This document is part of the HTML publication "**An Introduction to the Imperative Part of C++**"

The original version was produced by **Rob Miller** at **Imperial College London**, September 1996.

Version 1.1 (modified by **David Clark** at **Imperial College London**, September 1997)

Version 1.2 (modified by **Bob White** at **Imperial College London**, September 1998)

Version 1.3, 1.4, 2.0, ..., 2.15 (modified by **William Knottenbelt** at **Imperial College London**, September 1999-September 2016)

# Introduction to C++ Programming: Exercise Sheet 7

## Question 1

Predict the output of the following program. Run the program to see if your prediction is right.

```
#include <iostream>
using namespace std;

typedef int *IntPtrType;

int main()
{
        IntPtrType ptr_a, ptr_b, *ptr_c;

        ptr_a = new int;
        *ptr_a = 3;
        ptr_b = ptr_a;
        cout << *ptr_a << " " << *ptr_b << "\n";

        ptr_b = new int;
        *ptr_b = 9;
        cout << *ptr_a << " " << *ptr_b << "\n";

        *ptr_b = *ptr_a;
        cout << *ptr_a << " " << *ptr_b << "\n";

        delete ptr_a;
        ptr_a = ptr_b;
        cout << *ptr_a << " " << *&*&*&*&*ptr_b << "\n";

        ptr_c = &ptr_a;
        cout << *ptr_c << " " << **ptr_c << "\n";

        delete ptr_a;
        ptr_a = NULL;

        return 0;
}
```

(EXAMPLE ANSWER) (BACK TO COURSE CONTENTS)

## Question 2

Write a Boolean valued function which returns "True" if its first string argument is alphabetically smaller than its second string argument, "False" otherwise. You may assume that the two strings contain only lower case letters, and no blanks or other non-alphabetic characters. Test your function with a suitable main program. When you are satisfied it works properly, convert the function to pointer arithmetic syntax, and check that it still behaves in the same way.

(EXAMPLE ANSWER) (BACK TO COURSE CONTENTS)

## Question 3

Add the following 3 functions to Program 7.5.1 from Lecture 7, and modify the program to test them:

- The function

  ```
  void add_after(node_ptr &list, char a_word[], char word_after[])
  ```

  which inserts a node containing "word_after" in the linked list "list", after the first occurrence of a node containing "a_word". If "list" does not contain such a node, the function leaves it unchanged.
- The function

  ```
  void delete_node(node_ptr &a_list, char a_word[])
  ```

  which deletes the first node in "a_list" which contains "a_word".
- The function

  ```
  void list_selection_sort(node_ptr &a_list)
  ```

  which sorts a list alphabetically (use your answer to Question 2 to help).

Example input/output might be:

```
Enter first word (or '.' to end list): the
Enter next word (or '.' to end list): quick
Enter next word (or '.' to end list): brown
Enter next word (or '.' to end list): fox
Enter next word (or '.' to end list): jumped
Enter next word (or '.' to end list): over
Enter next word (or '.' to end list): the
Enter next word (or '.' to end list): lazy
Enter next word (or '.' to end list): dog
Enter next word (or '.' to end list): .

THE LIST IS NOW:
the quick brown fox jumped over the lazy dog

AFTER WHICH WORD WOULD YOU LIKE TO ADD AN EXTRA WORD? the
WHICH WORD WOULD YOU LIKE TO ADD? very

THE LIST IS NOW:
the very quick brown fox jumped over the lazy dog

WHICH WORD WOULD YOU LIKE TO DELETE? lazy

THE LIST IS NOW:
the very quick brown fox jumped over the dog

AFTER SORTING, THE LIST IS:
brown dog fox jumped over quick the the very
```

***Hints:***

*In words, a possible algorithm to add a node is:*
*(i) Use an extra node pointer to find and identify the node after which you would like to add the new node,*
*(ii) Use a second extra node pointer to create the new node,*
*(iii) Make appropriate adjustments to the pointer inside the node you identified in step (i) and the pointer inside the new node you created in step (ii).*

*A possible algorithm to delete a node is:*
*(i) Position one extra pointer pointing to the node before the one you wish to delete, and another extra pointer pointing to the actual node to be deleted.*
*(ii) Make an appropriate adjustment to the pointer inside the node before the one to be deleted, so that it skips over the obsolete node and points straight to the one after.*
*(iii) Delete the obsolete node.*

*The algorithm given in Lecture 6 to sort an array can be straightforwardly adapted to sort a linked list.*

(EXAMPLE ANSWER) (BACK TO COURSE CONTENTS)

---