University of Glasgow

Monday 17 May 2021
Available from 09:30 BST
Expected Duration: 1 hour 30 minutes
Time Allowed: 3
Timed exam within 24 hours

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

Systems Programming H

COMPSCI 4081

Answer All 3 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

(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

i. Which data type will you use to store the decimal value 10 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 'C' 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 255.1234567 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 = {"John Doe",1999};
```

[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 * left, * right; };

2.  struct A * newL(int v) {
3.      struct A * l;
4.      l = (struct A *)malloc(sizeof(struct A));
```

```
5.       l->val = v;
6.       l->left = NULL;
7.       l->right = NULL;
8.       return l; }

9.   addL(struct A ** l,int v) {
10.      struct A * li;
11.      li= newL(v);
12.      if(___)
13.          ___
14.      else {
15.          (*l)->left = li;
16.          li->right = *l;
17.          *l = li;}
18.  }

19.  int main (){
20.      struct A *lex;
21.      ___
22.      addL(&lex,1);
23.      addL(&lex,2);
24.      addL(&lex,3);
25.      printf("%d %d %d\n",lex->val,lex->right->val,lex->right->right-
     >val);
26.      ___
27.  }
```

i. This fragment has memory management issues relating to pointers. Identify two issues and add single line code to lines 21 and 26 to correct them; you can implement a helper function.

[4]

ii. In lines 12 and 13 of the fragment the handling of `newL` 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 22-25 of the fragment, referring to lines 2-8 and 9-18 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 a picture taken of a hand-drawn diagram

[8]

TOTAL MARKS [20]

Q2 Memory and Resource Management & Ownership

(a) In a computational science application, the distance between points in a 3D space needs to be computed. For two given points with coordinates (x, y, z) and (x', y', z'), their distance is defined as follows:

$$\sqrt{(x - x')^2 + (y - y')^2 + (z - z')^2}$$

A compute-intensive function that needs to be implemented efficiently, takes as arguments a reference point `pt` and a set of points `arr`, and should compute the sum over all distances of the points in `arr` from point `pt`. Two implementations of this function have been suggested (below), using different data structures for the set of points.

```
// global variable initialisation; used for both representations
ulong n = 2;
struct_t pt = NULL;
```

Version 1 (using an array of points, where each point is a 3-element `structure struct_t`):

```
// use an array-of-struct representation
typedef struct {
        ulong x;  ulong y;   ulong z;
} struct_t;

static double distance_aos(struct_t pt, ulong n, struct_t *arr) {
        int i;
        double sum = 0.0;

        for (i = 0; i < n; i++)
              sum += sqrt(sqr(arr[i].x-pt.x)+
                      sqr(arr[i].y-pt.y)+ sqr(arr[i].z-pt.z));

        return sum;
}
```

Version 2 (using a structure of 3 arrays, for the x, y, z coordinates, respectively):

```
// use a struct-of-array representation
typedef struct {
        ulong *xs;
        ulong *ys;
        ulong *zs;
} soa_t;

static double distance_soa(struct_t pt, ulong n, soa_t *s) {
        int i, j;
        double sum = 0.0;

        for (i = 0; i < n; i++)
              sum += sqrt(sqr(s->xs[i]-pt.x)+
```

```
                    sqr(s->ys[i]-pt.y)+ sqr(s->zs[i]-pt.z));

        return sum;
}
```

i. Identify which variables will be allocated to the stack and which variables will be allocated to the heap for each version. Illustrate your answers with the use of the following table, adding or removing rows as appropriate. Show the heap and stack allocations made for each distance function if `n = 2`.

[6]

	Version 1	Version 2
Stack	<first variable name>	<first variable name>
	<second variable name>	<second variable name>
	…	…
	<n^{th} variable name>	<n^{th} variable name>
	<n^{th} variable name>	<n^{th} variable name>
	…	…
	<second variable name>	<second variable name>
Heap	<first variable name>	<first variable name>

ii. Considering that data is stored in neighbouring addresses in memory (spatial locality) which of the two versions do you think would be better to pack data together and why?

[6]

(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 file. However, this can introduce memory management issues. Rewrite the fragment using the C++ RAII ownership technique, considering the **Programming Question Advice** and explain why the RAII version is better.

[3]

```
const char *filename = "./file.txt"
const char *mode = 'r'
FILE *f = fopen(filename, mode);
...
fclose(f)
```

TOTAL MARKS [20]

Q3 Concurrent Systems Programming

Tasked with performing a threadsafe Hashtable insertion Joe, a naïve programmer, has written the following `safe_insert_if_new()` function in a `hashTable struct`.

```
struct hashTable {
private:
  std::mutex mutex;
  std::unordered_map<std::string, std::string> table;

public:
  bool safe_insert_if_new(char *key, char *value) {
                // If key already in table don't insert
    if (table.find(key) != table.end()) {
      return false;
    }
                // Otherwise lock table and insert {key, value}
    std::unique_lock<std::mutex> lock(mutex);
    table.insert({key, value});
    std::unique_lock<std::mutex> unlock(mutex);
    return true; // name was inserted
  }
  …
}
```

(a) Using the correct technical terms carefully explain two issues with the `safe_insert_if_new` function.

[4]

(b) Joe has heard that semaphores are a powerful way of synchronising C++ threads with `acquire()` and `release()` operations corresponding to the famous `wait()` and `signal()` operations. Paying attention to the **Programming Question Advice** provided at the start of the paper, outline a thread safe and idiomatic `safe_insert_if_new()` implementation using semaphore(s), either counting or binary as appropriate.

[3]

(c) Would you recommend using mutexes or semaphores to protect the hash table functions? Justify your recommendation.

[2]

(d) Consider the following C++ program fragment that uses `randInt()` to populate the global `vec` vector with a million integers. The `sumChunk()` function sums a segment of `vec`.

```
auto size = 1 * 1024 * 1024;
auto vec = std::vector<int>(size);

// Sum a vector segment
int sumChunk(int lower, int upper) {
  int sum = 0;
  for(int i = lower; i < upper; i++)
    sum += vec[i];
  return sum;
}


int main() {
  std::generate(vec.begin(), vec.end(), randInt);
  …
}
```

i. Outline a program that creates a thread to sum the lower half of the vector, while summing the upper half in the main thread. Pay attention to the **Programming Question Advice** provided at the start of the paper.

ii. Outline a program that creates 9 threads to sum approximately equally sized segments of the vector, while summing the last segment in the main thread.

[11]

TOTAL MARKS [20]

END OF EXAM