C/C++ Exercise Sheet 2012–2013

Lecture 1

1. What is the difference between `'a'` and `"a"`?

2. Will `char i,j; for(i=0; i<10,j<5; i++,j++) ;` terminate? If so, under what circumstances?

3. Write an implementation of bubble sort for a fixed array of integers. (An array of integers can be defined as `int i[] = {1,2,3,4}`; the 2nd integer in an array can be printed using `printf("%d\n",i[1]);`.)

4. Modify your answer to (3) to sort characters into lexicographical order. (The 2nd character in a character array `i` can be printed using `printf("%c\n",i[1]);`.)

Lecture 2

1. Write a function definition which matches the declaration `int cntlower(char str[]);`. The implementation should return the number of lower-case letters in a string

2. Use function recursion to write an implementation of merge sort for a fixed array of integers; how much memory does your program use for a list of length n?

3. Define a macro `SWAP(t,x,y)` that exchanges two arguments of type `t` (K&R, Exercise 4-14)

4. Does your macro work as expected for `SWAP(int, v[i++], w[f(x)])`?

5. Define a macro `SWAP(x,y)` that exchanges two arguments of the same type (e.g. `int` or `char`) *without using a temporary*

Lecture 3

1. If `p` is a pointer, what does `p[-2]` mean? When is this legal?

2. Write a string search function with a declaration of `char *strfind(const char *s, const char *f);` which returns a pointer to first occurrence of `s` in `f` (and `NULL` otherwise)

3. If `p` is a pointer to a structure, write some C code which uses all the following code snippets: “`++p->i`”, “`p++->i`”, “`*p->i`”, “`*p->i++`”, “`(*p->i)++`” and “`*p++->i`”; describe the action of each code snippet

4. Write a program `calc` which evaluates a reverse Polish expression given on the command line; for example
   ```
   $ calc 2 3 4 + *
   ```
   should print `14` (K&R Exercise 5-10)

Lecture 4

1. What is the value of `i` after executing each of the following:

   (a) `i = sizeof(char);`

   (b) `i = sizeof(int);`

   (c) `int a; i = sizeof a;`

   (d) `char b[5]; i = sizeof(b);`

   (e) `char *c=b; i = sizeof(c);`

   (f) `struct {int d;char e;} s; i = sizeof s;`

   (g) `void f(int j[5]) { i = sizeof j;}`

   (h) `void f(int j[][10]) { i = sizeof j;}`

2. Use `struct` to define a data structure suitable for representing a binary tree of integers. Write a function `heapify()`, which takes a pointer to an integer array of values and a pointer to the head of an (empty) tree and builds a binary heap of the integer array values. (Hint: you'll need to use `malloc()`)

3. What other C data structure can be used to represent a heap? Would using this structure lead to a more efficient implementation of `heapify()`?

Lecture 5

1. Write an implementation of a class `LinkList` which stores zero or more positive integers internally as a linked list *on the heap*. The class should provide appropriate constructors and destructors and a method `pop()` to remove items from the head of the list. The method `pop()` should return -1 if there are no remaining items. Your implementation should override the copy constructor and assignment operator to copy the linked-list structure between class instances. You might like to test your implementation with the following:

```
int main() {
  int test[] = {1,2,3,4,5};
  LinkList l1(test+1,4), l2(test,5);
  LinkList l3=l2, l4;
  l4=l1;
  printf("%d %d %d\n",l1.pop(),l3.pop(),l4.pop());
  return 0;
}
```

   Hint: heap allocation & deallocation should occur exactly once!

Lecture 6

1. If a function `f` has a static instance of a class as a local variable, when might the class constructor be called?

2. Write a class `Matrix` which allows a programmer to define 2 × 2 matrices. Overload the common operators (e.g. `+`, `-`, `*`, and `/`)

3. Write a class `Vector` which allows a programmer to define a vector of length two. Modify your `Matrix` and `Vector` classes so that they interoperate correctly (e.g. `v2 = m*v1` should work as expected)

4. Why should destructors in an abstract class almost always be declared `virtual`?

Lecture 7

1. Provide an implementation for:
   `template<class T> T Stack<T>::pop();` and
   `template<class T> Stack<T>::~Stack();`

2. Provide an implementation for:
   `Stack(const Stack& s);` and
   `Stack& operator=(const Stack& s);`

3. Using meta programming, write a templated class `prime`, which evaluates whether a literal integer constant (e.g. 7) is prime or not at compile time.

4. How can you be sure that your implementation of class `prime` has been evaluated at compile time?

Lecture 8

Past exam questions can be found at:
`http://www.cl.cam.ac.uk/teaching/exams/pastpapers/t-ProgramminginCandC++.html`.