University of Glasgow

Monday 16 May 2022
14:00-15:30 BST
Duration: 1 hour 30 minutes
Additional time: 30 minutes
Timed exam – fixed start time

DEGREES of MSci, MEng, BEng, BSc, MA and MA (Social Sciences)

Systems Programming H

COMPSCI 4081

(Answer All Questions)

This examination paper is an open book, online assessment and is worth a total of 60 marks.

Programming Question Advice: Throughout this paper you are strongly recommended to outline the program fragments rather than attempting to produce fully working code. Producing correct compiling code will simply take too long. Minor syntax and logic errors will not be penalised.

Q1 C Programming and Data Types DONE

(a) Consider the following table of data types and corresponding bytes in memory.

Data Type	Size
char	1 byte
int	4 bytes
float	4 bytes
double	8 bytes
pointer	8 bytes

i. Which data type will you use to store the value 10.1 and how many bytes will it require in memory? Give an example variable declaration.

[1]

ii. Which data type will you use to store the value '1' and how many bytes will it require in memory? Give an example variable declaration.

[1]

iii. Which data type will you use to store the value 255.2 and how many bytes will it require in memory? Give an example variable declaration and an explanation for your choice.

[2]

iv. Consider the following struct definition and use. How many bytes in memory will the newPersonID variable occupy? Explain how you calculated your answer.

```
struct personID {
    const char * name;
    int date;
};
struct personID newPersonID = {"Happy Coding",2022};
```

[2]

(b) Consider the following C program fragment where ___ denotes a gap you will need to fill. Consider the **Programming Question Advice** as you answer this question.

```
1.  struct A { int val; struct A * next; };

2.  struct A * newL(int v) {
3.      struct A * l;
4.      l = (struct A *)malloc(sizeof(struct A));
5.      l->val = v;
6.      l->next = NULL;
7.      return l; }

8.  addL(struct A * l, int v) {
9.      struct A * li;
10.     struct node * l1;
11.     li= newL(v);
12.     if(___)
13.         ___
14.     while(l1->next!=NULL) l1 = l1->next;
15.     l1->next = li;
16.     return ___;
17. }

18. int main (){
19.     struct A *lex;
20.     ___
21.     addL(lex,1);
22.     addL(lex,2);
23.     addL(lex,3);
24.     printf("%d %d %d\n",lex->val,lex->next->val,lex->next->next->val);
25.     ___
26. }
```

i. This fragment has memory management issues relating to pointers. Identify two issues and add **a single line code** at each of the lines 20 and 25 to correct them; you can implement a helper function. **[4]**

ii. In lines 12, 13, and 16 of the fragment, the handling of return values is missing. Identify what checks should be performed and how these should be handled to ensure correct operation. Add the appropriate code to lines 12 and 13. **[2]**

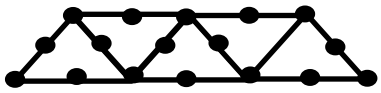
iii. Explain what happens on lines 18-26 of the fragment, referring to lines 2-7 and 8-17 as appropriate. Illustrate your answer with an appropriate diagram to show how lex changes. You may draw the diagram using any convenient tool, e.g. a drawing tool, or insert a picture taken of a hand-drawn diagram. **[8]**

TOTAL MARKS [20]

Q2 Memory and Resource Management & Ownership

(a) The Navier-Stokes equations allow for the simulation of fluid dynamics a well-known field of computationally intensive problems. A snippet of the implementation of the Nodal Data calculation is given below.

The fluid being simulated as a mesh of points connected via triangles as shown in the following diagram:



For each node the area of the sum of forces in each of the two axes x and y must be calculated, in a compute-intensive function that needs to be implemented efficiently. The following snippet of code shows a 2D implementation of this sum being calculated for one node with id `Nod=1`. The arguments of this function are the `ShareCount` of the nodes in a `Node[]` array of `struct` elements that correspond to each node in the above diagram.

```
1. for(Count = 0;Count <Node[Nod].ShareCount; Count++){
2.      SumXiByDi = SumXiByDi +
3.         (CDO[Node[Nod].ShareElements[Count]].CCVel[0] /
4.          Node[Nod].CC2NodeDist[Count]);
5.      SumYiByDi = SumYiByDi +
6.          (CDO[Node[Nod].ShareElements[Count]].CCVel[1] /
7.           Node[Nod].CC2NodeDist[Count]);
8.      SumTiByDi = SumTiByDi + (CDO[Node[Nod].ShareElements[Count]].CCT /
9.           Node[Nod].CC2NodeDist[Count]);
10.     Sum1ByDi = Sum1ByDi + (1/Node[Nod].CC2NodeDist[Count]);
11.}
```

i. Assuming the **stack** is empty at the start of the loop, if `ShareCount` is set to 2 illustrate the state of the **stack** at the end of the loop with the use of the following table, adding or removing rows as appropriate. Pay attention to data that is packed together such as elements belonging to the same `struct`, and elements of an array. **[6]**

Stack	<first variable name>
	<second variable name>
	.
	<n^{th} variable name>

ii. Considering that data is stored in neighbouring addresses in memory (**spatial locality**) review the following altered version of the same code where the '.' notation has been replaced by the '->' notation. Which of the two versions do you think would be better for faster execution and why? **[6]**

```
1. for(Count = 0;Count <Node[Nod]->ShareCount; Count++){
2.      SumXiByDi = SumXiByDi +
3.         (CDO[Node[Nod]->ShareElements[Count]]->CCVel[0] /
4.          Node[Nod]->CC2NodeDist[Count]);
```

```
5.      SumYiByDi = SumYiByDi +
6.          (CDO[Node[Nod]->ShareElements[Count]]->CCVel[1] /
7.           Node[Nod].CC2NodeDist[Count]);
8.      SumTiByDi = SumTiByDi + (CDO[Node[Nod]->ShareElements[Count]]->CCT /
9.           Node[Nod]->CC2NodeDist[Count]);
10.     Sum1ByDi = Sum1ByDi + (1/Node[Nod]->CC2NodeDist[Count]);
11.}
```

(b) Considering the challenges of manual memory management, identify the errors in the following code segments. For each segment **name the challenge** and **explain** your answer. **[5]**

i.
```
void * ptr1 = malloc(sizeof(int));
void * ptr2 = ptr1;
free(ptr1); free(ptr2);
```

ii.
```
node * left_child = create_node(...);
node * root = create_node(..., left_child, ...);
free(left_child);
```

iii.
```
char * mem = (char*) malloc(...);
mem = (char*) malloc(...);
```

(c) The following code fragment can be used in C++ to open, access, and then close a database connection. However, this is not a full implementation and there are gaps denoted by ___ on lines 8 and 12. Fill the gaps using the C++ RAII ownership technique with a **single line of code**, considering the **Programming Question Advice** and explain the benefit of using RAII. **[3]**

```
1. DBConnection* db_open(const char* url);
2. void db_close(DBConnection* connection);
3.
4. struct db_handle {
5.   DBConnection * connection;
6.
7.   db_handle(const char* url) {
8.     ___
9.   }
10.
11.       ~db_handle() {
12.         ___
13.       }
14.     }
```

TOTAL MARKS [20]

Q3 C++ Threading

(a) Sam has written the following C function that is simply intended to count how many times it is called:

```
1    int c = 0;
2    int count_calls() {
3        c = c + 1;
4        return c;
5    }
```

All is well until Sam creates two threads and has both of these repeatedly call count_calls(). Sam finds that the number of calls counted is wrong.

i. Which line of code is the root cause of the wrong counting? *(1 number)* **[2]**

3

ii. Using references to the above lines of code, give an account of one scenario where wrong counting might occur. *(20-30 words)* **[2]**

iii. By the end of the program, will the number of calls counted be less than or more than the actual number of calls if they are not equal? *(2 words)*

[2]

iv. Explain your answer above. *(20-30 words)* **[2]**

(b) What synchronisation primitive can be used to avoid deadlocks from occurring? *(10-20 words)*

[2]

(c) Briefly explain one reason why concurrent programming can be particularly difficult to debug. *(10-20 words)* **[1]**

(d) Consider a program with a critical section that needs to be worked on by more than one thread at any time. Which synchronization primitive can be used to maintain concurrency management of this program? Explain your answer. *(10-40 words)* **[3]**

(e) In addition to mutexes and condition variables, C++ provides *futures* and *promises* as higher-level synchronisation abstractions. Briefly outline what futures and promises are. *(20-40 words)* **[2]**

(f) Describe how futures and promises simplify writing multi-threaded code compared with using mutexes and condition variables. *(20-40 words)* **[2]**

(g) The code snippet below shows a thread using a mutex and a condition variable to synchronise with another thread. Assuming that the implementation of the other thread is correct, what is wrong with the given code? *(10 words)* **[2]**

```
thread 1:
    pthread_mutex_lock(&mutex);
    if (!condition)
        pthread_cond_wait(&cond, &mutex);
    run_critical_code();
    pthread_mutex_unlock(&mutex);
```

TOTAL MARKS [20]