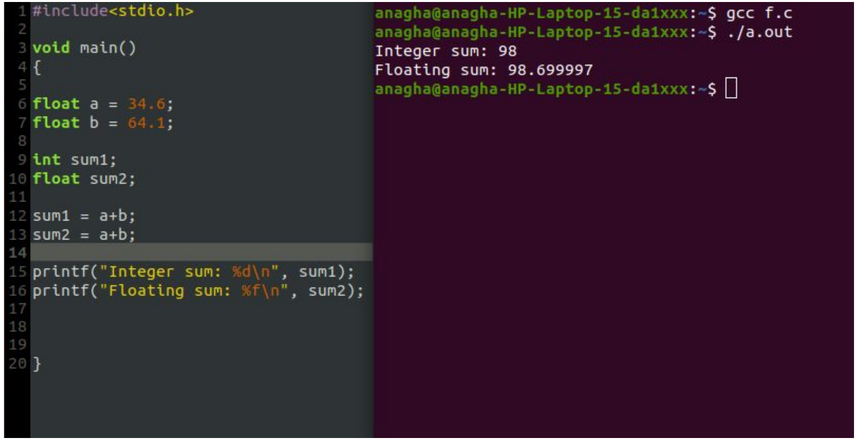
DATA TYPES

ANAGHA SETHU

Why have types..?



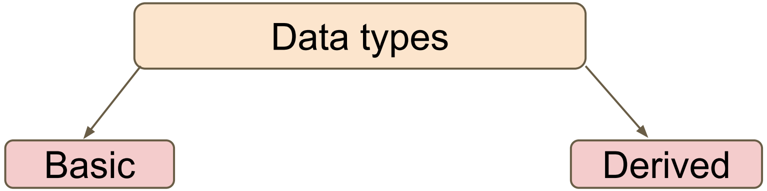
```c
#include<stdio.h>

void main()
{

float a = 34.6;
float b = 64.1;

int sum1;
float sum2;

sum1 = a+b;
sum2 = a+b;

printf("Integer sum: %d\n", sum1);
printf("Floating sum: %f\n", sum2);



}
```

```
anagha@anagha-HP-Laptop-15-da1xxx:~$ gcc f.c
anagha@anagha-HP-Laptop-15-da1xxx:~$ ./a.out
Integer sum: 98
Floating sum: 98.699997
anagha@anagha-HP-Laptop-15-da1xxx:~$ 
```

- Provide context for operations.
- a + b; → what kind addition?
- pointer p = new object → how much space.?

What is data types...?

- ❑ Declaration of variables.
- ❑ Specifies the size and type of data that a variable can store.
- ❑ Example- integer, floating, character, etc.

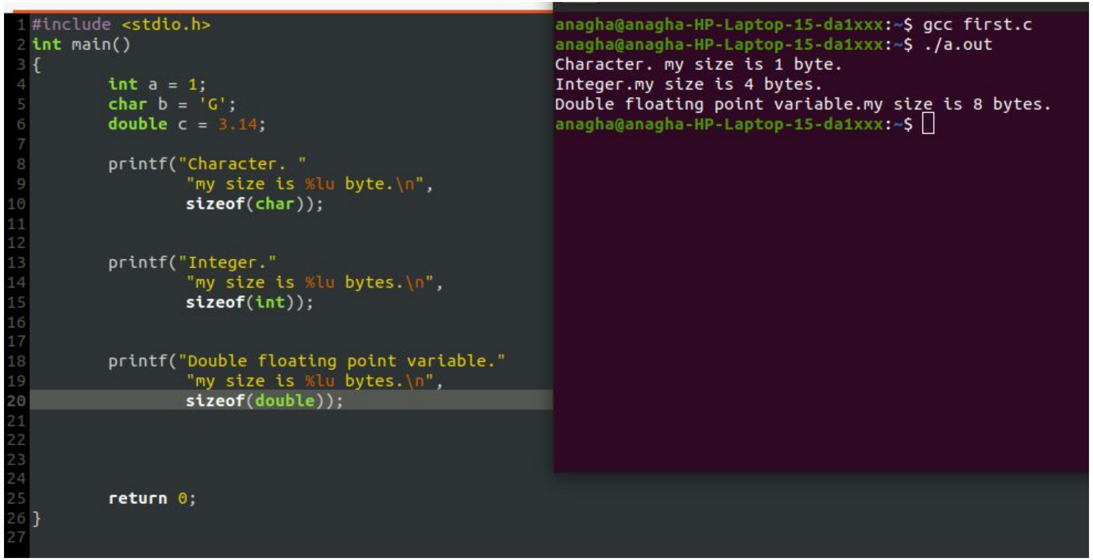

Data types

Basic

Derived

- int
- char
- float
- double

- array
- pointer
- structure
- union

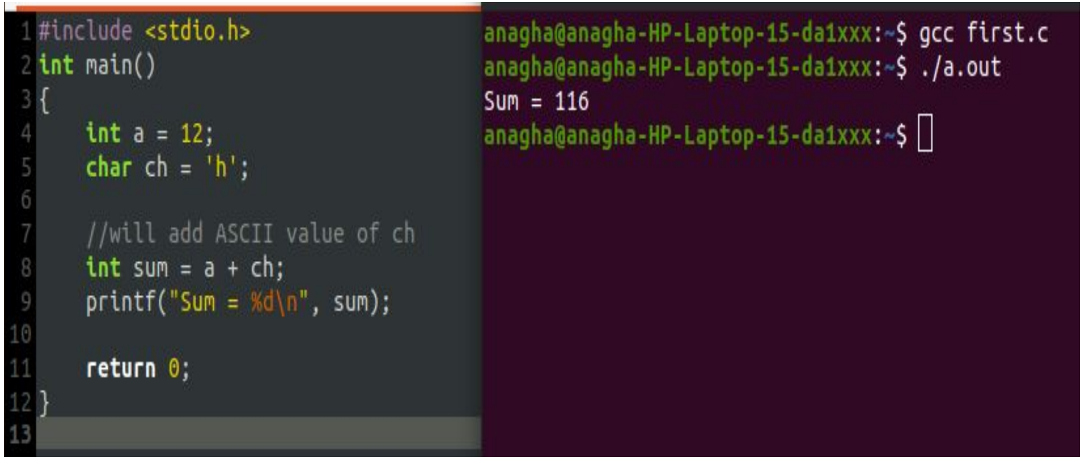


```c
#include <stdio.h>
int main()
{
        int a = 1;
        char b = 'G';
        double c = 3.14;

        printf("Character. "
                "my size is %lu byte.\n",
                sizeof(char));


        printf("Integer."
                "my size is %lu bytes.\n",
                sizeof(int));


        printf("Double floating point variable."
                "my size is %lu bytes.\n",
                sizeof(double));




        return 0;
}

```

```
anagha@anagha-HP-Laptop-15-da1xxx:~$ gcc first.c
anagha@anagha-HP-Laptop-15-da1xxx:~$ ./a.out
Character. my size is 1 byte.
Integer.my size is 4 bytes.
Double floating point variable.my size is 8 bytes.
anagha@anagha-HP-Laptop-15-da1xxx:~$
```

data type	size
char	1
int	4
float	4
double	8

Implicit Conversion



```c
#include <stdio.h>
int main()
{
    int a = 12;
    char ch = 'h';

    //will add ASCII value of ch
    int sum = a + ch;
    printf("Sum = %d\n", sum);

    return 0;
}
```

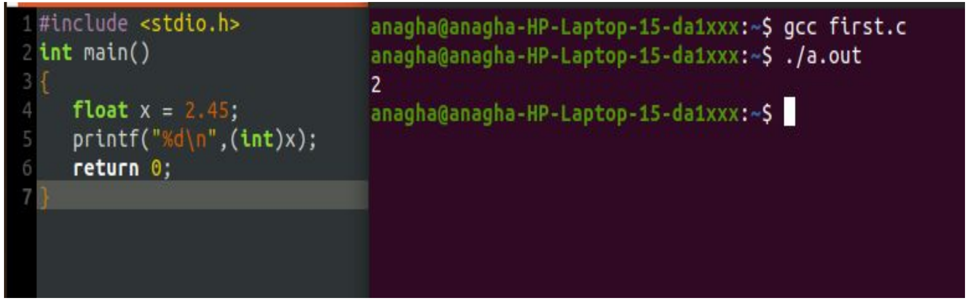
```
anagha@anagha-HP-Laptop-15-da1xxx:~$ gcc first.c
anagha@anagha-HP-Laptop-15-da1xxx:~$ ./a.out
Sum = 116
anagha@anagha-HP-Laptop-15-da1xxx:~$ 
```

34 + ‘e’ → **valid expression**

Automatic conversion of data types.

“All the character variables get converted to integers while performing arithmetic operations or in any other such expression.”

Explicit Conversion



```c
#include <stdio.h>
int main()
{
    float x = 2.45;
    printf("%d\n",(int)x);
    return 0;
}
```

```
anagha@anagha-HP-Laptop-15-da1xxx:~$ gcc first.c
anagha@anagha-HP-Laptop-15-da1xxx:~$ ./a.out
2
anagha@anagha-HP-Laptop-15-da1xxx:~$ 
```

Manually convert values from one data type to another as follows:

(data-type) value ;

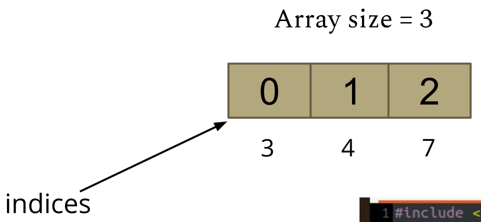
int (41.567) → 41

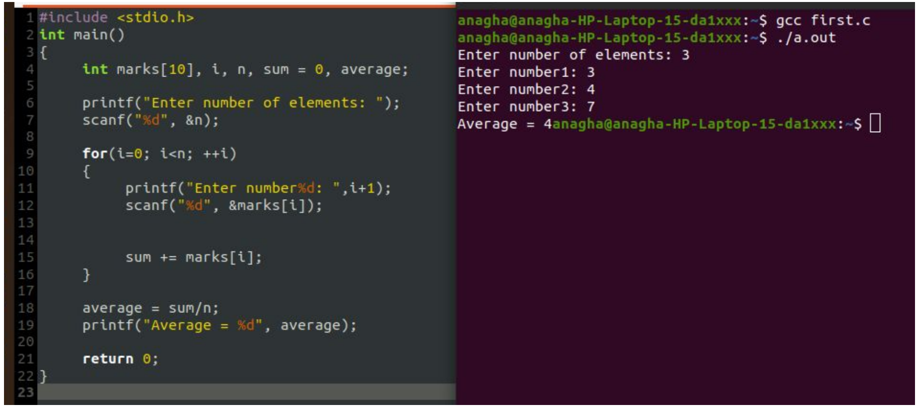
float(41)   → 41.00

Array

Array - variable that can store multiple values.

dataType arrayName[arraySize];

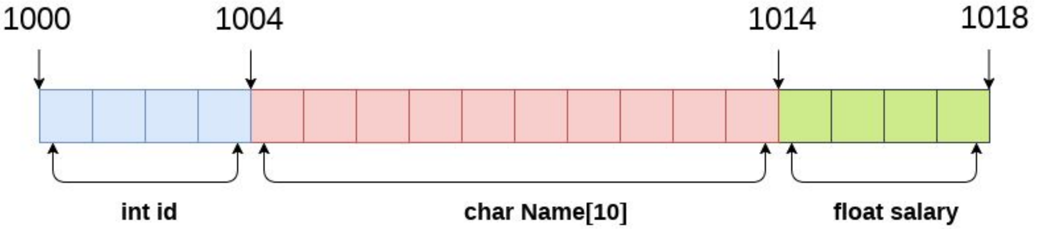




```c
#include <stdio.h>
int main()
{
        int marks[10], i, n, sum = 0, average;

        printf("Enter number of elements: ");
        scanf("%d", &n);

        for(i=0; i<n; ++i)
        {
                printf("Enter number%d: ",i+1);
                scanf("%d", &marks[i]);


                sum += marks[i];
        }

        average = sum/n;
        printf("Average = %d", average);

        return 0;
}
```

```
anagha@anagha-HP-Laptop-15-da1xxx:~$ gcc first.c
anagha@anagha-HP-Laptop-15-da1xxx:~$ ./a.out
Enter number of elements: 3
Enter number1: 3
Enter number2: 4
Enter number3: 7
Average = 4anagha@anagha-HP-Laptop-15-da1xxx:~$
```

Structure



```
struct Employee
{
     int id;
     char Name[10];
     float salary;
} emp;
```

sizeof (emp) =   4 + 10 + 4 = 18 bytes

where;
 sizeof (int) = 4 byte
 sizeof (char) = 1 byte
 sizeof (float) = 4 byte

Structure - Group of variables of different data types represented by a single name.

```
struct struct_name {
  DataType member1_name;
  DataType member2_name;
  DataType member3_name;
  ...
};
```

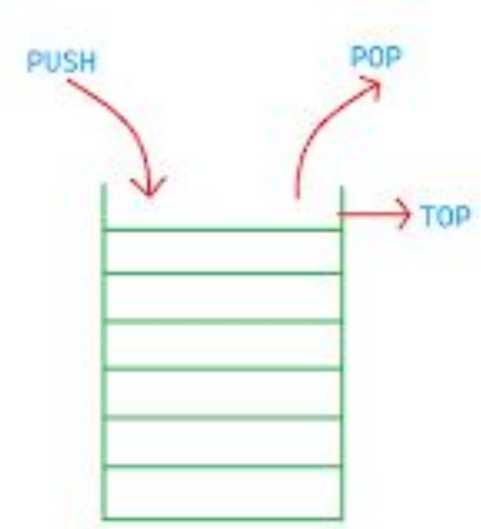
Array vs Structure

ARRAY	STRUCTURE
Data structure consisting of collection of elements each identified by array index.	Stores different data types in the same memory location.
Stores a set of elements of the same data type.	Stores different data types as a single unit.
Possible to access array element using index.	Access property of a structure using structure name dot operator.
No keyword	“Struct” is the keyword
Each element has the same size	Size of elements can differ.
Requires less time to access	Requires more time to access.

STACKS & QUEUES

Stacks

Linear data structure in which insertions and deletions are allowed only at the top.
LIFO data structure → Last In First Out
Non - primitive data structure.



Primary stack operations

- ❑ **push**(**data**)
  Adds an item in the stack.
  If stack is full, then it’s said to be in *overflow* condition.

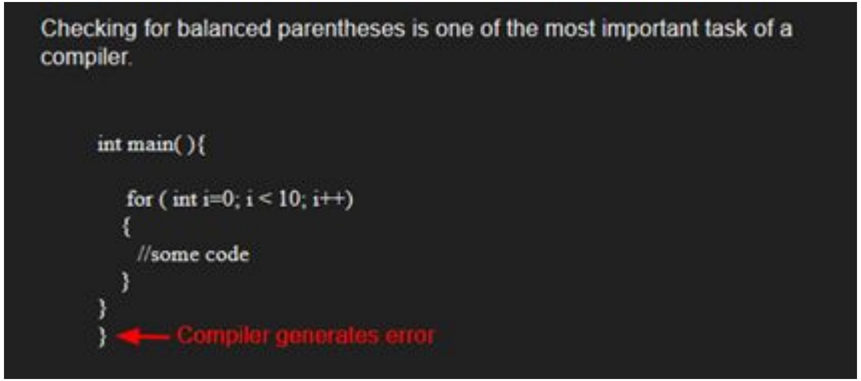
- ❑ **pop**(**data**)
  Removes an item from the stack.
  Items are popped in the reverse order in which they are pushed.
  If the stack is empty, then it is said to be an *underflow* condition.

- ❑ **top**
  Returns top element of the stack.

- ❑ **isEmpty**
  Returns true if the stack is empty , else false.

Application

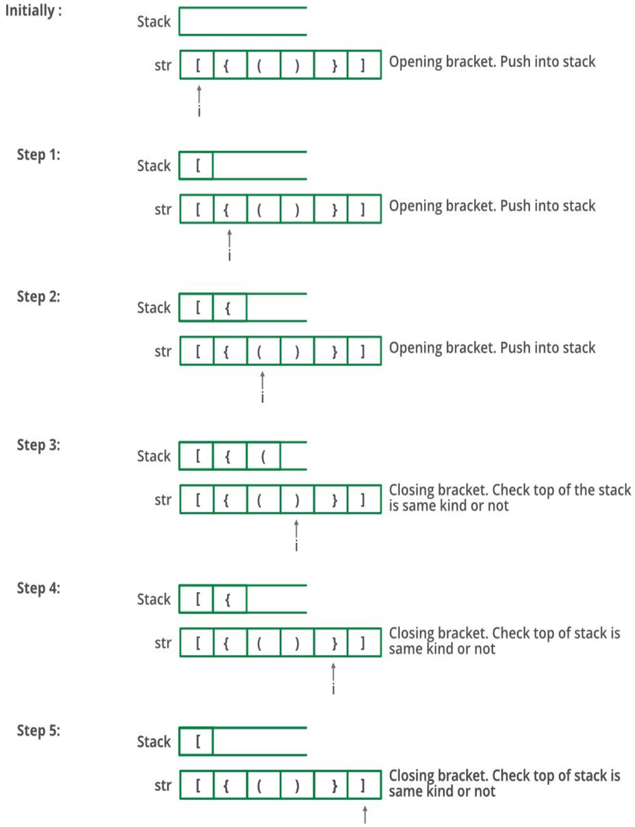
Check for balanced parentheses



Checking for balanced parentheses is one of the most important task of a compiler.

```
int main( ){

    for ( int i=0; i < 10; i++)
    {
      //some code
    }
}
} ← Compiler generates error
```

Check for the expression { ( ) } [ ]



Initially :

Stack

str [{ ( ) }] Opening bracket. Push into stack

i

Step 1:

Stack [

str [{ ( ) }] Opening bracket. Push into stack

i

Step 2:

Stack [{

str [{ ( ) }] Opening bracket. Push into stack

i

Step 3:

Stack [{ (

str [{ ( ) }] Closing bracket. Check top of the stack is same kind or not

i

Step 4:

Stack [{

str [{ ( ) }] Closing bracket. Check top of stack is same kind or not

i

Step 5:

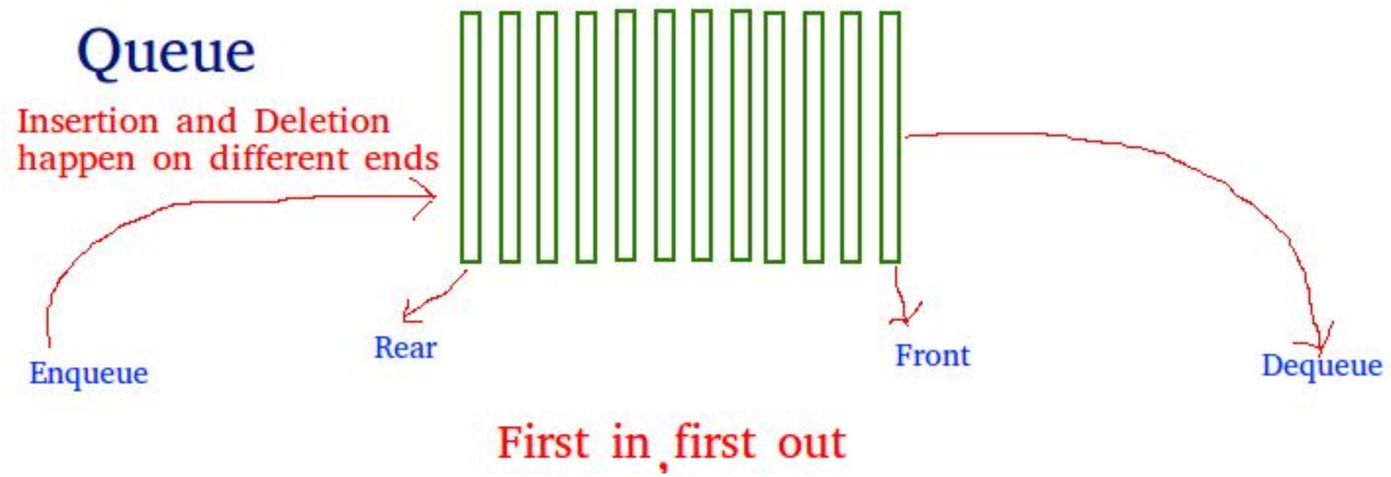
Stack [

str [{ ( ) }] Closing bracket. Check top of stack is same kind or not

- Declare a character stack S.
- Now traverse the expression string exp.
  1. If the current character is a starting bracket (**‘(‘ or ‘{‘ or ‘[‘**) then push it to stack.
  2. If the current character is a closing bracket (**‘)’ or ‘}’ or ‘]’**) then pop from stack and if the popped character is the matching starting bracket then fine else brackets are not balanced.
- After complete traversal, if there is some starting bracket left in stack then “not balanced”

Queue

Queue is a linear data structure, in which insertions and deletions are allowed only in a particular order.
FIFO data structure → First In First Out
Non - primitive data structure.



Primary queue operations

- ❑ **enqueue**
  Adds an item to the queue.
  If queue is full, then it’s said to be in *overflow* condition.

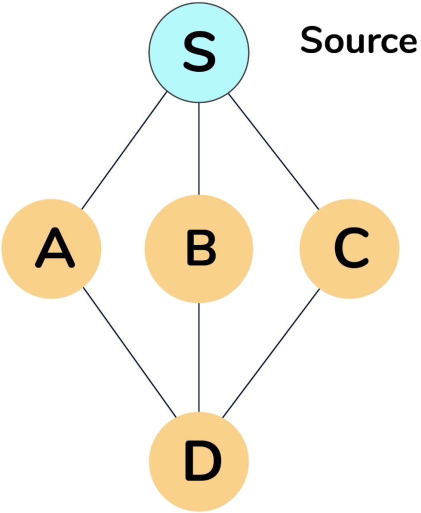
- ❑ **dequeue**
  Removes an item from the queue.
  Items are popped in the same order in which they are pushed.
  If the queue is empty, then it is said to be an *underflow* condition.

- ❑ **front**
  Get the front item of the queue.

- ❑ **rear**
  Get the last item of the queue.

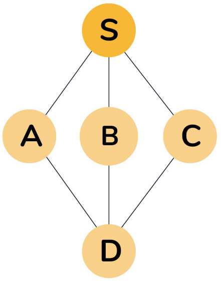
Application

Breadth First Search



Source Node

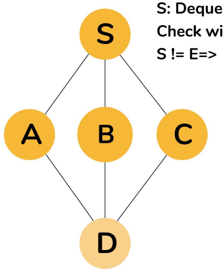
An algorithm for traversing or searching layerwise in tree or graph data structures



Queue

S		

Front

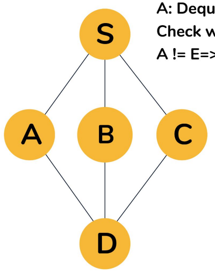


S: Dequeued
Check with Key (E)
S != E=> Proceed by enqueuing immediate neighbours of S to queue

Queue

A	B	C	

Front

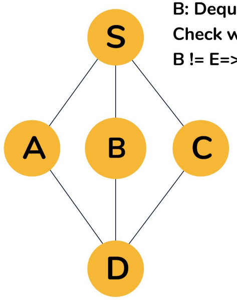


A: Dequeued
Check with Key (E)
A != E=> Proceed by enqueuing immediate neighbours of A to queue (i.e node D)

Queue

B	C	D	

Front

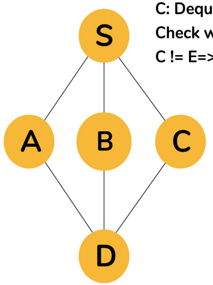


B: Dequeued
Check with Key (E)
B != E=> Proceed by enqueuing immediate neighbours of B to queue
No unvisited neighbours left.

Queue

C	D	

Front

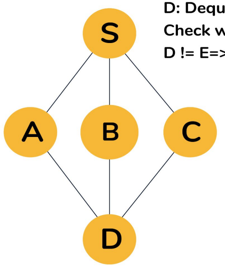


C: Dequeued
Check with Key (E)
C != E=> Proceed by enqueuing immediate neighbours of C to queue
No unvisited neighbours left.

Queue

D	

Front



D: Dequeued
Check with Key (E)
D != E=> Proceed by enqueuing immediate neighbours of D to queue
No unvisited neighbors left.

Queue

Empty

Front

- We enqueue S to the QUEUE.
- Mark S as Visited.
- Now, call the BFS function with S in the queue.
- Dequeue A and check whether A matches the key.
- Dequeue B and check whether B matches the key E.
- Dequeue C and check whether C matches the key E.
- Dequeue D and check whether D matches the key E.
- The queue is empty and there are no unvisited nodes left.

'E' not found