Objective

- Focuses on Command line arguments, String parsing, Expression evaluation, Random numbers.

Question No. 1

A Double Ended Queue (**Deque (also called as DECK)**) is a queue in which you can insert and remove elements both at the *front* and at the *rear end* of the queue. You task is to create a class named as **DECK**, with the following interface:

Public

- void insertAtTail (T).
- void insertAtHead(T).
- T deleteAtTail( ).
- T deleteAtHead( ).
- bool isEmpty():
- bool isFull():
- int getCapacity()
- int getNoOfElements()
- DECK()
- ~DECK()
- DECK(const DECK<T> &)
- DECK & operator = (const DECK<T> &)

Private

- void resize(int)

Please note that: tail = rear , head = front
The amortized cost of all of your function must be constant.
Note: I doubt that you might do some blunders while implementing this task so examine it thoroughly.

Question No. 2

Write a function, which receives an Infix expression in the form of string and returns the result of expression.
The expression may consists of following operators:

Operators Symbol	Description	Example
-	Unary -	-3+7 = 4
^	power	2^4 = 16
/		
*		
%	Remainder	3%5 = 3
+		
-	Binary -	

Note:

- Operand can be multi digit and fractional as well.
- Precedence and associativity should be considered same as in mathematics.

Example:
23*(-3+5)^(4+1)+6.5 = 742.5

Question No. 3

Write a program subset.cpp that takes a command-line integer *k*, reads in a sequence of words from standard input (*you have to understand command line arguments to accomplish this task*) and prints out exactly *k* of them, ***uniformly at random***. Each item from the sequence can be printed out at most once. You may assume that *k* ≥ 0 and no greater than the number of words on standard input. Subset must complete in linear time in the size of the input. You may use a DECK or a RandomizedQueue as well as a constant amount of extra memory.

Sample Run 1

```
C:/>uniform 8 AA BB BB BB BB BB CC CC
BB
AA
BB
CC
BB
BB
CC
BB
```

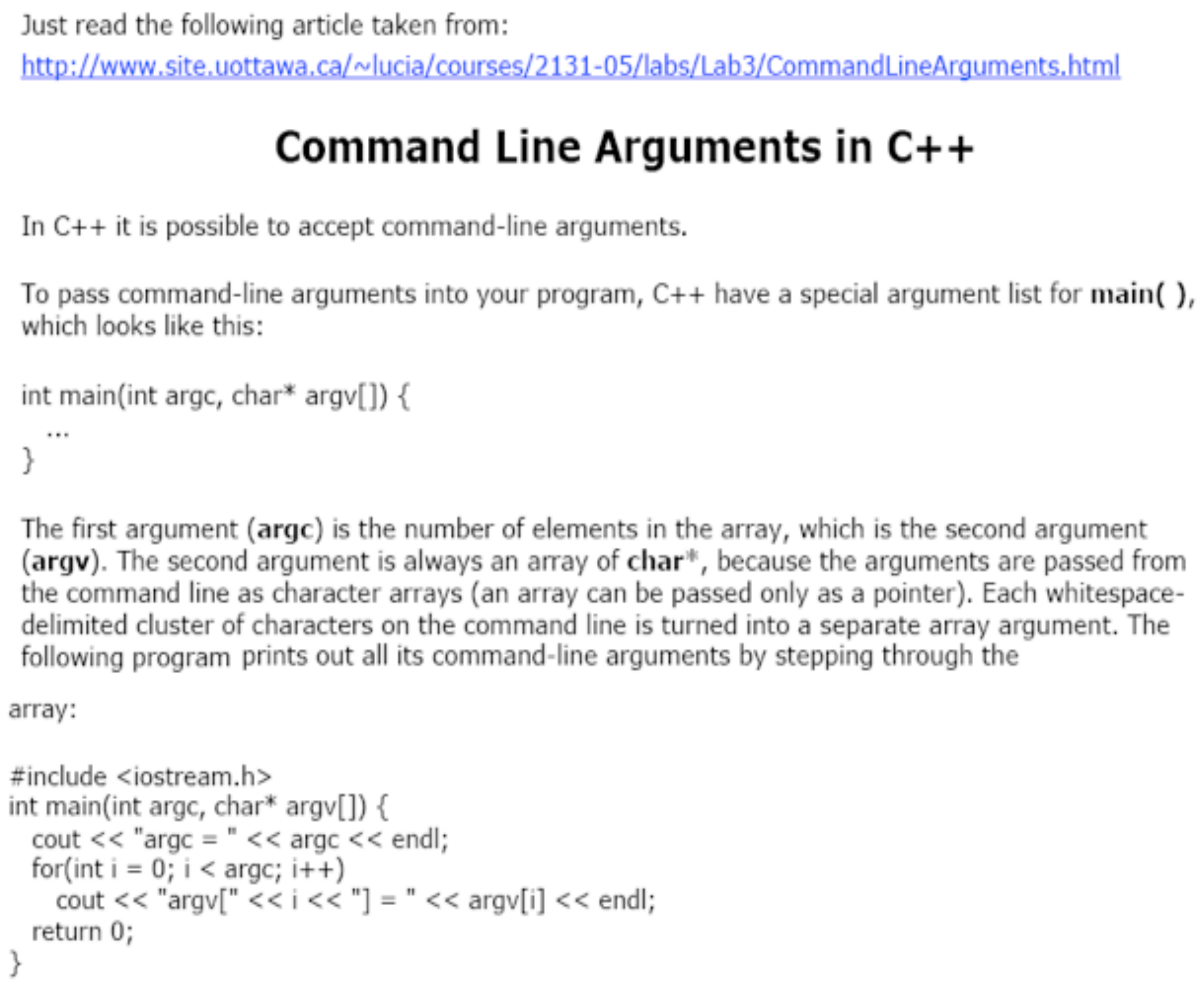
Sample Run 2

```
C:/>uniform 3 A B C D E F G H I
C
G
A
```

Sample Run 3

```
C:/>uniform 3 A B C D E F G H I
E
F
G
```

To understand command line argument, you may consult the following article:



Just read the following article taken from:

http://www.site.uottawa.ca/~lucia/courses/2131-05/labs/Lab3/CommandLineArguments.html

Command Line Arguments in C++

In C++ it is possible to accept command-line arguments.

To pass command-line arguments into your program, C++ have a special argument list for **main( )**, which looks like this:

```
int main(int argc, char* argv[]) {
   ...
}
```

The first argument (**argc**) is the number of elements in the array, which is the second argument (**argv**). The second argument is always an array of **char***, because the arguments are passed from the command line as character arrays (an array can be passed only as a pointer). Each whitespace-delimited cluster of characters on the command line is turned into a separate array argument. The following program prints out all its command-line arguments by stepping through the array:

```
#include <iostream.h>
int main(int argc, char* argv[]) {
  cout << "argc = " << argc << endl;
  for(int i = 0; i < argc; i++)
    cout << "argv[" << i << "] = " << argv[i] << endl;
  return 0;
}
```

You'll notice that **argv[0]** is the path and name of the program itself. This allows the program to discover information about itself. It also adds one more to the array of program arguments, so a common error when fetching command-line arguments is to grab **argv[0]** when you want **argv[1]**.

You are not forced to use argc and argv as identifiers in **main( ),** those identifiers are only conventions (but it will confuse people if you don't use them). Also, there is an alternate way to declare argv:

```
int main(int argc, char** argv) {
  ...
}
```

Both forms are equivalent.

All you get from the command-line is character arrays; if you want to treat an argument as some other type, you are responsible for converting it inside your program. To facilitate the conversion to numbers, there are some helper functions in the Standard C library, declared in **<cstdlib>**. The simplest ones to use are **atoi( )**, **atol( )**, and **atof( )** to convert an ASCII character array to an **int**, **long**, and **double**, respectively. Here's an example using **atoi( )** (the other two functions are called the same way):

```
#include <iostream.h>
#include <stdlib.h>
int main(int argc, char* argv[]) {
  for(int i = 1; i < argc; i++)
    cout << atoi(argv[i]) << endl;
  return 0;
}
```

In this program, you can put any number of arguments on the command line. You'll notice that the **for** loop starts at the value 1 to skip over the program name at **argv[0]**. Also, if you put a floating-point number containing a decimal point on the command line, **atoi( )** takes only the digits up to the decimal point. If you put non-numbers on the command line, these come back from **atoi( )** as zero.

Help →

You are allowed to take any sort of help in understanding/implementing command line arguments but not a bit about rest of the assignment.

How to submit?

1. Follow the same guidelines as you followed in submission of Assignment no. 2.
2. If you have any queries/confusions in submitting the assignment then feel free to discuss but discuss in time.

I hated every minute of training, but I said, "Don't quit. Suffer now and live the rest of your life as a champion.

[Muhammad Ali [Boxer]]