

CSE 1301 PRACTICAL SESSION 2 & 3

Creating Simple Algorithms and Starting to Code (12 marks)

This practical session aims to help you develop algorithms and introduces you to the first steps in writing a C program. Your program here will read, perhaps do a simple calculation and write. Complete Part 1 before the class.

PART 1 (to be completed before class)

(5 marks)

For parts (b) and (c) of this question, you should show evidence of top-down design. Remember, for top-down design you write down what you have to do, and break this down into 3-7 smaller steps and then break each step down into 3-7 smaller steps. This process continues until each step is a single instruction.

- (a) Write an algorithm for choosing clothes to wear for different activities involving different weather conditions (e.g. going to the beach when it is hot, skiing in the snow, etc.). The algorithm should make it clear how the choice of clothes depends on the weather and the activity, and should be able to cope with a range of weather conditions. Hint: use selection.
- (b) Write an algorithm for navigating down a corridor littered with obstacles. Use iteration.
- (c) Suppose that you have been given a \$100 music store gift voucher. Write an algorithm for buying some CDs with the voucher. Use selection and iteration.
- (d) Write an algorithm that converts temperature in the Celsius scale to the Fahrenheit scale. The two scales are related by:

$$F = 9/5(C + 32)$$

The temperature in Celsius is the input and the temperature in Fahrenheit is the output.

- (e) Study the ASCII table (see: King, Appendix E, p. 637 or Deitel & Deitel, Appendix D, p. 891). What is the relationship between the ASCII code of a lower case character and that of its upper case equivalent?
(For the curious: are there any other natural relationships in the table, e.g. between numbers and the characters that share their keys (e.g., 1 and !, 2 and @, etc), or between ASCII code and keyboard position?)

PART 2

(2 marks)

- (a) Execute the algorithm Starting Borland C

while (you are not logged in)

```
{
  if (PC is off)
  {
    turn PC on
  }
  login to your account
  click on Start
  In Start menu, click on
```

Start > Programs > Development > Borland C++ 5.02 > Borland C++

```
}
```

- (b) This algorithm is unlikely to be perfect. There may be situations where it does not specify clearly enough what is to be done, or where following it *literally* causes a problem. As you execute it, note one way in which the algorithm could be improved, to make it more precise, or clearer, or complete. Write down your improvement.

(c) Familiarisation

- Once Borland C is running, familiarize yourself with the menu options and the control buttons (icons) on the top part of the window. If you place the mouse pointer over an icon for a second or two, a description of the purpose of that button will appear. A more verbose explanation of what the button does also appears at the bottom of the screen.
- You will also notice that one of the items in the menu is Help. You can find useful information about the programming tools available to you in this programming environment. Browse through the topics available, so you know where to look in case you need further information about using Borland C for programming.

(d) Create a New File

- Create a new file by selecting **File > New > Text Edit**. Type in the program below, and save it as PART1.C. (Note: **File > Open** will let you open an existing file.)

```
#include <stdio.h> /* Standard header file contains function printf */

/*
** Program:  PRAC01/PART1.C
**
** This program takes the integers 23 and 7 and applies
** the operation of addition, subtraction, multiplication,
** division and remainder to them, in turn.
*/

int main()
{
    int x, y;

    x = 23 ;
    y = 7 ;

    printf("x + y = %d\n", x + y);
    printf("x - y = %d\n", x - y);
    printf("x * y = %d\n", x * y);
    printf("x / y = %d\n", x / y);
    printf("x %% y = %d\n", x % y);

    return 0; /* returns 0 at end of program (ignore this at present) */
}
```

- (e) **Predict** what your program will print, when it is successfully compiled and run.

(f) **Compile, Link and Run**

- One way to do all three steps at once---Compile, Link and Run---is to click on the lightning icon (**Run**). (Note that the compile-and-link steps are also referred to as “make” or “build.”)
- If there are errors in your program, a Message window will appear. Take note of the **error** and **warning** messages, as they can help you spot the problem. For example, suppose you forget a semicolon, you will see an error like this:

```
!    PART1.C    (27,9):    Statement    missing    ;
```

The error message tells you that in file PART1.C, line 27, the compiler thinks you may have forgotten a semicolon in the previous line. Remember the errors you encounter, and try to avoid making the same mistake. Take note that **C is case-sensitive**; that is, `Main()` is not the same as `main()`. Many errors are caused by such typos, such as typing `scanf` instead of `scanf`. The most common errors are called “parse errors,” which usually occur because of typos, such as missing brackets, semi-colons in the wrong place, forgotten quotations, and so on. The compiler is quite meticulous when it comes to those little things.

- To go back to editing your program and correct the problem, you may need to click on the **Windows** menu, and then select the file you were editing.
- If there are no **errors** (a.k.a. “bugs”) or **warnings** during compilation and linking, your program will run (or “execute”) right away. You’ll see a black window (or “console”) flash the output briefly, but then it will disappear before you can read anything, leaving you back at the message or program window.
- To put a “breakpoint” on the program so you can see the output: Position the cursor on the line `return 0;` and then click the **Toggle Breakpoint** icon at the top part of the window. You’ll see that the whole line is highlighted in red. When you run the program, the console appears, and then goes in the background. To show the console, select the item in the “taskbar” (that’s the bar at the bottom of the screen) with MS-DOS and PART1.EXE on it. (Note: You can also toggle breakpoints at other parts of the program, wherever you want the program’s execution to pause.) Unfortunately, breakpoints are not reliable and the alternative is to run the program from command line- refer to Practical Session 1 for this method.

(g) **Compare** your earlier predictions with what actually happened. Make sure you understand why the program printed the numbers it did.

(h) **Change from** `ints` to `floats`.

The above program works only with `ints`. Now we will see what happens with `floats`.

- Copy the above program into another file, and edit that file as follows. Firstly, change the `int` declaration to `float`, so variables `x` and `y` are now of type `float`. Secondly, for each `%d` inside a `printf` statement, change it to `%f`. This enables `printf` to print out `floats` instead of `ints`.
- Predict what will happen, again.

- Compile the program.
- What happens, and why?
- Delete the offending line. Compile, link, run again. How do the results differ from those you obtained when the variables were integers? Why?
- Should you have made other changes when you changed the program to deal with `floats`, above?

PART 3**(5 marks)**

Code your algorithm which converts temperature in the Celsius scale to the Fahrenheit scale from 1(d).

- (f) Firstly, use `int` for all your variables and numbers. Predict what effect this will have on the result. Then compile, link and run to see what happens.
- (g) Then use `float` for everything. Again, predict what will happen, then see what does happen.
- (h) Try declaring your variable for Fahrenheit as a `const`, and explain what happens.
- (i) What would be the effect of leaving brackets out of the expression you used in your program to do this calculation?
- (j) Now try making just one of the numbers in the formula a `float`, and all the rest `int`. What happens? Will the result depend on which of the number you make a float?

Submission:

PART 1: Show written preparation work to demonstrator at start of class for marking . If not done before class, it will not receive any marks, however you should still show it to your demonstrator during class to check your understanding.

PART 2: To be marked by the demonstrator during this class. No late submission.

PART 3: Attempt to complete this part during the two weeks allocated to practical session 2. If not marked in these two weeks, you are permitted to have this part marked at the start of practical session 4.

Advanced Component (2 BONUS MARK)

On a computer screen, the *pixels* (points) are given coordinates. The upper-left corner pixel is (0,0). The horizontal coordinate increases as you go from left to right. The vertical coordinate increases as you go from the top to the bottom. (This is upside-down from the Cartesian coordinate system you learned in geometry.)

Two squares are drawn on the screen. Both are positioned so that all sides are aligned with an axis. The first one has upper-left coordinate (x_1, y_1) and each side has a length of r_1 pixels, and the second square has upper-left coordinate (x_2, y_2) and each side has a length of r_2 pixels. Write a C program to input these 6 numbers (all integers), and determine if the squares overlap or not.