C/C++ Exercise Sheet 2022–23

Lecture 1

1. What is the difference in C 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 an array of characters into alphabetical 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 integer type (e.g. `int` or `char`) *without using a temporary*
6. What is the effect of `SWAP(*p,*q)` when `p==q`?

Lectures 3 and 4

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
   `const char *strfind(const char *needle, const char *hay);`
   that returns a pointer to first occurrence of `needle` in `hay` (and `NULL` otherwise). (You are not expected to implement Boyer-Moore algorithm but you might find it of general interest.)
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 + x`
   should print `14` We use 'x' for multiply since asterisk will be expanded by the shell into a file list.(K&R Exercise 5-10)

5. What is the value of `i` after executing each of the following on your laptop (and which might vary on a 2015 vintage Raspberry Pi?):

   (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;}`

6. Write a program that adds up all of the characters in a large file, treating them as unsigned 8-bit quantities. The program should print the total.

7. If you used `fopen` or `read` for the previous exercise, do it again, but this time use `mmap`. Or vice versa. Compare the performance of the two programs (e.g. simply bracket it as follows `date; ./mprog; date` or `time ./a.out`. Is the performance a linear function of file size? What is the performance of your file system in MByte per second? Does it make a difference if the program is run again just after it has been run on the same file or the file is accessed twice in the same run of a program?

8. Give alternative C code that does not use square brackets for each of the following expressions: `A[b]`, `b[A]`, `A[b, c]` and `A[b][c]`, Which form is suitable for a *jagged array*? [Hint: don't forget that C has a construct which is a comma-separated expression list.]

Lecture 5 – Tooling

1. Define a nest of one or more `union` and `struct` forms where a 32-bit integer and a 64-bit integer respectively alias a 32-bit float and a 64-bit double. Demonstrate the different behaviours arising from casting between `int` and `float` types and reading a different alias from the one written.

2. Describe what padding or other alignment rules your compiler implements in `struct` records where fields of different width are adjacent. You can use the `&` address-of operator to find the offset of a field and the `%p` `printf` format if you wish. You may need to include one or more `char` variables to explore the behaviour.

3. The 'where' command in a C debugger, like gdb, displays the stack of a thread. In basic terms, how is such a debugger able to display the stack, examine register and memory contents, and single step a C program? Run a large program, such as spotify or gcc, under valgrind. What performance change do you note? Note: some large 'programs' are actually shell scripts or connect to existing instances of them already running on your machine (e.g. evince or zoom) and so the file you think you are executing is different from what ends up running.

   Which of ASan, MSan, UBSan and valgrind require the source code of a program to be available or modifies the executable code? Discuss why.

Lecture 6 and 7 – Trees, Arenas, Memory allocation.

1. 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 *on the heap* of the integer array values. (Hint: you'll need to use `malloc()`. Note the two different meanings of the word 'heap' here.)

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

3. Write your own, very basic implementation of `malloc` and `free` and use this for the heap on the heap exercise (or a similar test). You should get your memory from either allocating a large static array or make just one call to the system-provided implementation of `malloc`. Use a linked list to hold blocks that have been freed. Why might having multiple linked lists for blocks of different sizes be sensible? Why might rounding up the argument to every `malloc` call to the next highest value in a Fibonacci series be sensible?

Lecture 8 – Cache-Friendly Programming

```
int initialise(int dasize)
{
  for (j=0; j!=dasize; j++)
      for (i=0; i!=dasize; i++)
        adata[ i + (j*dasize)] = i;
}
```

Prof David May pointed out that, given a 2-D array filled with the identity function as above, the following three ways of iterating through it run at very different speeds.

```
int dm_sumarray0(int dasize)
{ int sum = 0; for (i=0; i!=dasize; i++)
    for (j=0; j!=dasize; j++)
      {
        sum += adata[ j + (i*dasize)];
      }}
  return sum; }

int dm_sumarray1(int dasize)
{ int sum=0; for (i=0; i!=dasize; i++)
    for (j=0; j!=dasize; j++)
      {
        sum += adata[ i + (j*dasize)];
      }
  return sum; }

int dm_sumarray2(int dasize)
{ int sum=0; for (j=0; j!=dasize; j++)
    { int i=0;
      while (i!=dasize)
        {
          int v = adata[ i + (j*dasize)];
          sum += v;
          i = v+1; // Data-dependent loop - cannot unroll.
        }
```

```
  }
return sum; }
```

1. Explore the run time of each of the three programs. Perhaps make some plots of performance as dasize is increased. Compare dasize with the square root of your L1 and L2 cache size. `cat /proc/cpuinfo` will give you some basic CPU info. Explain any kinks you see.

2. When you've finished the Computer Design course, consider which of the above programs leads to good pipeline behaviour.

Lecture 9 – Debugging

1. Investigate the differences arising from adding '-g' to your C compiler arguments. Ditto '-O2'. What does 'strip' do and when might you use it?

2. Investigate several symbol tables report generated by the programs 'nm' and 'objdump' with various flags supplied. You should be able to see a full disassembly. 'readelf' may also be of use.

3. What differences do you see between the `_start` code for C and C++? You can see this most easily by disassembling binaries you have created yourself.

C++ Questions

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!

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

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

4. 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)

5. Give an example where failure to make a destructor `virtual` causes a memory leak.

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

7. You have been given the following seemingly working C code

```
int process_file(char *name) {
  FILE *p = fopen(name, "r");
  if (p == NULL) return ERR_NOTFOUND;
  while (...) {
    ...
    if (...) return ERR_MALFORMED;
    process_one_option();
    ...
  }
  fclose(p);
  return SUCCESS;
}
```

   but a user reports a bug that after reading several files with malformed input, the program fails to open files.

   - Explain and identify the bug and fix it (using C code).
   - By wrapping the calls to `fopen` and `fclose` within a suitable class, give a C++ solution. Your solution should work sensibly if `process_one_option()` might raises an exception.

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

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

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

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

12. You have been given the following C code:

```
1  #include<stdio.h>
2
3  #define init_employee(X,Y) {(X),(Y),wage_emp}
4  typedef struct Employee Em;
5  struct Employee {int hours,salary;int (*wage)(Em*);};
6  int wage_emp(Em *ths) {return ths->hours*ths->salary;}
7
8  #define init_manager(X,Y,Z) {(X),(Y),wage_man,(Z)}
9  typedef struct Manager Mn;
10 struct Manager {int hours,salary;int (*wage)(Mn*);int bonus;};
11 int wage_man(Mn *ths) {return ths->hours*ths->salary+ths->bonus;}
12
13 int main(void) {
14   Mn m = init_manager(40,10,20);
15   Em *e= (Em *) &m;
16   printf("%d\n",e->wage(e));
17   return 0;
18 }
```

    Rewrite it using C++ primitives and give four reasons why your C++ solution is better than the original C version.

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