5 Iterations

5.1 Introduction

A computer can perform the same task many times without human intervention. Suppose we want to print the word GO three times. We could write

```
printf(" GO");
printf(" GO");
printf(" GO");
printf("!");
```

But if we want to print the word GO one hundred times, then this approach is a little tedious. We need a better method. Such a method is used in the following program.

```
/* program 5.1 - prints GO 3 times. */

#include <stdio.h>

int main()
{
  int i = 0;

  while (i < 3) {
    printf(" GO");
    i = i + 1;
  }
  printf("!\n");
  return 0;
}
```

When executed, this program prints

```
 GO GO GO!
```

on the screen. How does it work?

Initially, the variable *i* contains the value zero. *i* is less than three and so *GO* is printed and *i* is increased by one.

i now contains one. *i* is still less than three and so *GO* is printed and *i* is increased by one.

i now contains two. *i* is still less than three and so *GO* is printed and *i* is increased by one.

i now contains three. *i* is not less than three and so the exclamation mark is printed.

So, for as long as (that is, while) the Boolean expression *(i < 3)* remains TRUE, *GO* is printed

and the value held in *i* is increased by 1. Eventually, there comes a time when the Boolean expression *(i < 3)* is not TRUE; then the ! is printed and the program terminates.

Repeatedly executing a group of statements is known as a looping. The following diagram shows why.

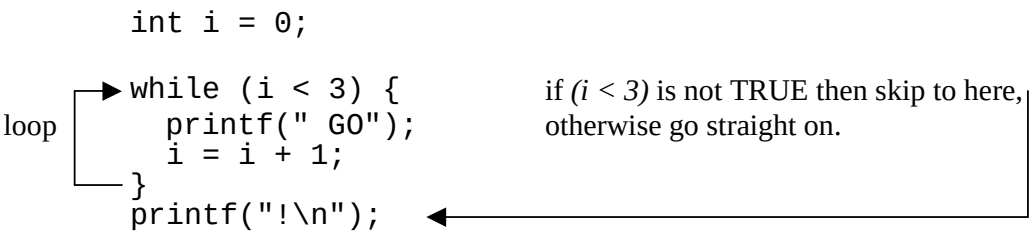

Figure 5.1 - The two statements printf(" GO") and i = i + 1 are repeatedly executed for as long as i remains less than 3.

The construct

```
while (i < 3) {
   ...
}
```

is an example of an iteration. An iteration construct is used whenever a sequence of statements is to be repeatedly executed. An iteration construct is also known as a loop or repetition construct.

5.2 Simple Validation

Students are asked to rate, on a scale from 0 (appalling) to 5 (excellent), the quality of food supplied by the refectory. The results of the survey are to be held in a computer system for subsequent analysis - but we shall not go into that here. However, it is important that only values within the range 0 to 5 inclusive are stored. So, if a value outside this range is entered, then the user is again invited to enter an appropriate value; this is repeated until a value between 0 and 5 inclusive is entered.

To input one value is straightforward.

```
rating =
     intNumberRead("Rating on a scale of 0 to 5 inclusive? ");
```

We need to determine whether the number entered is within the required range.

```
isInRange(rating, 0, 5);
```

If the number is not in the required range, we invite the user to re-enter the rating and check whether the number entered is within the required range.

```
if (!isInRange(rating, 0, 5)) {
  printf("Please choose a value between 0 and 5 only.\n");
  rating = intNumberRead("Rating? ");
}
```

We need to repeatedly execute these last two statements for as long as the number entered remains out of range.

```
while (!isInRange(rating, 0, 5)) {
  printf("Please choose a value between 0 and 5 only.\n");
  rating = intNumberRead("Rating? ");
}
```

Putting it all together we get

```
int rating =
     intNumberRead("Rating on a scale of 0 to 5 inclusive? ");

while (!isInRange(rating, 0, 5)) {
  printf("Please choose a value between 0 and 5 only.\n");
  rating = intNumberRead("Rating? ");
}

printf("Thank you.\n");
```

Here is the complete program.

```
/* program 5.2 - simple validation. */

#include <stdio.h>

int intNumberRead(char prompt[]);
/* post-condition: returns number entered at the keyboard. */

int isInRange(int value, int min, int max);
/* returns 1 if min <= value <= max, otherwise returns 0. */

int main()
{
  int rating =
     intNumberRead("Rating on a scale of 0 to 5 inclusive? ");

  while (!isInRange(rating, 0, 5)) {
    printf("Please choose a value between 0 and 5 only.\n");
    rating = intNumberRead("Rating? ");
  }
  printf("Thank you.\n");
  return 0;
}
```

```
int intNumberRead(char prompt[])
{
  char string[BUFSIZ];
  int number = 0;

  printf("%s", prompt);
  gets(string);
  sscanf(string, "%d", &number);
  return number;
}

int isInRange(int value, int min, int max)
{
  return ((min <= value) && (value <= max));
}
```

Two examples of program runs are

(1)
```
Rating on a scale of 0 to 5 inclusive? -1
Please choose a value between 0 and 5 only.
Rating on a scale of 0 to 5? 0
Thank you.
```

(2)
```
Rating on a scale of 0 to 5 inclusive? 6
Please choose a value between 0 and 5 only.
Rating on a scale of 0 to 5 inclusive? 5
Thank you.
```

5.3 Summing a Sequence of Numbers

A point-of-sale terminal in a supermarket is used to input the price of each item in pence (data entry is quicker if the decimal point is not entered) and to add the price to a running total. When zero is entered (for example, by pressing the return key alone), the total to be paid by the customer is displayed.

We consider three cases:

(a) before any item is entered
(b) when only one item is entered
(c) when several items are entered

(a) Before any item is entered, we expect the running total to be zero.

```
int runningTotal = 0;
```

(b) When one item is entered, we add its cost to the running total but only if the cost is not zero.

```
costOfItem = intNumberRead("Price? ");
if (costOfItem != 0)
  runningTotal = runningTotal + costOfItem;
```

(c) But when several items are entered, we repeatedly add the cost of each item to the running total provided the cost of an item is not zero.

```
costOfItem = longNumberRead("Price? ");
while (costOfItem != 0) {
  runningTotal = runningTotal + costOfItem;
  costOfItem = intNumberRead("Price? ");
}
```

Items are entered for as long as the cost of an item is not zero.

Putting the three cases together we obtain

```
int runningTotal = 0;
int costOfItem = intNumberRead("Price? ");

while (costOfItem != 0) {
  runningTotal = runningTotal + costOfItem;
  costOfItem = intNumberRead("Price? ");
}
```

Here is a simple program which simulates a point-of-sale terminal.

```
/* program 5.3 - point of sale simulation. */

#include <stdio.h>

int intNumberRead(char prompt[]);
/* post-condition: returns the number entered at the keyboard.
*/

int main()
{
  int runningTotal = 0;
  int costOfItem = intNumberRead("Price? ");

  while (costOfItem != 0) {
    runningTotal = runningTotal + costOfItem;
    costOfItem = intNumberRead("Price? ");
  }
  printf("Total bill is £%0.2f\n", runningTotal/100.0);
  return 0;
}

int intNumberRead(char prompt[])
{
  char string[BUFSIZ];
  int number = 0;

  printf("%s", prompt);
  gets(string);
  sscanf(string, "%d", &number);
  return number;
}
```

Here are three examples of program runs.

(1)
```
Price? 89
Price? 89
Price? 79
Price? 179
Price? 0
The balance due is £4.36
```

(2)
```
Price? 100
Price? 0
The balance due is £1.00
```

(3)
```
Price? 0
The balance due is £0.00
```

5.4 Printing a Table

A Boat Company offers four-berth boats for hire for up to six days at a time. An example of their hire charges is shown in the table below.

Boat Hire Charges

Period	Charge
1 day	£10.00
2 days	£20.00
3 days	£30.00
4 days	£40.00
5 days	£50.00
6 days	£60.00

We are to design a program which will input a daily charge and output the table. We consider four cases:

(a) before any entries for any one day is printed
(b) a table with just one entry for one day
(c) a table with two entries for two days
(d) a table with six entries

(a) Before any entries are printed we input the daily charge and print the headings.

```
costPerDay = doubleNumberRead("Charge per day? ");
printf("BOAT HIRE CHARGES\n\n");
printf("Period     Charge\n\n");
```

(b) To print just one entry we print the first line of the table.

```
dayNumber = 1;
printf("%d day   £%2.2f\n", dayNumber, costPerDay);
```

(c) To print the next entry we add one to dayNumber and print the second line of the table.

```
dayNumber = dayNumber + 1;
printf("%d days   £%2.2f\n", dayNumber, costPerDay);
```

(d) We print the last line when dayNumber contains six.

```
dayNumber = dayNumber + 1;
while (dayNumber < 7) {
  printf("%d days    £%0.2f\n", dayNumber, costPerDay *
                dayNumber);
  dayNumber = dayNumber + 1;
}
```

The following program inputs the hire charge for one day and then prints the table on the screen.

```
/* program 5.4 - prints a table of hire charges. */

#include <stdio.h>

double doubleNumberRead(char prompt[]);

int main()
{
  double costPerDay = doubleNumberRead("Charge per day? £");
  int dayNumber = 1;

  printf("\nBOAT HIRE CHARGES 1994\n\n");
  printf("Period     Charge\n\n");
  printf("%d day      £%0.2f\n", dayNumber, costPerDay);
  dayNumber = dayNumber + 1;
  while (dayNumber < 7) {
    printf("%d days     £%0.2f\n", dayNumber,
                costPerDay * dayNumber);
    dayNumber = dayNumber + 1;
  }
}

double doubleNumberRead(char prompt[])
{
  char string[BUFSIZ];
  double number = 0.0;

  printf("%s", prompt);
  gets(string);
  sscanf(string, "%lf", &number);
  return number;
}
```

An example of a program run is

```
Daily charge? £10

BOAT HIRE CHARGES 1994

Period   Charge

1 day    £10.00
2 days   £20.00
3 days   £30.00
4 days   £40.00
5 days   £50.00
6 days   £60.00
```

5.5 Designing a Loop Construct

We illustrate the principles with an example. We are required to design and write a program which will provide examination statistics for an unspecified number of students. We are required to provide statistics showing the total number of students who sat an examination and, of those, how many passed. The examination marks are in the range 0 to 100 and a pass is awarded to each mark which is 50 or more.

First, we establish the initial conditions by looking at the case when the number of students who sat the exam is zero.

Solution for totalStudents == 0

```
int studentsPassed = 0;
int studentsProcessed = 0;
```

Then, using this solution, we obtain the solution for the case when only one student sat the exam.

Solution for totalStudents == 1

```
int examinationMark = intNumberRead("Mark? ");

if (examinationMark >= passMark)
   studentsPassed = studentsPassed + 1;
studentsProcessed = studentsProcessed + 1;
```

But we do not know how many students sat the exam. We need to be able to determine when there are no more examination marks to be entered. Since an examination mark is between 0 and 100 inclusive, we could use a mark of -1 to mean no more input. So we write

```
examinationMark = intNumberRead("Mark? ");
while (examinationMark != -1) {
  if (examinationMark >= passMark)
     studentsPassed = studentsPassed + 1;
  studentsProcessed = studentsProcessed + 1;
  examinationMark = intNumberRead("Mark? ");
}
```

The completed design is

```
enum { passMark = 50 };
int studentsPassed = 0;
int studentsProcessed = 0;
examinationMark = intNumberRead("Mark? ");
while (examinationMark != -1) {
  if (examinationMark >= passMark)
     studentsPassed = studentsPassed + 1;
  studentsProcessed = studentsProcessed + 1;
  examinationMark = intNumberRead("Mark? ");
}
writeTotals(studentsProcessed, studentsPassed);
```

Check:

When there are no students and the mark entered is -1, the loop is not executed and both *studentsPassed* and *studentsProcessed* are zero.

When one student's mark is entered, the loop is executed once only and *studentsProcessed* is one.

That completes the design process.

Incidentally, we could replace

```
studentsPassed = studentsPassed + 1;
```

with

```
studentsPassed++;
```

++ is known as the increment operator; it increases the value stored in an integer variable by one.

5.6 Documentation

Statements to be repeatedly executed are indented by two character spaces under the while keyword. A block of statements to be repeatedly executed is enclosed within braces that could be positioned as shown below.

while (boolean-expression) {
* statement-1;*
* statement-2;*
* ...*
* statement-n;*
}

5.7 Programming Principles

Use a loop whenever a sequence of statements is to be repeatedly executed.

Remember to use a function such as *doubleEqual* if you must use *double* (or *float*) values in Boolean expressions which control the execution of a loop; equality of values of type *float* or *double* is not always easy to determine - see chapter four for example.

Always provide simple validation of user input. Remember that it is up to you, the programmer, to ensure that valid values are passed to function parameters no matter what the user does. A value is valid if it falls between some pre-determined limits.

Your loops should execute the required number of times exactly, not one more, not one less. When testing a loop, consider choosing data values which cause the loop

(a) not to be executed at all
(b) to be executed once only
(c) to be executed the maximum number of times.

Exercise 5.1

1 Design a loop which will print a horizontal line of ten asterisks. Then test your design in a simple program.

2 Design a loop which will print the thirteen times table up to 12 x 13. Then write a program to test your design.

3 Write and test a function which will input a person's age in years. If the age input is less than zero or more than 110 then output the message "Don't be silly!" and invite the user to input a person's age again; this is to be repeated until the age input lies between 0 and 110. The function is to return an age between 0 and 110 inclusive.

4 Write a program which will input the wages for an unspecified number of employees and output the total wage bill, the number of employees whose wage has been input and the average wage. If the number of employees is zero, your program should display a suitable message if the number of employees is zero and not calculate the average wage (why?).

5 A waste removal company offers skips for hire on a daily or weekly basis. The weekly charge is six times the daily charge. Write a program which will input the daily charge and output a table of charges in the following format:

Time Period	Charge
1 day	
2 days	
...	
6 days	
1 week	
...	
4 weeks	

6 The value of a computer system depreciates by 50% of its current value at the beginning of each year of its life. For example, suppose a system was purchased for £1000, then immediately its value would be £500. After twelve months, its value would be £250 pounds and one year after that its value would be £125. Write a program which will

(a) input the purchase price of a computer system
(b) write a table showing its value at the beginning of each year of ownership for five years.
(c) output the first year in which its value is less than 20% of its purchase price.