

School of Computer Science and Software Engineering  
Clayton Campus, Monash University

**CSE1303 Part A**  
**Summer Semester, 2002**

## Practical Session 3: Dynamic Memory



### ***Aims of this Practical Session***

1. To familiarise yourself with allocating and deallocating dynamic memory.
2. To understand some binary operations.

**Project:** To write a program which performs various operations on binary numbers.

### ***Background:***

In this prac we will represent a binary number as an array of **char**, where each element is either a '0' or '1'. (This is a very inefficient representation in terms of space and a more efficient representation would be to represent the number as an array of *unsigned char* and use *bitwise* operations – not implemented in this prac).

The binary operations we will consider in this prac are:

- One's complement
- Left Shift
- Right Shift

**One's complement** (also known as bitwise not) changes every '0' to '1', and every '1' to '0'. **Left Shift** by **n**, shifts all the bits left by **n** places, filling vacated bits with '0'. This is equivalent to multiplication by  $2^n$ . **Right Shift** by **n**, removes the first **n**-bits. This is equivalent to integer division by  $2^n$ . (We will always assume that **n** is a positive integer.)

For example, consider the following binary number:

**100101100**

The one's complement of this binary number is:

**011010011**

and when we left shifted it by **3** and right shift it by **5** we obtain, respectively:

**100101100000** and **1001**

**Preparation (3 marks if completed before class)**

Before this prac:

1. For the following binary numbers:

**11011011** and **1011000100**

- Find their one's complement.
  - Shift them both left by **4**.
  - Shift them both right by **3**.
2. Write a C declaration for a structure, **Binary**, to store a pointer of type **char\***, which will point to an array storing the binary number, and an **int**, which will record the size of the array.
  3. Write a C program which reads in an **int**, (lets call it **n**), and a binary number with **n** bits. The program should store the binary number in a variable of type **Binary** and print it out, and **deallocate any dynamic memory allocated**.  
(Recall lecture A0 5 on ADTs)

**Question 1: (3 marks)**

1. Write a C function **void readBinary(Binary\* bPtr)** which reads in a binary number and stores it in the structure pointed to by **bPtr**.
2. Write a C function **void printBinary(const Binary\* bPtr)** which prints out a binary number stored in the structure pointed to by **bPtr**.
3. Extend the program you wrote in the preparation so that it has a menu with the following options to manipulate a **Binary** structure:
  - **Read** which asks the user for an integer, say **n**, and a binary number with **n** bits and stores it in a **Binary** structure.
  - **Print** which prints out the last binary number read.
  - **Quit** which allows the user to quit the menu and exit the program.

(Remember Tutorial 1 had a question about a menu system).

⊕ **Make sure you deallocate any memory that is no longer required.**

**Question 2: (4 marks)**

1. Write a C function **void complement(const Binary\* oldPtr, Binary\* newPtr)** which finds the one's complement of the binary number stored in the structure pointed to by **oldPtr**, and stores result in the structure pointed to **newPtr**. .  
(You will need to allocate memory for the new binary number).
2. Write a C function **void shiftLeft(const Binary\* oldPtr, int n, Binary\* newPtr)** which finds the left shift by **n** of the binary number stored in the structure pointed to by **oldPtr**, and stores result in the structure pointed to **newPtr**. . (Again, you will have to allocate memory for the new number).

**Note: If a binary number has  $m$  bits, then after it has been left shifted by  $n$  it has  $m+n$  bits.**

3. Write a C function **void shiftRight(const Binary\* oldPtr, int n, Binary\* newPtr)** which finds the right shift by **n** of the binary number stored in the structure pointed to by **oldPtr**, and stores result in the structure pointed to **newPtr**. (Again, you will have to allocate memory for the new number).

⊕ **Note: If a binary number with  $m$  bits is right shifted by  $n$  bits, where  $n = m$ , then the result is 0.**

⊕ **Note: If a binary number has  $m$  bits, then after it has been right shifted by  $n$  it has  $m-n$  bits (when  $n < m$ ), and 1 bit otherwise.**

4. Extend the program your wrote in Question 1 so that it has a menu with the following extra options to manipulate a **Binary**.
  - **One's complement:** which prints out the one's complement of the last binary number read in.
  - **Shift Left:** which asks the user for an integer, say **n**, and shifts left by **n** the last binary number read in, and prints it out.
  - **Shift Right** which asks the user for an integer, say **n**, and shifts right by **n** the last binary number read in, and prints it out.

**Make sure you deallocate any memory that is no longer required.**

**Advanced Questions: (2 marks)**

1. Extend the program you have written so the menu includes the following functions (*See lectures B04 and B05*):
  - **Add**: which takes two binary numbers, adds them together and prints the result
  - **Multiply**: which takes two binary numbers, multiplies them together and prints the result.
  - **Bitwise AND**: which takes two binary numbers, **ANDs** them together and prints the result.
  - **Bitwise inclusive OR**: which takes two binary numbers, **ORs** them together and prints the result.
  - **Bitwise exclusive OR**: which takes two binary numbers, **exclusive ORs** them together and prints the result.