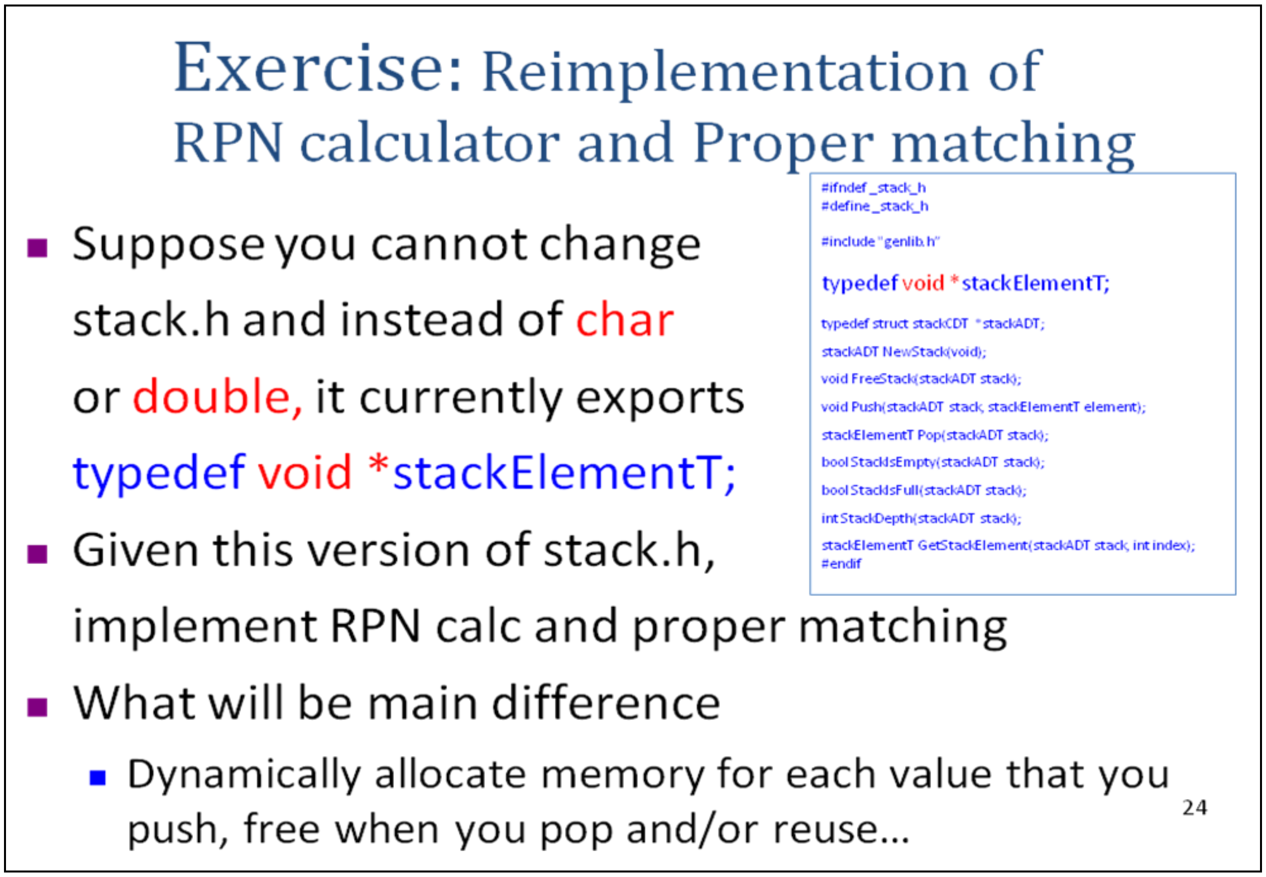
# CS 2213 Advanced Programming Recitation - Exercise

**(Library – Abstract Data Type)**



## Exercise: Reimplementation of RPN calculator and Proper matching

- Suppose you cannot change stack.h and instead of char or double, it currently exports typedef void *stackElementT;
- Given this version of stack.h, implement RPN calc and proper matching
- What will be main difference
  - Dynamically allocate memory for each value that you push, free when you pop and/or reuse…

```
#ifndef _stack_h
#define _stack_h

#include "genlib.h"

typedef void *stackElementT;

typedef struct stackCDT *stackADT;

stackADT NewStack(void);

void FreeStack(stackADT stack);

void Push(stackADT stack, stackElementT element);

stackElementT Pop(stackADT stack);

bool StackIsEmpty(stackADT stack);

bool StackIsFull(stackADT stack);

int StackDepth(stackADT stack);

stackElementT GetStackElement(stackADT stack, int index);
#endif
```

24

In this exercise, you are simply asked to reimplement two driver/client programs: one for RPN calculator and one for proper matching by using stack.h that exports

```
typedef void *stackElementT;
```

---

So you should first copy stack.h, stack.c files from the class web page (follow the link "programs from the textbook" and then go to 08-Abstract-Data-Types). And make sure stack.h has **`typedef void *stackElementT;`** then implement your driver programs….

---

As always, make sure you release (free) the dynamically allocated memories if you allocate any memory in your programs. So, before submitting your program, run it with `valgrind` to see if there is any memory leakage…

---

Also if you need to debug your program, compile your programs with –g option and then run it with `gdb` and/or `ddd`.

/* Don't forget to include comments about the problem, yourself and each major step in your program! */

---

You must submit your work using Blackboard Learn and respect the following rules:

1) All assignments must be submitted as either a zip or tar archive file unless it is a single pdf file.
2) Assignments must include all source code.
3) Assignments must include an output.txt file which demonstrates the final test output run by the student.
4) If your assignment does not run/compile, the output.txt file should include an explanation of what was accomplished, what the error message was that prevented the student from finishing the assignment and what the student BELIEVES to be the underlying cause of the error.

---