ECE 20010 Data Structures

- *Check your attendance – it matters!*
- ***Homework05***
  - *two assignments*
  - *group assignment*
  - *3 points each*
  - *due: April 5, 11:55 pm*
  - *how to submit – use DropBox*
    - *reduce to upload & download time*
    - *accept the dropbox invitation and install it*

ECE 20010 Data Structures

Data Structures

Chapter 3

- *ADT*
- *stacks & queues using dynamic arrays*
- *some applications*
- ***Homework05***

Chapter 4

- *linked lists*
- *stack & queue*
- *some applications*

Chapter 3 – Stacks and queues

Homework 05 – 1 StackOfIntegers

1. **Complete the StackOfIngers.cpp** given including the following items;
   resize the stack by repeated doubling when the stack is full
   resize the stack by one half when the stack is a quarter full.
   peep() // refer to the source code what to implement
   freeStack() // refer to the source code what to implement

2. Don't change function signatures and/or return types in StackOfIntegers.cpp and StackOfIntegers.h file which you may download from piazza. My test program may call the same function names and arguments with other (big) data set from my own driver program.

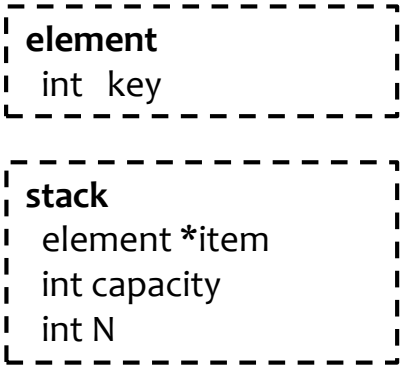
3. StackOfIntegersHWDriver.cpp program is provided for your basic testing.

Chapter 3 – Stacks and queues

Homework 05 – 1 StackOfIntegers

1. **Complete the StackOfIngers.cpp** given including the following items;
   resize the stack by repeated doubling when the stack is full
   resize the stack by one half when the stack is a quarter full.
   peep() // refer to the source code what to implement
   freeStack() // refer to the source code what to implement

```
#ifndef StackOfIntegers_h
#define StackOfIntegers_h
typedef struct element {
  int  key;
  // add other fields as needed
} element;

typedef struct stack {
  element *item;          // this is called "stack" in the "Horwitz" book
  int capacity;           // stack size
  int N;                  // number of items; 0 if empty
} stack;

// function prototypes come here ……
#endif StackOfIntegers_h
```



Chapter 3 – Stacks and queues

Homework 05 – 1 StackOfIntegers

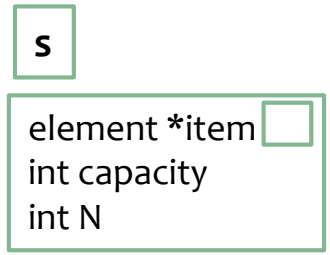
1. **Complete the StackOfIngers.cpp** given including the following items;
   resize the stack by repeated doubling when the stack is full
   resize the stack by one half when the stack is a quarter full.
   peep() // refer to the source code what to implement
   freeStack() // refer to the source code what to implement

```
// StackOfIntegers_h
stack *createStackOfIntegers(int capacity);// stack is created with capacity, top=0

void freeStack(stack *s);      // deallocate items and stack itself
void showStack(stack *);       // print all info about the stack including items
int size(stack *s);            // return nItems
int capacity(stack *s);        // return its capacity (array size)
int pop(stack *s);             // pop the top item in the stack
int peep(stack *s);            // return the top item, no change in stack
int push(stack *s, int key);   // return key if pushed, (key^FFFF)  if failed
int isFull(stack *s);          // return true/false
int isEmpty(stack *s);         // return true/false
char *toString(stack *);       // return a string that contains all items in stack
```

Chapter 3 – Stacks and queues

Homework 05 – 1 StackOfIntegers

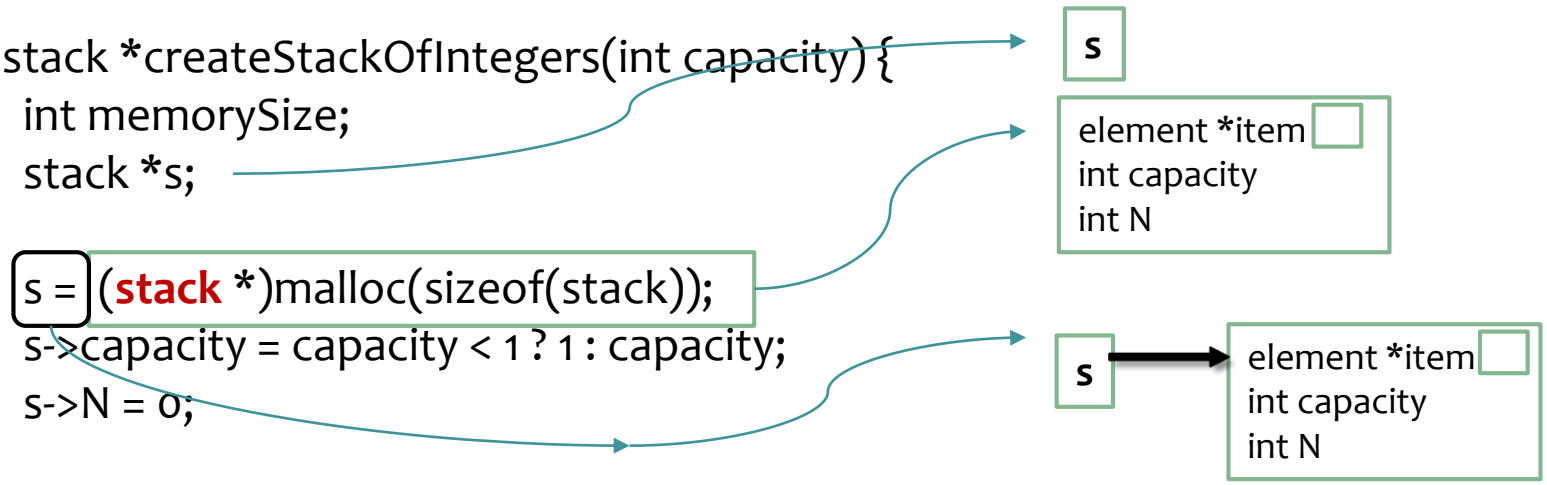
```
/** Create and initializes an empty stack.  */

stack *createStackOfIntegers(int capacity){
  int memorySize;
  stack *s;

  s = (stack *)malloc(sizeof(stack));
  s->capacity = capacity < 1 ? 1 : capacity;
  s->N = 0;

  memorySize = sizeof(element) * s->capacity;
  s->item =  (element *)malloc(memorySize);

  return s;
}
```



Chapter 3 – Stacks and queues

Homework 05 – 1 StackOfIntegers

```
/** Create and initializes an empty stack.  */

stack *createStackOfIntegers(int capacity){
  int memorySize;
  stack *s;

  s = (stack *)malloc(sizeof(stack));
  s->capacity = capacity < 1 ? 1 : capacity;
  s->N = 0;

  memorySize = sizeof(element) * s->capacity;
  s->item =  (element *)malloc(memorySize);

  return s;
}
```



Chapter 3 – Stacks and queues

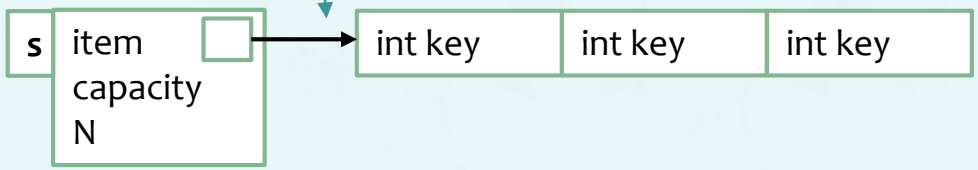
Homework 05 – 1 StackOfIntegers

```
/** Create and initializes an empty stack.  */

stack *createStackOfIntegers(int capacity) {
  int memorySize;
  stack *s;

  s =  (stack *)malloc(sizeof(stack));
  s->capacity = capacity < 1 ? 1 : capacity;
  s->N = 0;

  memorySize = sizeof(element) * s->capacity;
  s->item = (element *)malloc(memorySize);

  return s;
}
```

Chapter 3 – Stacks and queues

Homework 05 – 1 StackOfIntegers

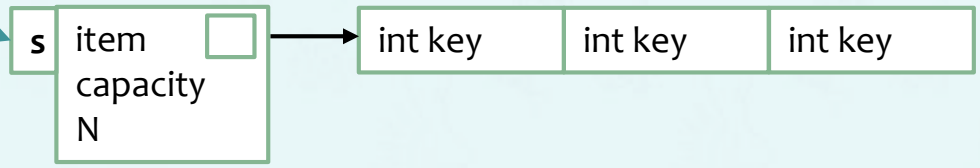
```
/** Create and initializes an empty stack.  */

stack *createStackOfIntegers(int capacity) {
  int memorySize;
  stack *s;

  s =  (stack *)malloc(sizeof(stack));
  s->capacity = capacity < 1 ? 1 : capacity;
  s->N = 0;

  memorySize = sizeof(element) * s->capacity;
  s->item = (element *)malloc(memorySize);

  return s;
}
```



Chapter 3 – Stacks and queues

Homework 05 – 1 StackOfIntegers

```
/** Create and initializes an empty stack.  */

stack *createStackOfIntegers(int capacity) {
  int memorySize;
  stack *s;

  s =  (stack *)malloc(sizeof(stack));
  s->capacity = capacity < 1 ? 1 : capacity;
  s->N = 0;

  memorySize = sizeof(element) * s->capacity;
  s->item =  (element *)malloc(memorySize);

  return s;
}
```



Chapter 3 – Stacks and queues

Homework 05 – 1 StackOfIntegers

```
// StackOfIntegers.cpp
/** deallocate a stack.  */
void freeStack(stack *s) {
  printf("ADD YOUR CODE HERE\n");
}

/** return the stack size N or number of items  */
int size(stack *s) {   return s->N;  }

/** return the stack capacity or number of items  */
int capacity(stack *s) {  return s->capacity;  }

/** Adds the item to this stack. - resizing the stack by repeated doubling when it is full.  */
int push(stack *s, int key) {
  if (isFull(s)) {
    printf("ADD YOUR CODE HERE to double the stack size by reallocating...\n");
  }
  return s->item[s->N++].key = key;
}
```

Chapter 3 – Stacks and queues

Homework 05 – 1 StackOfIntegers

Why do we need these one-line simple functions? Can we just make it when necessary?

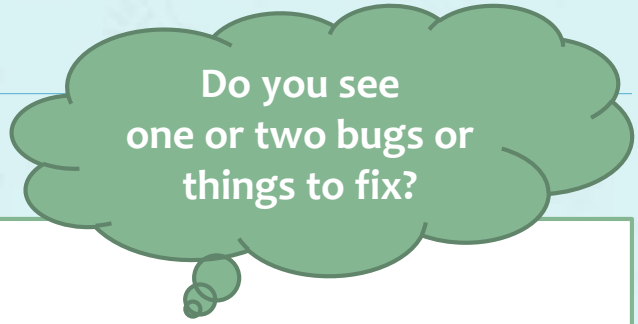
```
// StackOfIntegers.cpp
int peep(stack *s) {
  printf("ADD YOUR CODE HERE ....\n");
  return -1;
}

/** Removes and returns the item most recently added to this stack.  */
int pop(stack *s) {
  if (isEmpty(s)) return emptyStack();
  if (1) {   printf("ADD YOUR CODE HERE\n");    }
  return s->item[- -s->N].key;
}

/** Is this stack empty?  */
int isEmpty(stack *s) {  return s->N <= 0 ? true : false;  }

/**  Is this stack full?  */
int isFull(stack *s) {     return (s->N) == s->capacity ? true : false;  }
```

Chapter 3 – Stacks and queues

Homework 05 – 1 StackOfIntegers

```
// StackOfIntegers.cpp
/** reads all items in this stack into a string   */
char *toString(stack *s) {
  int i;
  char string[16];                  // one element storage in chars
                                    // max int: 10 digits, sign, null, ' '
  int bufferLength;                 // estimated buffer size
  char *buffer;                     // the entire stack items in chars

  if (isEmpty(s)) return "";

  bufferLength = (s->N) * sizeof(char) * 16;
  buffer = (char *)malloc(bufferLength);

  strcpy_s(buffer, bufferLength, "");
  for (i = 0; i < s->N; i++) {
    sprintf_s(string, "%d ", s->item[i].key);
    strcat_s(buffer, bufferLength, string);
  }
  return buffer;
}
```

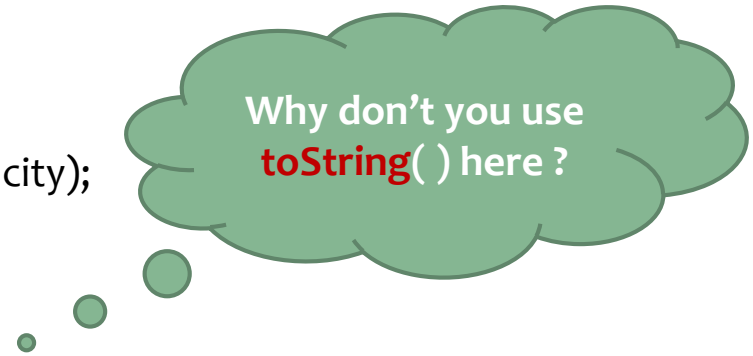
Homework 05 – 1 StackOfIntegers




```
// StackOfIntegers.cpp
/** reads all items in this stack into a string   */
char *toString(stack *s) {
  int i;
  char string[16];                  // one element storage in chars
                                    // max int: 10 digits, sign, null, ' '
  int bufferLength;                 // estimated buffer size
  char *buffer;                     // the entire stack items in chars

  if (isEmpty(s)) return "";

  bufferLength = (s->N) * sizeof(char) * 16;
  buffer = (char *)malloc(bufferLength);

  strcpy_s(buffer, bufferLength, "");
  for (i = 0; i < s->N; i++) {
    sprintf_s(string, "%d ", s->item[i].key);
    strcat_s(buffer, bufferLength, string);
  }
  return buffer;
}
```

Homework 05 – 1 StackOfIntegers

```
// StackOfIntegers.cpp
/** displays all items in this stack */
void showStack(stack *s) {
  int i;
  if (isEmpty(s)) return;

  printf("stack capacity:%d \n", s->capacity);
  printf("stack N items: %d \n", s->N);
  printf("stack elements: ");
  for (i = 0; i < s->N; i++) {
    printf("(%d)=%d ", i, s->item[i].key);
  }
  printf("\n");
}
```



Chapter 3 – Stacks and queues

Homework 05 – 1 StackOfIntegers

```
// StackOfIntegersDriver.cpp
int main(int argc, char *argv[]) {
  stack *s;
  s = createStackOfIntegers(1);                  // begin with 10 or larger during initial testing
  printf("Stack should be empty: pop()=%d\n", pop(s));
  printf("push %d\n", push(s, 1));
  printf("push %d\n", push(s, 2));
  printf("push %d\n", push(s, 3));
  printf("The stack should have 1 2 3. Now the stack has %s\n", toString(s));
  printf("The stack capacity should be 4. Now it is %d\n", capacity(s));
  // some more here
  printf("pop %d\n", pop(s));
  printf("push %d\n", push(s, 9));
  printf("The stack should have 5 6 9. Now the stack has %s\n", toString(s));
  printf("pop %d, pop %d\n", pop(s), pop(s));
  printf("The stack capacity should be 2, Now it is %d \n", capacity(s));

  showStack(s);
  freeStack(s);
  printf("The end of the stack world\n");
}
```

```
printf("pop %d\n", pop(s));
printf("pop %d\n", pop(s));
```

Homework 05 – 2 Evaluate postfix expression

1. Complete EvaluatePostfixHW with the following points in mind;
   You may refer to Program 3.13 to begin with, but it has very significant short comings. Make it work with StackOfIntegers. Don't change function signatures and/or return types in StackOfIntegers.cpp and ~.h file. My test program may call the same function names and arguments with other (big) data set from my driver program.

   Don't use any global variables – I consider that is a significant malpractice as a computer scientist or engineers.
   Make it work with blanks and multiple digits operands. You may assume that the operand and the operator are separated by at least a blank or more. For example, you assume that the postfix expression should be `2 3 4 * + instead of 234*+.`

   HINT: The given program works with **a single digit** and no blanks in postfix expression. You may expand the **getToken( )** to handle multiple digits of operand properly as follows;
   getToken(stack *s, string, &n); // **char string[256];**

2. You test it thoroughly and should describe what does **not** work in the source code document before I or TA find them.

3. EvaluatePostfixHWDriver.cpp program is provided for your basic testing.

Homework 05 – 2 Evaluate postfix expression

```
#ifndef Postfix_h
#define Postfix_h

typedef enum { lparen, rparen, plus, minus, times, divide, mod, eos, operand, blank } precedence;
precedence getPrecedence(char symbol);
precedence getToken(char *expr, char *symbol, int *n);
int EvaluatePostfix(char *postfix);

#endif Postfix_h
```

Homework 05 – 2 Evaluate postfix expression

```
// EvaluatePostfixHW.cpp
precedence getToken(char *expr, char *symbol, int *n) {
  *symbol = expr[(*n)++];
  return getPrecedence(*symbol);
}
```

```
precedence getPrecedence(char symbol) {
  switch (symbol) {
    case '(': return lparen;                    // postfix expr does not have ( ) at all
    case ')': return rparen;
    case '+': return plus;
    case '-': return minus;
    case '/': return divide;
    case '*': return times;
    case '%': return mod;
    case ' ': return blank;
    case '\0': return eos;                      // end of string
    default: return operand;                    // a char such as '5' 'a' ' '
  }
  return operand;
}
```

Homework 05 – 2 Evaluate postfix expression

```
int EvaluatePostfix(char *postfix) {
  char symbol;
  int n = 0, op1, op2;
  stack *s;
  precedence token;
  s = createStackOfIntegers(100);                    // NOTE: Begin with 1 when you test your code
  token = getToken(postfix, &symbol, &n);

  while (token != eos) {
    if (token == operand)   {   push(s, (int)(symbol - '0'));   }
    else {                                           // pop two operands, perform operation, and push result to the stack
      op2 = pop(s);     op1 = pop(s);
      switch (token) {
        case plus:              push(s, op1 + op2);        break;
        case minus:             push(s, op1 - op2);        break;
        case times:             push(s, op1 * op2);        break;
        case divide:            push(s, op1 / op2);        break;
        case mod:               push(s, op1 % op2);        break;
      }
    }
    token = getToken(postfix, &symbol, &n);
  }
  return pop(s);                                     // returns the result from the stack
}
```

Chapter 3 – Stacks and queues

Homework 05 – 2 Evaluate postfix expression

```
void main(int argc, char *argv[]) {
char postfix[256];
   // NOTE: the following three lines are only for initial testing.
   // it should be replaced by the next block #if 0 block code
  strcpy_s(postfix, sizeof(postfix), "234*+");
  printf("postfix = %s\n", postfix);
  printf("result = %d \n", EvaluatePostfix(postfix));
// the following tests should work when you complete it
#if 0
  strcpy_s(postfix, sizeof(postfix), "2 3 4 *  +  ");
  printf("postfix = %s\n", postfix);
  printf("result = %d \n", EvaluatePostfix(postfix));

  strcpy_s(postfix, sizeof(postfix), "6 2 / 3 - 4 2 * +");
  printf("postfix = %s\n", postfix);
  printf("result = %d \n", EvaluatePostfix(postfix));

  strcpy_s(postfix, sizeof(postfix), "1 2 444  *  +  ");
  printf("postfix = %s\n", postfix);
  printf("result = %d \n", EvaluatePostfix(postfix));
#endif
}
```