CSE410 aka CSE306
Software Quality in Practice

Dr. Carl Alphonce
alphonce@buffalo.edu
343 Davis Hall

http://www.cse.buffalo.edu/faculty/alphonce/SP17/CSE410
https://piazza.com/class/iybn33z3aro2p
compiling and running

without debugger

● compile using gcc, with `-o` flag if you want to specify a name for the resulting executable (other than "a.out")
  
  gcc -o factorial factorial.c

● launch program using by running executable:

  factorial 5

with debugger

● compile using gcc, with `-g` flag to include debugging information in executable (name of executable is up to you, but adding .debug is a reminder that debugging information is included).

  gcc -g -o factorial.debug factorial.c

● launch program using gdb

  gdb factorial.debug

NB: no program argument supplied in gdb invocation
basic commands

- quit - get out of gdb
- help - on-line help system
- run (with program arguments)
short demo

- `bt` (backtrace)
- `up / down / frame N`
- `info frame / info args / info locals`
- `break <function> / break <line> / break <bp> if <expr>`
- `enable / disable`
- `ignore <bp> N`
- `tbreak (a once-only breakpoint)`
- `run / step / continue / next`

[https://www.recurse.com/blog/7-understanding-c-by-learning-assembly](https://www.recurse.com/blog/7-understanding-c-by-learning-assembly)

[https://sourceware.org/gdb/current/onlinedocs/gdb/](https://sourceware.org/gdb/current/onlinedocs/gdb/)
Inspecting/changing variables

- print \(<var> (= <expr>)\)
- set var \(<var> = <expr>\)
- print \(<expr> \rightarrow\) evaluate and print, carrying out function calls
- call \(<expr> \rightarrow\) evaluate, do not print
  returning from a function call
- return \(\rightarrow\) discard frame (and subframes)
- return \(<expr> \rightarrow\) as above, \(<expr>\) is returned
- finish \(\rightarrow\) complete execution of this function normally
- kill \(\rightarrow\) terminate execution of the program being debugged
Define a stack of int.

makeEmptyStack() returns a pointer to a new (empty) stack (e.g. Stack *)

isEmptyStack(Stack *) indicates whether the stack is empty

push(int, Stack *) adds the int to the top of the stack.

pop(Stack *) removes the int at the top of the stack and returns it.
Instructions

- Talk with your group members (group: your PRE Project team or the folks sitting around you) about how to solve this problem.

- Put your laptops away - don't write and compile the code.

- Write down what you talk about on paper. You can write down some code, but you don't need to.
What did groups come up with?
```c
struct Stack{
    int top;
    int array[256];
}

def makeEmptyStack()
    Stack s

push(int n)
    push int n to position top

pop()
    return s[top]
```
group 1

struct

function top

pop();

return s[top]
group 2

```c
struct stack_node {
    stack_node* next;
    int val;
}

struct stack {
    stack_node* top;
    int size;
}

void init_stack(stack* st) {
    st->size = 0;
    st->head = NULL;
}

bool is_empty(stack* st) {
    return st->size == 0;
}

void push(int val, stack* st) {
    stack_node* node = init_node(val, st->head);
    st->head = node;
    st->size++;
}

int pop(stack* st) {
    stack_node* node = st->head;
    st->head = node->next;
    int val = node->val;
    free(node);
    return val;
}
```
group 3

```c
struct Stack {
    int top = -1;
    int size = 0;
    int list[5] = {NULL, NULL, NULL, NULL, NULL};
}

void push (int n, struct Stack* st) {
    st->top++;
    st->list[st->top] = n;
}

int pop () {
    return st->list[st->top--];
}

int isEmptyStack (struct Stack* st) {
    return st->size == 0 ? 1 : 0;
}
```
Fields needed
- maintain "top" counter (keeping track of the first empty index of array)
- easy to implement isEmpty

pop()
- need to ensure not empty (call is empty, throw exception)
- underlying method to resize an array (in push, if full, then resize dynamically)

→ Method to resize an array (in push, if full, then resize dynamically)

Push (if full, keep refreshing)
- using int[]
Not just code: Diagrams and discussion of alternate implementations, possible errors, basic functionality.
Point of exercise

While code may be the deliverable end product, it should not be your first (or only) go-to activity when developing software.

The exercise (happily? sadly?) proved a point: the majority of groups focused on writing code.
Some groups did discuss possible errors.

No group wrote tests for correct functionality.
Error handling

- C does not have an exception mechanism.
- You can roll-your-own: [http://www.on-time.com/ddj0011.htm](http://www.on-time.com/ddj0011.htm)
- You can go the option route: SOME(valid) or NONE