COMP9024 25T1

Week 1 Practical
Elementary Data and Control Structures in C

Data Structures and Algorithms

1. (ADO client)

An integer stack can be used to convert a positive decimal number *n* to a different numeral system with base *k* according to the following algorithm:

while *n*>0 **do**
  push *n%k* onto the stack
  n = *n* / *k*
end while

The result can be displayed by printing the numbers as they are popped off the stack. Example (*k*=2):

```
n = 13          --> push 1 (= 13%2)
n = 6  (= 13/2) --> push 0 (= 6%2)
n = 3  (=  6/2) --> push 1 (= 3%2)
n = 1  (=  3/2) --> push 1 (= 1%2)
n = 0  (=  1/2)
Result: 1101
```

Using your stack ADO from Week 1 Tutorial, Exercise 5, write a C-program that implements this algorithm to transform a decimal number given on the command line into a binary (i.e. base *k*=2) number.

Examples of the program executing could be

```
prompt$ ./binary
Enter a number: 13
1101
prompt$ ./binary
Enter a number: 128
10000000
prompt$ ./binary
Enter a number: 127
1111111
```

We have created a script that can automatically test your program. To run this test you can execute the `dryrun` *program that corresponds to this exercise. It expects to find three programs in the current directory:*

- *`IntStack.h` – your header file for the integer stack from Week 1 Tutorial, Exercise 5*
- *`IntStack.c` – your implementation of the integer stack Week 1 Tutorial, Exercise 5*
- *`binary.c`*

You can use dryrun as follows:

```
prompt$ 9024 dryrun binary
```

Answer:

`IntStack.h`

```
// Integer Stack ADO header file
```

```
#define MAXITEMS 10

void StackInit();     // set up empty stack
int  StackIsEmpty();  // check whether stack is empty
void StackPush(int);  // insert int on top of stack
int  StackPop();      // remove int from top of stack
```

IntStack.c

```
// Integer Stack ADO implementation

#include <assert.h>
#include "IntStack.h"

typedef struct {
   int item[MAXITEMS];
   int top;
} stackRep;                // defines the Data Structure

static stackRep stackObject;  // defines the Data Object

void StackInit() {         // set up empty stack
   stackObject.top = -1;
}

int StackIsEmpty() {       // check whether stack is empty
   return (stackObject.top < 0);
}

void StackPush(int n) {    // insert int on top of stack
   assert(stackObject.top < MAXITEMS-1);
   stackObject.top++;
   int i = stackObject.top;
   stackObject.item[i] = n;
}

int StackPop() {           // remove int from top of stack
   assert(stackObject.top > -1);
   int i = stackObject.top;
   int n = stackObject.item[i];
   stackObject.top--;
   return n;
}
```

binary.c

```
#include <stdlib.h>
#include <stdio.h>
#include "IntStack.h"

int main(void) {
   int n;
   char str[BUFSIZ];

   printf("Enter a number: ");
   scanf("%s", str);
   n = atoi(str);
   StackInit();
   while (n > 0) {
      StackPush(n % 2);
      n = n / 2;
   }
   while (!StackIsEmpty()) {
```

```
      printf("%d", StackPop());
   }
   putchar('\n');
   return 0;
}
```

2. (Queue ADO)

   Modify your integer stack ADO from Week 1 Tutorial, Exercise 5 to an integer queue ADO.

   Hint: A *queue* is a FIFO data structure (first in, first out). The principal operations are to *enqueue* and to *dequeue* elements. Elements are dequeued in the same order in which they have been enqueued. Below is the header file (`IntQueue.h`) with the functions that your ADO should provide.

   `IntQueue.h`

```
// Integer Queue ADO header file ... COMP9024 25T1

#define MAXITEMS 10

void QueueInit();        // set up empty queue
int  QueueIsEmpty();     // check whether queue is empty
void QueueEnqueue(int);  // insert int at end of queue
int  QueueDequeue();     // remove int from front of queue
```

   We have created a script that can automatically test your program. To run this test you can execute the `dryrun` *program that corresponds to this exercise. It expects to find two files named* `IntQueue.c` *and* `IntQueue.h` *in the current directory that provide an implementation of a queue ADO with the four queue functions shown above. You can use dryrun as follows:*

```
prompt$ 9024 dryrun IntQueue
```

   Answer:

   `IntQueue.c`

```
// Integer Queue ADO implementation

#include <assert.h>
#include "IntQueue.h"

static struct {
   int item[MAXITEMS];
   int top;
} queueObject;  // defines the Data Object

void QueueInit() {          // set up empty queue
   queueObject.top = -1;
}

int QueueIsEmpty() {        // check whether queue is empty
   return (queueObject.top < 0);
}

void QueueEnqueue(int n) {  // insert int at end of queue
   assert(queueObject.top < MAXITEMS-1);
   queueObject.top++;
   int i;
   for (i = queueObject.top; i > 0; i--) {
      queueObject.item[i] = queueObject.item[i-1]; // move all elements up
   }
```

```
    queueObject.item[0] = n; // add element at end of queue
}

int QueueDequeue() {        // remove int from front of queue
    assert(queueObject.top > -1);
    int i = queueObject.top;
    int n = queueObject.item[i];
    queueObject.top--;
    return n;
}
```

Due Date

Tuesday, 25 February, 11:59:59. Late submissions will not be accepted.

Submission

This first weekly assignment is meant to give you additional practice and will not count towards your mark for the weekly assessment component.

However, in order to familiarise yourself with the submission and auto-marking process, you may submit your solutions.

You should submit your files using the following `give` command:

```
prompt$ give cs9024 week1b binary.c IntQueue.h IntQueue.c
```

Alternatively, you can select the option to "Make Submission" at the top of this page to submit directly through WebCMS3.

Important notes:

- Make sure you spell all filenames correctly.
- You can run `give` multiple times. Only your last submission will be marked.
- Where you're expected to submit multiple files, ensure they form a single submission. If you separate them across multiple submissions, each submission will replace the previous one.
- Whether you submit through the command line or WebCMS3, it is your responsibility to ensure it reports a successful submission. Failure to submit correctly will not be considered as an excuse.
- You cannot obtain marks by e-mailing your code to tutors or lecturers.

Assessment

This first weekly assignment is meant to give you additional practice and will not count towards your mark for the weekly assessment component.

- Each question is worth 1 mark, for a total of 2 marks.
- Submissions will be auto-marked shortly after the deadline.
- It is important that the output of your program follows exactly the format of the sample output, otherwise auto-marking will result in 0 marks.
- Ensure that your program compiles on a CSE machine with the standard options, i.e. `-Wall -Werror -std=c11`. Programs that do not compile will receive 0 marks.
- Auto-marking will use test cases different to `dryrun`. (Hint: do your own testing in addition to running `dryrun`).

Collection

Once marking is complete you can collect your marked submission using the following command:

```
prompt$ 9024 classrun -collect week1b
```

You can also view your marks using the following command:

```
prompt$ 9024 classrun -sturec
```

You can also collect your marked submission directly through WebCMS3 from the "Collect Submission" tab at the top of this page.

Plagiarism

Group submissions will not be allowed. Your programs must be entirely your own work. Plagiarism detection software will be used to compare all submissions pairwise (including submissions for similar assessments in previous years, if applicable) and serious penalties will be applied, including an entry on UNSW's plagiarism register.

- ***Do not copy ideas or code from others***
- ***Do not use a publicly accessible repository or allow anyone to see your code***

Please refer to the on-line sources to help you understand what plagiarism is and how it is dealt with at UNSW:

- Plagiarism and Academic Integrity
- UNSW Plagiarism Policy Statement
- UNSW Plagiarism Procedure