The University of Nottingham Ningbo China

SCHOOL OF COMPUTER SCIENCE

A LEVEL 1 MODULE, AUTUMN SEMESTER 2017-2018

PROGRAMMING AND ALGORITHMS

Time allowed: ONE hour

Candidates may complete the front cover of their answer book and sign their desk card but must NOT write anything else until the start of the examination period is announced.

Answer ALL questions.

No calculators are permitted in this examination.

Dictionaries are not allowed with one exception. Those whose first language is not English may use a standard translation dictionary to translate between that language and English provided that neither language is the subject of this examination. Subject specific translation dictionaries are not permitted.

No electronic devices capable of storing and retrieving text, including electronic dictionaries, may be used.

DO NOT turn examination paper over until instructed to do so.

ADDITIONAL MATERIAL: None.

INFORMATION FOR INVIGILATORS: Collect both the exam papers and the answer booklets at the end of exam.

1) Write a declaration in C for a type which contains a student ID number, and the student's final mark for each of 6 modules they study this year (whole numbers).

(3)

2) A common way to declare the `main` function is `int main(int argc, char *argv[])`. Explain what the arguments `argc` and `argv` represent.

(3)

3) Write a function which takes an integer (greater or equal to 2) as a parameter and returns the number of integer *factors* that the number has. A *factor* is a number (greater than one) which divides the other number with no remainder, without including 1 or itself. For example, 16 has 3 factors (2, 4, 8), 9 has 1 factor (3), but a prime number like 7 has 0 factors.
The function should return the number of factors, or -1 if the parameter is not greater or equal to 2.

(4)

4) In general, can you search a *sorted singly-linked list* for a value as fast as a *binary search tree*? If so, explain how. If not, explain why.

(2)

5) A programmer has written a program in C that asks the user to enter a whole number between 6 and 14 (inclusive of both). The program reads the user's input as a string then converts it to an integer. *Write the different string values* you would use to test the program works correctly and *justify why you chose them.* Choose your values intelligently and write only as many as you think you require.

(3)

6) The following function takes an integer array and the length of that array. It should calculate the sum of that array and return the computed value. Identify and explain the *two* errors in this function and explain how to fix them.

```
#include <stdio.h>
#include <stdlib.h>
int sum(int nums[], int nums_size)
{
   int *total;

   if((total = (int *) malloc(sizeof(int))) == NULL)
   {
      exit(1);
   }

   for(int i = 0 ; i < nums_size; i++)
   {
      *total += nums[i];
   }
   return *total;
}
```

(3)

7) Functions can accept a variable number of arguments using the Variable Length Arguments feature of C, however, there is no built-in way of finding out how many arguments have been passed. Describe *two* different ways in which you can design your function to know how many arguments have been passed.

(2)

8) A common mistake in C is to use a single equals (`=`) instead of a double equals (`==`) when intending to compare two integer values. Write a short example (few lines) of this mistake and describe why this mistake is still a valid C program.

(3)

9) C has several standard library functions which have dangerous flaws. Describe the flaw in each of the following functions that makes it easy to use incorrectly:

(a) `char *strcpy(char *src, const char *dest)` — copy the source string to the given destination buffer.

(2)

(b) `int atoi(const char *nptr)` — convert the initial portion of nptr to an integer value.

(2)

10) Write the output of the following program:

```
#include <stdio.h>
int f1(int x, int y)
{
   x = x + 2;
   y = y + 3;
   return x + y;
}
int f2(int *x, int y)
{
   *x = *x + 2;
   y = y + 3;
   return *x + y;
}
int f3(int *x, int *y)
{
   *x = *x + 2;
   *y = *y + 3;
   return *x + *y;
}
int main(int argc, char *argv[])
{
   int k = 3, m = 5, r = 0;
   printf("1) %d %d %d \n", k, m, r);
   r = f1(k, m);
   printf("2) %d %d %d \n", k, m, r);
   r = f2(&k, m);
   printf("3) %d %d %d \n", k, m, r);
   r = f3(&k, &m);
   printf("4) %d %d %d \n", k, m, r);
   return 0;
}
```

(3)