yes/no answer
string getAnswer()
{
  //data
  string answer; // a variable owned by the function

  //input user response
  while (true)
  {
    cout << "Your answer [yes/no]: ";
    getline(cin, answer);
    if (answer == "yes") break;
    if (answer == "no") break;
    cout << "Let's try this again.\n";
  } //while true
  return answer; // its value replaces call in "main"
}//getAnswer

//main program
int main()
{
...
  if (getAnswer() == "yes") // details left to subprogram
...
...
} //main
```

- getAnswer() is the "call"
- subprogram name is a valid identifier, just like a variable name
- to distinguish from a *variable*, "calls" use **parentheses**
- the call is an *expression*, resulting in a string value
- actually, can be any data type, including int, double, or char
- the "return" statement: last statement; matches "return type"

Parameter lists

- A function's parentheses identify it as a function instead of a variable.
- A function's parentheses also are a container for *inputs* to the function.
- It is a way for the call to send values to the function for processing.

Parameter lists

- If a function has empty parentheses, then it has no inputs.
- The number and type of the values in the parameter list of a call *must* match what the function expects them to be.
- This is specified in the *header*.

paramList.cpp

```
…
//libraries
#include <iostream>
using namespace std;
…
//programmer defined functions
//calculate the average of 2 numbers
double calcAverage(int a, int b)
{
  double avg = 0.0;
  avg = (a + b) / 2.0;
  return avg;
} // calcAverage

//main program
int main()
{
  //data
  int x = 100;
  int y = 200;
  double z = calcAverage(x, y);

  //output
  cout << z << endl;
} // main
```

- It is important to remember that parameters in calls are evaluated as values, and they are used to initialize the variables in the function's parameter list.
- That is why calls can have values, variables, or expressions - because the all can be reduced to values.
- Note how the variable names in "main" do not match those in "calcAverage" -- they do not have to.
- It is not their names that determine how they transfer their values to the function -- it is their location in the parameter list that does.
- Also note that the setting of **avg** to a default value of zero is not really necessary.

Alternative Ways To Write calcAverage

```
double avg = 0.0; // a default
avg = (a + b) / 2.0;
return avg;
```

```
double avg; // no default needed
avg = (a + b) / 2.0;
return avg;
```

```
double avg = (a + b) / 2.0;
return avg;
```

```
return (a + b) / 2.0;
```

```
return ((a + b) / 2.0);
```

Parameter Lists

- Parameter lists are used to exchange data between main and the function.
- Remember: main's variables not accessible to function.
- Solution: put values inside the call's parentheses (comma-separated) as values, variables, or expressions.
- Put matching declarations in function definition to specify data type of "passed" value(s) to name function's copies of values.

Examples of Value-Returning Functions

EXAMPLE

askQuestion.cpp

Void function

- Sometimes functions produce their own output and do not need to return anything to "main".
- In this case, the return type can be specified as **void** to indicate that there is no **return** statement.

How to create a void function

- move code block into a subprogram...
- ...replace with a "call" *statement*

Examples of Void Function

```
…
//libraries
#include <iostream>
using namespace std;
…
//programmer defined functions
//output greeting
void greetings()
{
  cout << "Hello!" << endl;
  cout << "How are you today?" << endl;
}//greetings

//main program
int main()
{
  greetings();
  greetings();
}
```

```
//libraries
#include <iostream>
#include <string>
using namespace std;
…
//programmer defined functions
//output greeting
void greetings(string x)
{
  cout << "Hello, " << x << '!' << endl;
  cout << "How are you today?" << endl;
}//greetings

//main program
int main()
{
  //data
  string name = "Robert Burns";

  //output personalized greeting
  greetings(name);
}//main
```

```
…
//libraries
#include <iostream>
#include <string>
using namespace std;
…
//programmer defined functions
//output greeting
void greetings(string x, string y)
{
  cout << "Hello, " << x << ' ' << y << '!' << endl;
  cout << "How are you today?" << endl;
}//greetings

//main program
int main()
{
  //data
  string fname = "Robert";
  string lname = "Burns";

  //output personalized greeting
  greetings(fname, lname);
}//main
```

EXAMPLE

addition.cpp

Comparing programs

With and without Functions

itsAboutYou.cpp without functions

```
//libraries
#include <iostream>
#include <string>
using namespace std;
…
//main program
int main()
{
  // read an int value from the keyboard
  int age;
  cout << "What is your age? ";
  cin >> age;
  cin.ignore(1000, 10);

  // read a double value from the keyboard
  double gpa;
  cout << "What is your grade point average? ";
  cin >> gpa;
  cin.ignore(1000, 10);

  // read a string value from the keyboard
  string name;
  cout << "What is your name? ";
  getline(cin, name);

  // read a char value from the keyboard
  char gender;
  cout << "What is your gender? [M/F]: ";
  cin >> gender;
  cin.ignore(1000, 10);
  // process the input
   ...

}//main
```

itsAboutYou.cpp with Functions

```
//libraries
#include <iostream>
#include <string>
using namespace std;
…
//programmer defined functions
int getAge()
{
  int age;
  cout << "What is your age? ";
  cin >> age;
  cin.ignore(1000, 10);
  return age;
}//getAge
double getGpa()
{
  double gpa;
  cout << "What is your grade point average? ";
  cin >> gpa;
  cin.ignore(1000, 10);
  return gpa;
}//getGpa
string getName()
{
  string name;
  cout << "What is your name? ";
  getline(cin, name);
  return name;
}//getName
char getGender()
{
  char gender;
  cout << "What is your gender? [M/F]: ";
  cin >> gender;
  cin.ignore(1000, 10);
  return gender;
}//getGender

//main program
int main()
{
  // read personal data from keyboard
  int age = getAge();
  double gpa = getGpa();
  string name = getName();
  char gender = getGender();
  // process the input
  ...
}//main
```

itsAboutYou.cpp Without Validation

```
#include <iostream>
#include <string>
using namespace std;

int getAge()
{
  int age;
  cout << "What is your age? ";
  cin >> age;
  cin.ignore(1000, 10);
  return age;
}

double getGpa()
{
  double gpa;
  cout << "What is your grade point average? ";
  cin >> gpa;
  cin.ignore(1000, 10);
  return gpa;
}

string getName()
{
  string name;
  cout << "What is your name? ";
  getline(cin, name);
  return name;
}

char getGender()
{
  char gender;
  cout << "What is your gender? [M/F]: ";
  cin >> gender;
  cin.ignore(1000, 10);
  return gender;
}

int main()
{
  // read personal data from keyboard
  int age = getAge();
  double gpa = getGpa();
  string name = getName();
  char gender = getGender();

  // process the input
  ...

  return 0;
}
```

itsAboutYou.cpp With Validation

```
#include <iostream>
#include <string>
using namespace std;

int getAge()
{
  int age;
  while (true)
  {
    cout << "What is your age? ";
    cin >> age;
    cin.ignore(1000, 10);
    if (age >= 0 && age < 100) break;
    cout << "That can't be right -- try again\n";
  }
  return age;
}

double getGpa()
{
  double gpa;
  while (true)
  {
    cout << "What is your grade point average? ";
    cin >> gpa;
    cin.ignore(1000, 10);
    if (gpa >= 0.0 && gpa <= 4.0) break;
    cout << "That can't be right -- try again\n";
  }
  return gpa;
}

string getName()
{
  string name;
  do
  {
    cout << "What is your name? ";
    getline(cin, name);
  } while (name.length() == 0);
  return name;
}

char getGender()
{
  char gender;
  while (true)
  {
    cout << "What is your gender? [M/F]: ";
    cin >> gender;
    cin.ignore(1000, 10);
    if (gender == 'M' || gender == 'F') break;
    cout << "I was expecting M or F -- try again\n";
  }
```

Random numbers

- C++ provides a *random number generator*. To draw an integer with a value between 0 and 9, inclusive, use this expression:

rand() % 10

- Each result is equally probable. Add 1 to the above in order to get numbers in the range 1 to 10, inclusive.
- Here's how to simulate the roll of a six-sided die:

1 + (rand() % 6)

- Here's how to simulate the roll of two six-sided dice:

(1 + (rand() % 6)) + (1 + (rand() % 6))

Random numbers

- Use of random numbers requires two includes:

  #include <ctime> and **#include <cstdlib>**

- You also need this as the first statement in "main":

  srand(time(0));

- Here's something you need to remember -- if you use **srand(time(0));** , it should not appear anywhere else in your program except as the first statement in "main", so that it is never executed more than once in a run of a program.

EXAMPLE

keepingScore.cpp

Shorthand operator examples

- **score ++;**
- **score += 1;**
- **score --;**
- **score -= 1;**
- **result *= 10;**
- **average /= n;**

- **score = score +1;**
- **score = score + 1;**
- **score = score - 1;**
- **score = score - 1;**
- **result = 10 * result;**
- **average = average / n;**

Function prototypes

- At some point you may start writing programs that are *very* complicated, using multiple source files for a single program (although this is not covered in this text).
- In such cases it is common practice for C++ programmers to write the "main" function above all others, instead of putting it last as we have just learned to do.
- This practice requires the use of *function prototypes*, which are actually nothing more than promises to the compiler that you will write a function definition *after* the compiler sees a reference to the function.

Function prototypes

- C++ compiler needs identifiers declared *before* they are used for more advanced applications, may need to worry about order in which functions appear in code.
- solution: declare function, but leave definition until *after* main.
- List function prototypes under the programmer defined functions in the template in order of when called in the program

passwordProto.cpp

```
…
//libraries
#include <iostream>
#include <string>
using namespace std;
…
//programmer defined functions
void getPassword();
…
//main program
int main()
{
…
  getPassword();
  // do stuff...
} // main

//getPassword
void getPassword()
{
  //data
  //none

  //input password and verify
  while (true)
  {
    string password;
    cout << "Enter the password: ";
    getline(cin, password);
    if (password == "12345") break;
    cout << "INVALID. ";
  } // while
} // getPassword
```

- The prototype is the function's header, with a semicolon at the end. It tells the compiler "if you see a reference to a function named "getPassword" with an empty parameter list, don't worry because I will define that function later".
- Prototypes go where the function definitions would have gone.
- Prototypes are listed in the order that they are called

EXAMPLE

askQuestion2.cpp

Compare

askQuestion.cpp

and

askQuestion2.cpp

What program template do we use now?

- From now on in the course, we will be using function prototypes.
- So from now on in the course, use:

programTemplateProto.cpp

How to write algorithms for subprograms

.... (1. objective, 2. requirements for INPUT, PROCESSING, OUTPUT, DATA)

3. Algorithm Instructions

algorithm for subprogram "getGrade"

1000 grade = ‘X’
1005 output prompt “What is your grade [A, B, C, D, F or X to exit]: “
1010 input grade from console
1020 if grade is A, B, or C, skip to instruction 1070
1030 if grade is D or F, skip to instruction 1070
1040 if "grade" is X or x, skip to xxx
1050 output “INVALID, TRY AGAIN!”
1060 skip to instruction 1005
1070 Return from subprogram getGrade with grade value

algorithm for "main"

10 grade= grade from subprogram "getGrade"
20 if grade is X or x, skip to instruction 50
30 if grade is A, B, or C, output “YOU PASS!”
40 skip to instruction 10
50 output THANKS!

…(test cases)

How to write algorithms for subprograms with parameters

…(objective, requirements for INPUT, PROCESSING, OUTPUT, DATA)

***algorithm for subprogram add(input: number1, number2)
1000 sum = number1 + number2
1010 return from subprogram add with value sum

***algorithm for introduction (input: labNumber, objective)
2000 output objective
2020 output author
2030 output editor and compiler used
2040 output program filename
2050 output date and time compiled
2060 output instructions
2070 return from subprogram introduction

***main algorithm
10 subprogram introduction(“example”, “sum 2 numbers”)
20 output prompt “Please input first number: “
30 input number1
40 output prompt “Please input second number: “
50 input number2
60 result = subprogram add(number1, number2)
70 output number1, “ + “, number2, “ = “, result
80 END

…(test cases)

References

- INTRODUCTION TO PROGRAMMING USING C++ by Robert Burns