UNIVERSITY OF DUBLIN

TRINITY COLLEGE

Faculty of Engineering, Mathematics & Science
School of Computer Science & Statistics

Integrated Computer Science Programme **Trinity Term 2013**

B.A. (Mod.) Business and Computing

Systems Programming I and II (CS2014/5)

Friday 10th May 2013 **RDS Main Hall** **09:30-12:30**

Dr David Gregg

Instructions to Candidates:

- Answer 4 out of the 6 questions
- All questions are marked out of 25
- All program code should be commented, indented and use good programming style

Materials permitted for this examination:

- Calculator.

1. Bad programming often makes programs difficult to understand, and unfortunately we often have to understand poorly written code. Describe what the following pieces of C code do, and write new versions that provide the same functionality but with simple programming style, appropriate function and variable names, and suitable indentation.

```
/* code section A */
int b (char c, int x) {
x=0; switch(c) {case '9': x++; case'8': x++; case '7': x++; case '6': x++;
case '5': x++; case '4': x++; case '3': x++; case '2': x++; case '1': x++;
default: break;} return x;
}
```

[5 marks]

```
/* code section B */
int f (int * a, int b) {
    int c = 0; while (b--) c+= !a[b]; return c;
}
```

[5 marks]

```
/* code section C */
void e (char * a, char *b) {do; while (*b++ = *a++);}
```

[7 marks]

```
/* code section D */
void h (int * a, int b) {
for (int x=b; --x;)
        if (a[x] > a[x-1]) {
                int c = a[x];
                a[x] = a[x-1];
                a[x-1] = c; x = b;
        }
}
```

[8 marks]

2. You are asked to write an application that needs to keep track of sets of web addresses (URLs). Write a C (not C++) abstract data type (ADT) to represent sets of strings using a hash table. Your hash function should be suitable for hashing URLs. Your ADT should provide functions to add, remove, and check for membership of the set, as well as functions to implement set union and set intersection.

   [25 marks]

3. The C programming language allows characters, integers and other types to be written to files, but it provides no direct support for writing information to a file in chunks of less than one byte. This is unfortunate because many applications can save file space by saving streams of individual bits, or irregular sized chunks of data.

   Write a **C** (not C++) abstract data type (ADT) of a file that allows information to be written to the file one bit at a time. Your ADT should support operations to open a new file, write a bit to the file, and close the file. Note that you will need to use the normal C file features within your ADT. You also need to consider the situation where the file ends at an uneven bit number. Hint: The C standard function fputc(unsigned char c, FILE * file) is probably the most convenient for writing a byte to a file.

   [25 marks]

4. C++ Standard Template Library (STL) provides a set of standard container classes for use in C++ programs. One of these is the doubly-ended queue class, which implements a queue where values can be pushed or popped from either end. The following shows the broad outline of a simplified version of the STL list class.

```
template <class T>
class mydeque{
private:
        // add your own private variables and methods here
public:
        mydeque( );             // create new, empty deque
        ~mydeque( );            // destructor
        void push_back(T item);     // add new item onto end of deque
        void push_front(T item);    // add a new item to front of deque
        T pop_back( );          // remove and return last item of deque
        T pop_front( );         // remove and return first item of deque
        T& operator[ ](int  i); // return the i'th element of the deque
};
```

Add the remaining declarations to this class, and provide the bodies of methods to implement each of the methods listed above. You may use the basic building blocks of the C++ language (arrays, classes) to construct your class, but you may not use the STL in your code.

[25 marks]

5. A spam filter is a program that attempts to identify junk email. A commonly used approach to identifying spam is to search for particular strings in the body of the email. For example, the filter might search for words such as “shares”.

   One complication is that senders of spam sometimes rewrite some characters in a word to make it unrecognisable to the spam filter, but still readable for a human. For example, the spammer might replace the character “a” with “ä” and “e” with “3”. This would mutate the word “shares” into “shär3s”. One way to counteract this is for the spam filter to keep a list of possible alternative characters for each of the 256 ASCII characters. Thus, when searching incoming email for the word “shares” it will look up the lists, and check for variations which include replacing “a” with “ä” and “e” with “3”.

   Write a C++ function that searches for a given string *str* in a line of text *line*. The function should also check for all possible mutations of *str*, based on replacing individual characters in *str* with alternatives. The prototype for the function will look like:
   bool spam_search(string str, string line, string * mutations);

   Where *str* and *line* are C++ strings, and *mutations* is an array of C++ strings with 256 array elements. Each of the 256 array elements in mutations consists of a C++ string, which contains the list of characters that the corresponding original character can be replaced with when mutated.

   [25 marks]

6. An index in a book contains an alphabetic list of the words in the book, and a sorted list of the pages where each word appears. Given a C++ Standard Template Library (STL) vector of strings, where each string contains the text for one page of the book, write a function which writes out to a text file an index for the book. For the purposes of this question, a word is a maximal sequence of letters in the text. A letter is a character in the range 'a' to 'z' or the range 'A' to 'Z'. Your function should have the following prototype:

   ```
   void write_index(vector<string> pages, ofstream & file);
   ```

   In the prototype *pages* is a vector of strings, where each string contains the full text for one page of the book. The page number of each page corresponds to its position in the vector. So, for example, page 5 is stored in element 5 of the vector. If the same word appears more than once on a single page, only one appearance of that word on that page should be recorded in the index.

   You may use classes from the STL in your solution. Note also, to simplify sorting the words, you may convert them to lower case before sorting them and writing them out. You may also use the following built in functions:

   ```
   bool is_letter(char c)    // returns true if c is a letter
   bool is_upper(char c)     // returns true if c is an upper case letter
   bool is_lower(char c)     // returns true if c is a lower case letter
   char to_lower(char c)     // converts upper case letter c to lower case
   char to_upp(char c)       // converts lower case letter c to upper case
   ```

[25 marks]