COMP2113 Programming Technologies / ENGG1340 Computer Programming II

Module 10. C programming (Part 1) – printf() and scanf()

Objectives

You should have learned some C basics in Module 6. 😊 This set of self-learning notes is intended to refresh your memory on those C basics while supplementing more information.

At the end of this self-learning lab, you should be able to:

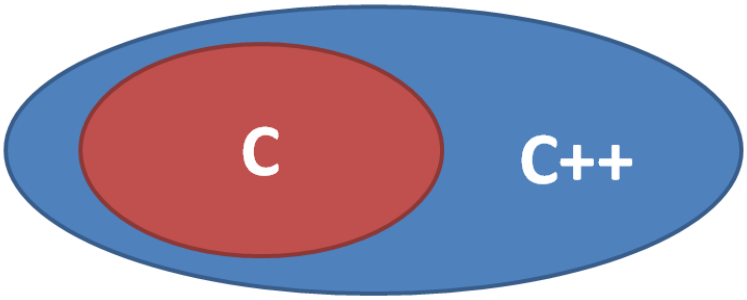
- Recall some basic differences between C and C++.
- Recall how to write simple C programs that can get user input and display output on screen with proper format.

Section 1. Background

- The C programming language is a predecessor of C++. We learn this older language mainly for two reasons.
  - C is a more low-level language and less functionality is provided. E.g., there is no classes or STL. In return, **the programs are usually faster than their C++ counterpart**.
  - A lot of existing systems like Linux and Unix use C extensively. For the operating systems and networking area, many libraries are available in C only.
- This chapter gives a brief introduction to C. You will get familiar with it once you study the courses **operating systems** and **networking**.

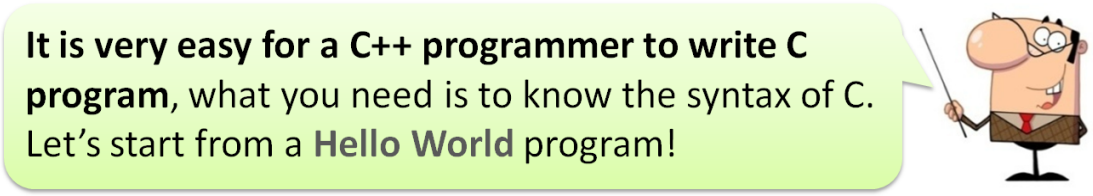
Some history

- The C programming language was designed by Dennis Ritchie at Bell Laboratories in the early 1970s.
- The C language was standardized in 1989 by ANSI (American National Standards Institute) known as **ANSI C**.
- C programming language became International standard (ISO) in 1990 which was adopted by ANSI and is known as *C89.*
- As part of the normal evolution process the standard was updated in 1995 (*C95*) and 1999 (*C99*).



- C++ extends C to include support for Object Oriented Programming and other features that facilitate large software development projects.
    - In other word, a valid C program is also a valid C++ program.
    - **However, the reverse is not true**, because some functionality used in C++ is not available in C.

Section 2. Hello World



- Let's browse to `~/module10/`

```
$ cd module10
```

- Consider the file `hello.c`

```
$ vi hello.c
```

```
//Filename: hello.c
1 #include <stdio.h>
int main(){
2   printf( "Hello world!\n" );
   return 0;
}
```

Code explanations:

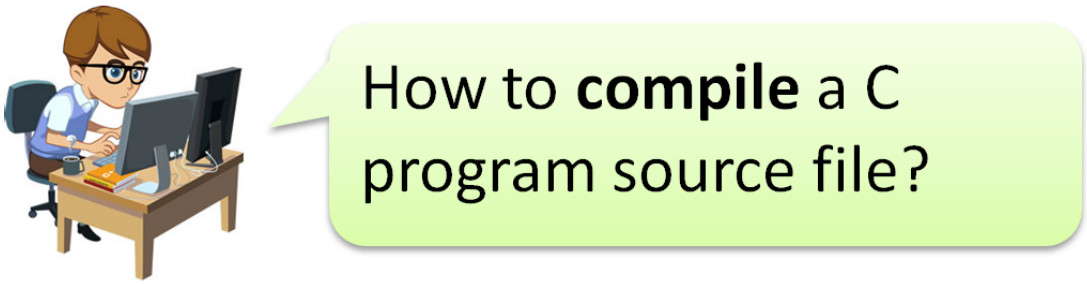
- First of all, we usually name the source code of a C program with extension "`.c`".
- **Commenting** is the same as C++ (i.e., you can use `//` and `/* */`) for marking comments in a C program.

1. The preprocessing directive `#include <stdio.h>` stands for standard I/O. `<iostream>` is not present in C.

- **NO namespace in C:** We do not need to use the `namespace std`, since there is no namespace in C.

2. **`printf()` instead of `cout`** - In C we use the `printf()` function to print a string. Objects like `cout`, `cin` and `endl`, which are part of the `<iostream>` library, are not available in C.

- As there is no `endl`, we use the escape character "`\n`" to signify the newline in the output string.



- To compile the program, use the `gcc` command. It will compile the source file `hello.c` and generate the executable named `hello`.

- As a remark, any C program is also a C++ program, so you can also compile a C program using the g++ compiler.

- Let's try to compile the program using `gcc`.

```
$ gcc hello.c -o hello
```

- Run the code, simply the same as running a C++ executable.

```
$ ./hello
Hello World!
```

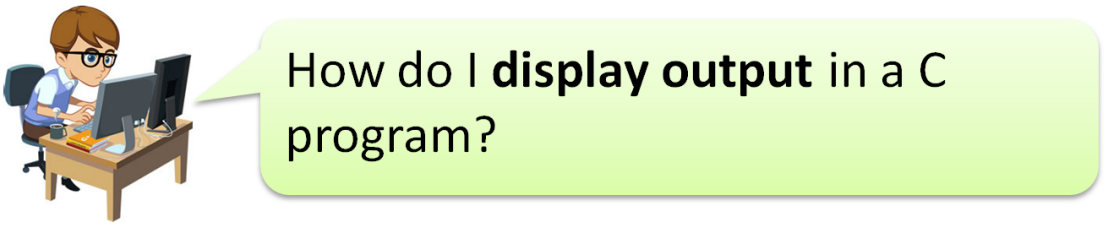
- For illustration purpose, let's try to compile the program using `g++`.

```
$ g++ hello.c -o hello
```

- Run the code, it should work as expected ☺. Easy ☺

```
$ ./hello
Hello World!
```

Section 3. printf() for output



- As we have seen, printing a string can be done by `printf()` and passing a string literal as the input parameter. The definition of the `printf()` function is included in the `<stdio.h>` header file.

```
printf( "Hello world!\n" );
```

- To print the value of a **variable**, we include the corresponding **conversion specifiers** inside the string literal and supply the variables inside the `printf()` function.
- As an example, the following function prints the value of the `int` variable `a` on screen.

```
int a = 10;
printf( "The value of a is %d.\n", a );
```

- The `%d` is called the conversion specifier for integer values.
- The value of the variable `a` (an `int` variable) will be inserted in the conversion specifier `%d` in the string literal. Therefore the output is "`The value of a is 10.`"
- The following table shows some popular conversion specifiers.

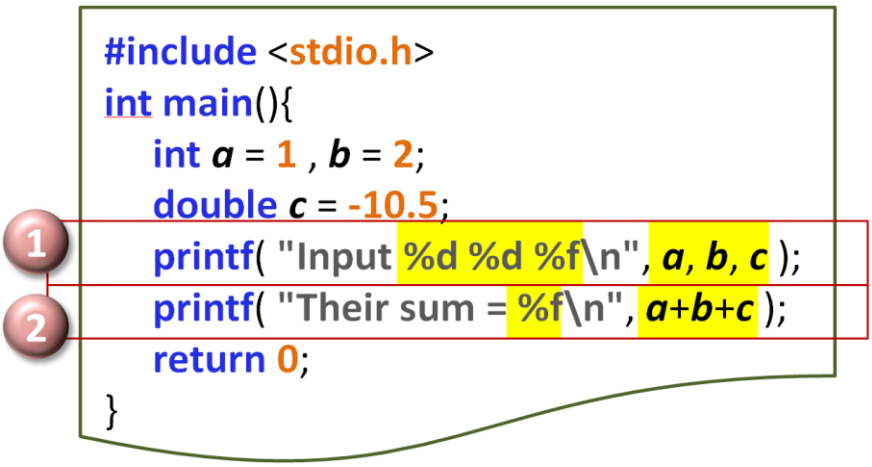
Variable type	Conversion specifier
`int`	`%d`
`float`	`%f`
`double`	`%f`
`char`	`%c`
`String`	`%s`
Removing trailing zeros for `float` and `double`	`%g`

- Please note that the table is not a complete list of all conversion specifiers, please refer to the following URL if you want to output variables of various types.

  http://www.cplusplus.com/reference/cstdio/scanf/

- Let's consider `output1.c` as an illustration of the use of `printf()`.

```
$ vi output1.c
```



Code explanations:

- Reminded that the preprocessing directive that include input and output handling in C is `#include <stdio.h>` (NOT `<iostream>`). And there is no namespace in C.

1 **Multiple conversion specifiers printed in order**

- In this example we will print "`Input 1 2 -10.500000`" . It is because we have three conversion specifiers in order : `%d` `%d` and `%f`, and in the value part we have `a`, `b`,`c`:
  - The value of `int` variable `a` will be printed in the position of the first specifier `%d`.
  - The value of `int` variable `b` will be printed in the position of the second specifier `%d`.
  - The value of `double` variable `c` will be printed in the position of the third specifier `%f`.
- From this example we learn that we can include more than one specifiers inside the string literal, and the specifiers may appear in the middle of the string literal.

2 **We can provide an expression in the value part, but the conversion specifier has to match the result of the expression.**

- In this case the expression is `a+b+c`, which is `int + int + double`, the result will be a `double` value, therefore we use the `%f` conversion specifier to print it in the output string.
- Let's try to compile the program and run it.

```
$ gcc output1.c -o output1
$ ./output1
1  2  -10.500000
Their sum = -7.500000
```

- Note that `%f` prints 6 digits after the decimal place; we will see how we can remove those trailing zeros later.

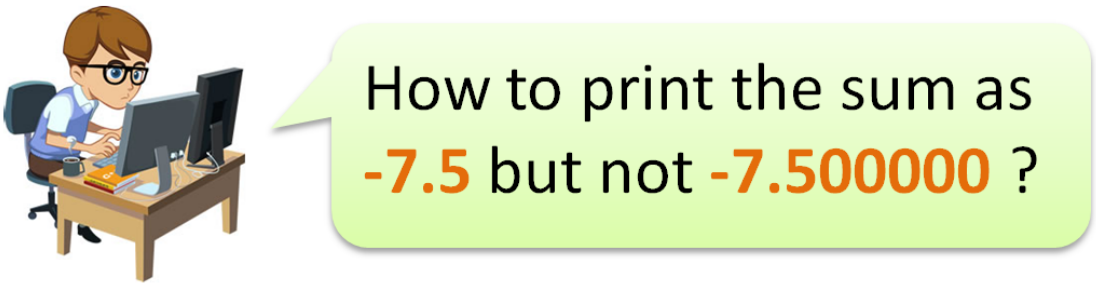
- What happen if we use the wrong conversion specifier? Let's try to see it if we change the 3rd conversion specifier to `%d` (an integer specifier), which is used to display a `double` value (Note: `c` is a `double` type variable).

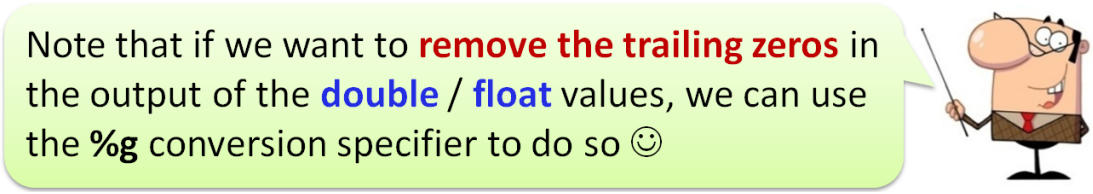
```
printf( "Input %d %d %d\n", a, b, c );
```

- Let's try to compile the program and run it again. What will be display?

```
$ gcc output1.c -o output1
$ ./output1
1  2  0
Their sum = -7.500000
```

- As you notice, C is pretty primitive in outputing values on screen, and it is important to use the conversion specifier that matches the data type in C to display the right value.





- Let's try to update the conversion specifier for the sum of `a,b,c` from `%f` to `%g` to see the effect.

```
printf( "Input %d %d %g\n", a, b, c );
printf( "Their sum = %g\n", a+b+c );
```

- Compile the program and run it. What will be displayed?

```
$ gcc output1.c -o output1
$ ./output1
1  2  -10.5
Their sum = -7.5
```

- Note that the conversion specifier does not change the value of the variable. It just indicates how the data stored in the variable should be displayed.

Section 4. No string class in C

- Note that C does not have a string class ☹. A string in C is simply an array of `char`.
- Consider the file `output2.c` that shows the printing of the content of a `char` array.

```
$ vi output2.c
```

```
#include <stdio.h>
int main(){
    char name[] = "Alan";
    int age = 20;
    float weight = 60.5;
    printf( "%s is %d years old and weights %g kg\n",
            name, age, weight );
}
```

Code explanations:

1. **There is no string class in C, we can only use an array of `char` to represent string values.**
   - In this example we define `name` as an `array` of `char`.
   - Same as C++, when we initialize the `array` variable in its declaration, we can choose not to specify the length of the `array`. After the line `char name[] = "Alan";` we have the following content for the `name` array.

char name[i]	A	l	a	n	\0
i	0	1	2	3	4

   - Note that "`\0`" is the **null character** that signify **the end of the string**.
   - We will have more discussions on C-string in the next part.
2. The `%s` conversion specifier means to print a string value of a `char` array, therefore we can supply a `char` array `name` to match `%s`.
   - The `printf()` function prints the array content of the `name` variable, and **when it reaches "`\0`" it understands that is the end of the string and stops outputting.**

- Let's try to compile the program and run it.

```
$ gcc output2.c -o output2
$ ./output2
Alan is 20 years old and weights 60.5 kg
```

- The `printf()` function supports a number of advanced features.
  - For example, you can specify the width in which the data is printed by inserting an integer in the middle of the specifier.
  - If the width is larger than the actual length of the variable, the variable will be printed in a **right-justified manner**.
- Consider the file `output3.c` as an illustration.

```
$ vi output3.c
```

```
#include <stdio.h>
int main(){
   char p1[] = "Alan", p2[] = "Ben";
   int a1 = 9, a2 = 30;
   float w1 = 30, w2 = 40.25;
   printf( "12345678901234567890123456789\n");
   printf( "%8s %8s %8s\n", "Name", "Age", "Weight");
   printf( "%?? %?? %??\n", p1, a1, w1);
   printf( "%?? %?? %??\n", p2, a2, w2);
}
```

Code explanations:

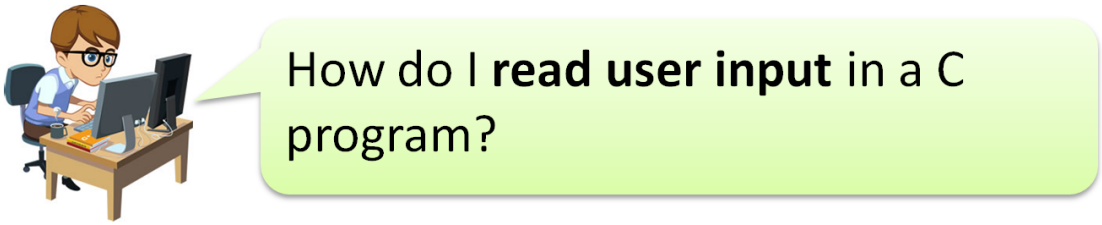
1. We define two `char` arrays `p1` and `p2`, and initialize the two arrays's values as "`Alan`" and "`Ben`", respectively.
2. We print the header row, since the name of the three columns are three string values (i.e., "`Name`", "`Age`" and "`Weight`"), we use "`%s %s %s`" as the conversion specifier.
- We use "`%8s %8s %8s`" to format the width of the column, i.e., each column will be of width of 8 characters. Therefore the first column will have the `Name` printed right-justified in the column of width 8 characters.

Question: What should be the conversion specifiers in 3 so that the data of `Alan` and `Ben` will be printed in columns with 8 characters width and in right-justified manner?

- Compile and run `output3.c`

```
$ gcc output3.c -o output3
$ ./output3
12345678901234567890123456789
    Name       Age    Weight
    Alan         9        30
     Ben        30     40.25
```

- **Answer:** "`%8s %8d %8f`". Please try.
- Note that the values printed are right justified in the column of 8-characters wide (the extra character in between each column is due to the space between two specifiers.).



- C provides the `scanf()` function to read an input.
- We need to use the conversion specifiers to indicate the type of variable we are reading.
- **Then we provide the addresses of the variables that will store the value.**
- Consider the file `input1.c` as an illustration.

```
$ vi input1.c
```

Code explanations:

```c
#include <stdio.h>
int main(){
  int a;
  float b;
  printf( "Enter an int and a float: " );
  scanf( "%d%f", &a, &b );
  printf( "Their product = %g\n", a*b );
}
```

1. The `scanf()` function is used to get user input (just like `cin` in C++). However the use of the `scanf()` function is a bit more complicated than `cin`.
   - The conversion specifiers in `scanf()` are "`%d%f`", which means that we will first read in an integer value, and then read in a float value. The two values will be stored in variables `a` and `b` respectively.
   - With this sequence of specifiers, the first input variable must be an `int` type variable, and the second variable must be a `float` type variable. The order has to be matched.
   - Note that there is no space between the conversion specifiers inside `scanf()`.
   - Reminded that we need to pass in the **addresses of the variables** `a` and `b` to `scanf()`, that is **`&a`** and **`&b`**, respectively. By passing the address it act likes **pass by reference** so that the value read from user input is updated to the variables `a` and `b` that we pass in.

   > **Important!** This is the C-style pass by reference technique, which we have to pass in the address of the memory cell that contain the value of the variable using the `&` operator when we call the `scanf()` function.

2. We use `%g` conversion specifier to output the result of `a*b` to remove the trailing zeros.

- Compile and run `input1.c`

```
$ gcc input1.c -o input1
$ ./input1
```

```
Enter an int and a float: 11 12.5
Their product = 137.5
```

How about reading string value into a **char** array variable in a C program?

- There are many alternatives to read in string into a `char` array in C.
- In the following simplest approach, we define a `char` array variable that is large enough to store the string value, and use `scanf()` to read in user input into the `char` array.
- Consider the file `input2.c` as an illustration.

```
$ vi input2.c
```

```
#include <stdio.h>
int main(){
  char name[100];
  printf( "What is your name? " );
  scanf( "%s", name);
  printf( "Hello %s!", name );
}
```

Code explanations:

1. We define name as an array of `char`, the array contains 100 slots (Like C++, we must provide the size of the array if we do not initialize the array's content in the declaration).
2. We use the `%s` conversion specifier to indicate we are reading a string value into the `char` array, `scanf()` will append a **null character** "`\0`" at the end of the string.

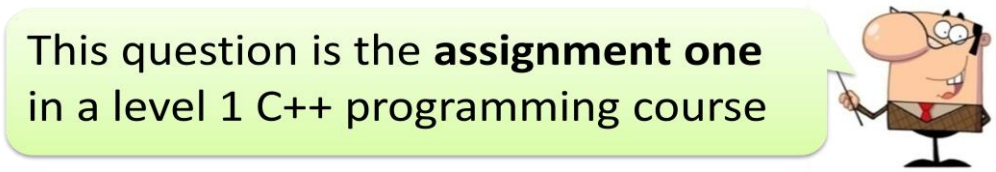
- **Important technical note:** Note that the second parameter to `scanf()` is `name` but not `&name` because `name` is an array. `name` is actually a pointer that stores the **address of the first slot** of the `char` array. Therefore we do not need to use the address-of operator "`&`" when printing an array with `scanf()`.

  Yes it is pretty confusing but this is the syntax requirement of C programming language :P

- Compile and run `input2.c`

```
$ gcc input2.c -o input2
$ ./input2
What is your name? Chim
Hello Chim!
```

Checkpoint 10.1 **(Please submit your answer to Moodle.)**



Write a C program that determines some of the results of an election using the largest remainder method.

The **largest remainder method** is used in Hong Kong Legislative Council Election.

1. Candidates contest the election in the form of **lists**, and voters are required to select one candidate among all the participating lists.
2. The number of **votes** for each list is divided by a **quota** representing the number of votes required for a seat. The result of the division consists of an integer part and a fractional remainder.
3. Each list is first allocated a number of **seats** (called automatic seats) equal to the integer. When there are any seats unallocated, the lists with largest remainder will be allocated one additional seat.

Here's an example:

Suppose that there are 6 **seats** and 900,000 **votes** in total. To get one seat, a **quota** of 150,000 votes (= 900000/6) is needed. If there are 8 **lists** with the following distribution of votes, the results of the election are shown in the last two rows.

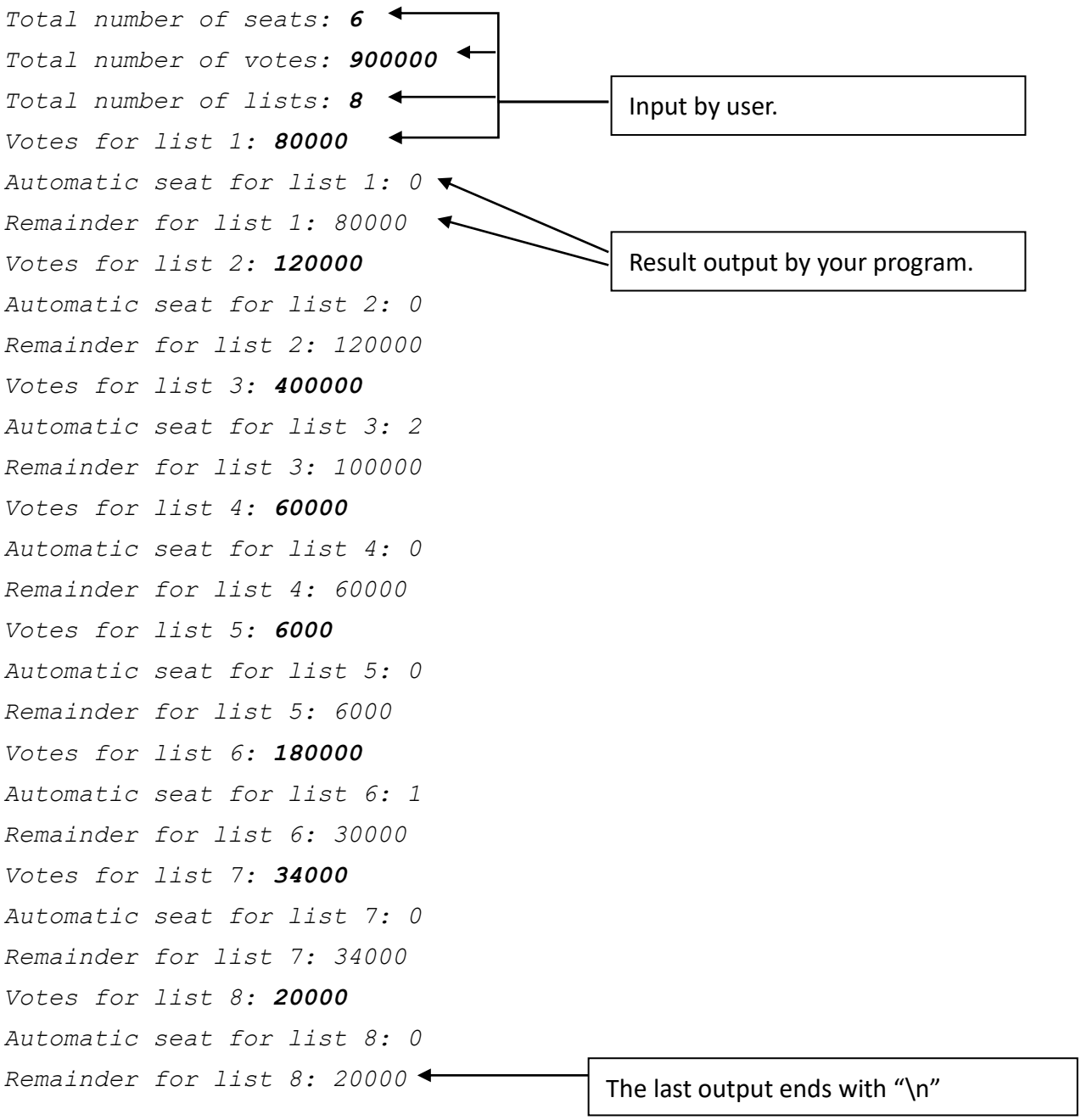
List	1	2	3	4	5	6	7	8
Votes	80,000	120,000	400,000	60,000	6,000	180,000	34,000	20,000
Automatic Seats	0	0	2	0	0	1	0	0
Remainder	80,000	120,000	100,000	60,000	6,000	30,000	34,000	20,000

You are only required to determine two things:

1. Number of seats automatically obtained by each list.
2. The remainder after deducting the votes required for the automatic seats.

Please note that you do not need to determine the additional seats obtained by the lists.

The following is a sample run. (**Bolded text** is the user input; *italic text* is the output of your program.)



- Of course you are not required to solve this problem again from scratch ☺ Please look at the C++ solution to this problem.

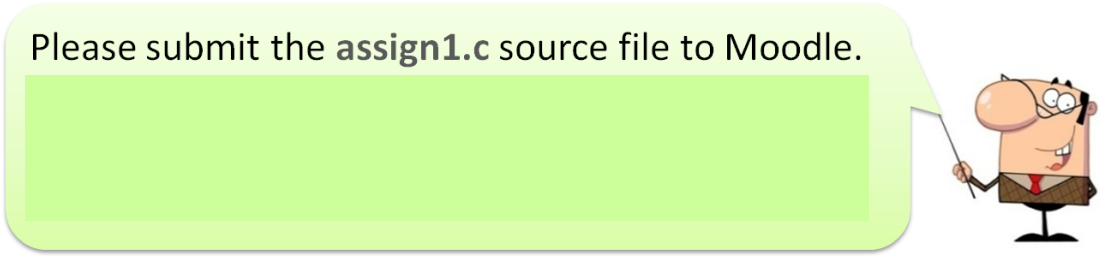
```
$ vi assign1.cpp
$ g++ assign1.cpp –o assign1_cpp
$ ./assign1_cpp
(You can test the code, it is error free)
```

- Your task is to create a C implementation of the `assign1.cpp`.

```
$ cp assign1.cpp  assign1.c
$ vi assign1.c
```

- Compile and run `assign1.c`

```
$ gcc assign1.c –o assign1
$ ./assign1
(Test the code against the test case in the previous page)
```



References

- *Application programming in ANSI c (3rd edition)*, by Richard Johnsonbaugh & Martin Kalin. Prentice Hall.
- *Data Structures and Algorithm Analysis in C(2nd edition)*, by Mark Allen Weiss. Addison Wesley.
- Similer to C++, `scanf()` will stop reading once a space or newline is encountered.
- To read in a line of string incluing space into a variable, we need to use the `getline()or fgets()` function. Please refer to the following references if you want to learn more about those functions.
- Tutorial on `getline()`
  http://crasseux.com/books/ctutorial/getline.html
- Cprogramming.com
  http://www.cprogramming.com/tutorial.html
  http://www.cprogramming.com/tutorial/c/lesson9.html