Data Structures

- theory that tells us how to best organise data for easy storage and access

---

Arrays

- are data structures
- given a location (index), we can directly go to that location and save or retrieve data
- command line arguments : argv

Stacks

- LIFO: last in first out
- it cannot arbitrarily store elements in any location
- it cannot arbitrarily retreive elements from any location
- pop: retreive an element from top of stack
- push: put an element on top of stack
- by definition, it has no bound on size

```
  10  -5  55  65  __ __ __ __  ........
   0   1   2   3   4   5   6   7  ........
                 ^(top of stack)
```

- array + index_to_top = stack
- intially index_to_top = -1

```
#include<stdlib.h>
#include<stdio.h>

#define SIZE 10

struct stack
{
   int a[SIZE];
   int top;
};

void push(struct stack *ptr , int a);
int pop(struct stack *ptr);
int is_empty(struct stack *ptr);
int is_full(struct stack *ptr);

int main()
{
   struct stack s;
   char c;
   int a;
```

```
    s.top = -1;
    while(1)
    {
        printf("ENTER YOUR CHOICE (P/p/Q): ");
        scanf(" %c",&c);
        if(c == 'P')
        {
            if(is_full(&s) == 1)
            {
                printf("STACK IS FULL\n");
            }
            else
            {
                scanf("%d",&a);
                push(&s,a);
            }
        }
        else if(c == 'p')
            if(is_empty(&s) == 1)
            {
                printf("STACK IS EMPTY\n");
            }
            else
            {
                printf("%d\n" , pop(&s));
            }
        else if(c == 'Q')
        {
            break;
        }
        else
        {
            printf("INVALID INPUT\n");
        }
    }
    return 0;
}

void push(struct stack *ptr , int a)
{
    (ptr->top)++;
    ptr->a[ptr->top] = a;
}

int pop(struct stack *ptr)
{
    int t = ptr->a[ptr->top];
    (ptr->top)--;
    return t;
}

int is_empty(struct stack *ptr)
{
    if(ptr->top == -1)
```

```
    return 1;
  else
    return 0;
}

int is_full(struct stack *ptr)
{
  if(ptr->top == SIZE - 1)
    return 1;
  else
    return 0;
}
```

Parenthesis Matchmaking

- (...), {...}, [...]
- the stack must be empty at the end
- for every closing paranthesis, there must be an opening paranthesis in the stack
- it is a good example of stacks

```
#include<stdlib.h>
#include<stdio.h>

#define SIZE 100

struct stack
{
  char a[SIZE];
  int top;
};

void push(struct stack *ptr , char a);
char pop(struct stack *ptr);
int is_empty(struct stack *ptr);

int main()
{
  struct stack s;
  char str[SIZE];
  char x;
  int i;
  s.top = -1;
  scanf("%s",str);
  for(i=0; str[i]!='\0'; i++)
  {
    if((str[i]=='(') || (str[i]=='[') || (str[i]=='{'))
    {
      push(&s,str[i]);
    }
    else
```

```
        {
            if(is_empty(&s)==1)
            {
                printf("Mismatch!\n");
                break;
            }
            else
            {
                x = pop(&s);
                if(!(((str[i] == ')') && (x == '(')) ||  ((str[i] == ']') && (x == '[')) || ((str[i] == '}') && (x == '{'))))
                {
                    printf("Mismatch!\n");
                    break;
                }
            }
        }
    }
    if(str[i]=='\0')
    {
        if(is_empty(&s)==1)
        {
            printf("Match!\n");
        }
        else
        {
            printf("Mismatch!\n");
        }
    }
    return 0;
}

void push(struct stack *ptr , char a)
{
    (ptr->top)++;
    ptr->a[ptr->top] = a;
}

char pop(struct stack *ptr)
{
    char t = ptr->a[ptr->top];
    (ptr->top)--;
    return t;
}

int is_empty(struct stack *ptr)
{
    if(ptr->top == -1)
        return 1;
    else
        return 0;
}
```