University of Glasgow

May 2019
(Duration: 1 hour 30 minutes)

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

System Programming H
Mock Exam

(Answer all 3 questions.)

This examination paper is worth a total of 60 marks

The use of a calculator is not permitted in this examination

INSTRUCTIONS TO INVIGILATORS

Please collect all exam question papers and exam answer scripts and retain for school to collect. Candidates must not remove exam question papers.

1. C Programming and Data Types

(a) Describe how strings are represented in C. How much bytes of memory do the string literal "`Hello World`" require in a C program? **[2]**

(b) Implement a *binary search tree of strings* in the C programming language.

Sketch the implementation based on the following interface. Minor syntax errors will not be penalised. Make sure that you handle pointers correctly and that you do not leak memory. Provide comments explaining your implementation.

```
// a node in the tree should store a string label
typedef struct node Node;

// create a Node with the given label. Returns NULL if unsuccessful
Node* node_create(const char* label);

// destroys the node and connected nodes in the tree
void node_destroy(Node* n);

// inserts a node in the right place to maintain the order of the tree
void insert(Node* n, const char* label);

// lookup a node in the tree. Returns NULL if the label is not present
Node* lookup(Node* n, const char* label);
```

You should use these string handling functions:

```
int strcmp(const char* s1, const char* s2);
```
returns 0 is `s1` and `s2` are equal, a value `<0` if `s1` is smaller than `s2`, `>0` if `s1` is bigger than `s2`

```
int strlen(const char* s);
```
returns the number of printable characters in the string

```
char* strcpy(char* destination, const char* source);
```
copies the string at `source` to the `destination`

- Define the `struct`. A `Node` should store a string label. **[2]**

- Implement `node_create` and `node_destroy`.
  Be careful to handle all allocations and pointers properly. **[5]**

- Implement `insert_node`. Ensure that the node is inserted at the right place to maintain the search order of the tree. If a node with the given label exists already in the tree no node should be added. **[4]**

- Implement `lookup`. **[3]**

- Implement a `main` function that uses the provided interface to create a binary search tree with three nodes with the labels “Systems”, “Programming”, “2019”. Write a function that prints all values in the tree and show how it is used. The tree should be destroyed at the end of the program and there should be no memory leaks. **[4]**

TOTAL MARKS [20]

2. Memory and Resource Management & Ownership

(a) Explain the purpose of a `void`-pointer in C. Give an example use case. **[2]**

(b) Explain the difference between the two memory regions *stack* and *heap*. Explain briefly how the programmer manages memory on the heap in C. **[2]**

(c) Values are passed to function via *call-by-value*. Explain briefly what this means. Explain how arrays are treated when passed to a function and how this relates to pointers. **[3]**

(d) Explain what a *segmentation fault* is and describe a strategy for finding out which part of a C program triggered this error. Give a common cause for a segmentation fault. **[3]**

(e) Explain the concept of *Ownership* for memory management. Describe the benefits over the direct use of `malloc` and `free`. Explain how in C++ the RAII technique works and how this relates to the lifetime of variables and what role the special *constructor* and *destructor* functions play in this context. **[4]**

(f) C++ provides two main smart pointers: `std::unique_ptr` and `std::shared_ptr`. Explain the difference between the two and discuss in which situation which one should be used. **[2]**

(g) Give the definition of a `struct` for a Node with a string label in a directed acyclic graph (DAG) [a graph with directed edges and without cycles] that uses C++ containers and C++ smart pointers to model and manage the graph data structure. Explain your solution briefly.
Would a similar implementation be possible for a graph with cycles?
Justify your answer.

Describe (but do not implement) what implementation would be required in C to achieve a correct and leak free implementation. **[4]**

TOTAL MARKS [20]

3. Concurrent Systems Programming

(a) Describe the term *Thread* and distinguish it from the term *Process*. **[2]**

(b) Give an example showing the need for mutual exclusion to ensure the correctness of the results of a computation. **[2]**

(c) C++ provides *futures* and *promises* as higher-level synchronisation abstractions. Describe how they work together to simplify writing multi-threaded code. **[2]**

(d) Look at the following API definition of the `ts_set` class.

```
struct ts_set {
private:
  std::set<int> set;
  std::mutex m;

public:
  std::set<int>::iterator find(int i) {
    std::unique_lock<std::mutex> lock(m);
    return set.find();
  }
  std::set<int>::iterator begin() {
    std::unique_lock<std::mutex> lock(m);
    return set.begin();
  }
  std::set<int>::iterator end() {
    std::unique_lock<std::mutex> lock(m);
    return set.end();
  }
  void insert(int i) {
    std::unique_lock<std::mutex> lock(m);
    set.insert(i);
  }
  void erase(std::set<int>::iterator pos) {
    std::unique_lock<std::mutex> lock(m);
    set.erase(pos);
  }
};
```

The provided interface and implementation is not safe to use in multithreaded code despite using a mutex. Explain why and give an example describing a problematic scenario using the API with multiple threads.

[4]

(e) Design a thread safe interface for a stack of `int` values with a limited size and implement it in C++ or PThreads.

The stack should allow to *push* new elements onto the stack and to *pop* (i.e. remove) elements from the stack.
When the stack is full *push* should block and wait until there is again space, similarly *pop* should block when the stack is empty until there are elements to be removed.
Use condition variables in your solution to avoid busy waiting. **[6]**

Implement *try_push* and *try_pop* that will not block but instead immediately return when the stack is full/empty. Both functions should indicate if the operation was performed successfully or not.

[4]

TOTAL MARKS [20]