Stack[1]

R. Inkulu
http://www.iitg.ac.in/rinkulu/

[1] C code is developed on the board (and the same is not available in slides)

Introduction

- Motivation for Last-In-Fist-Out (LIFO) data structure:
  - decimal to binary conversion of an integer (ex. $25_{10} \rightarrow 11001_2$)
  - postfix expression evaluation (ex. $6\ 2\ /\ 3\ -\ 4\ 2\ *\ +\ \rightarrow 8$)
  - infix to postfix conversion (ex. $a\ /\ b\ -\ c\ +\ d\ *\ e\ -\ a\ *\ c \rightarrow a\ b\ /\ c\ -\ d\ e\ *\ +\ a\ c\ *\ -$)
  - system stack

- Essential operations[2] to be supported include:

  push an element onto stack

  pop the last element pushed from the stack

  peek to know the topmost element in he stack (peek can be implemented using push and pop though)

[2] *abstract data type* (ADT): data type defined by its behavior from the point of view of a user of the data i.e., range of values, possible operations on data of this type, and the behavior of these operations

Fixed-size stack

```
#define STACKSIZE 1000
typedef struct {...} Data;
typedef struct {
   int topElem;
   Data arrOfObjs[STACKSIZE];
} Stack;
void initialize(Stack *ptrToStack);
int push(Stack *ptrToStack, Data *ptrToData);
  //may return overflow error
int pop(Stack *ptrToStack, Data *p);
  //may return underflow error
int peek(Stack *ptrToStack, Data *p);
```

Fixed-size stack

```
#define STACKSIZE 1000
typedef struct {...} Data;
typedef struct {
   int topElem;
   Data arrOfObjs[STACKSIZE];
} Stack;
void initialize(Stack *ptrToStack);
int push(Stack *ptrToStack, Data *ptrToData);
  //may return overflow error
int pop(Stack *ptrToStack, Data *p);
  //may return underflow error
int peek(Stack *ptrToStack, Data *p);
```

drawbacks:

- unnecessary duplication of objects onto stack
- Stack type need to know the type of the objects' being stored
- all objects on the Stack must be of the same type
- constant size

Fixed-size stack: improved

```
#define STACKSIZE 1000
typedef struct {
   int topElem;
   void *ptrToDatas[STACKSIZE];
      //objects as satellite data
} Stack;
void initialize(Stack *ptrToStack);
int push(Stack *ptrToStack, void *ptrToData);
void *pop(Stack *ptrToStack);
void *peek(Stack *ptrToStack);
```

- a pointer to an object is pushed/popped from the stack
- user of the stack API is the owner of the object

Fixed-size stack: improved

```
#define STACKSIZE 1000
typedef struct {
   int topElem;
   void *ptrToDatas[STACKSIZE];
      //objects as satellite data
} Stack;
void initialize(Stack *ptrToStack);
int push(Stack *ptrToStack, void *ptrToData);
void *pop(Stack *ptrToStack);
void *peek(Stack *ptrToStack);
```

- a pointer to an object is pushed/popped from the stack
- user of the stack API is the owner of the object
- homework: make pop and peek to return error codes

Fixed-size stack: improved

```
#define STACKSIZE 1000
typedef struct {
   int topElem;
   void *ptrToDatas[STACKSIZE];
      //objects as satellite data
} Stack;
void initialize(Stack *ptrToStack);
int push(Stack *ptrToStack, void *ptrToData);
void *pop(Stack *ptrToStack);
void *peek(Stack *ptrToStack);
```

- a pointer to an object is pushed/popped from the stack
- user of the stack API is the owner of the object
- homework: make pop and peek to return error codes

drawbacks:

- constant size

Dynamic-sized stack

```
typedef struct {
   int capacity;
   int topElem;
   void **ptrToDatas;
} Stack;
int initialize(Stack *ptrToStack,
                int initialCapacity);
int destroy(Stack *ptrToStack);
int push(Stack *ptrToStack, void *ptrToData);
void *pop(Stack *ptrToStack);
void *peek(Stack *ptrToStack);
```

Considering time-space tradeoffs, typical strategies in expanding and shrinking the dynamic array:

- when the stack is full, increase its capacity to twice the current capacity
- when only one-fourth of the current capacity is being used, halve the stack capacity

Homework

using dynamic-sized stack implementation, implement the following applications -

- *parenthesis matching*: check the validity of an expression that is parenthesized with various kinds of brackets ex. $\{, (, [,],), \}$
- evaluate a parenthesized infix expression, parenthesized with (and)

Asymptotic time complexity of stack operations

- push: $O(1)$ time
- pop: $O(1)$ time
- peek: $O(1)$ time

homework: analyze the respective space complexities

analysis of dynamic-sized stack is bit involved; hence, will not be presented in this course