BASICS OF C++

DATATYPES

Type	Keyword
Boolean	bool
Character	char
Integer	int
Floating point	float
Double floating point	double
Valueless	void
Wide character	wchar_t

Combine the types with these if need be

- signed
- unsigned
- short
- long

Type	Typical Bit Width	Typical Range
char	1byte	-127 to 127 or 0 to 255
int	4bytes	-2147483648 to 2147483647
unsigned int	4bytes	0 to 4294967295
short int	2bytes	-32768 to 32767
unsigned short int	2bytes	0 to 65,535
long int	8bytes	-9223372036854775808 to 9223372036854775807
signed long int	8bytes	same as long int
unsigned long int	8bytes	0 to 18446744073709551615
long long int	8bytes	-(2^63) to (2^63)-1
unsigned long long int	8bytes	0 to 18,446,744,073,709,551,615
float	4bytes	
double	8bytes	
long double	12bytes	

DECISION MAKING AND BRANCHING

1. if statement
2. switch statement

ITERATIVE AND LOOPING STATEMENTS

1. for
2. while
3. do-while
4. for-each

```
void decision()
{
    int a,b,c,max;
    cout << "Enter the numbers: " << endl;
    cin >> a >> b >> c;
    if(a > b)
    {
        if(a > c)  max = a;
        else    max = c;
    }
    else
    {
        if(b > c) max = b;
        else    max = c;
    }
    cout << max << endl;
}
```

```
void switchCase()
{
    int ch;
    cout << "Enter the choice: " << endl;
    cin >> ch;
    switch(ch)
    {
        case 1:
            cout << "1 is selected" << endl;
            break;
        case 2:
            cout << "2 is selected" << endl;
            break;
        default:
            cout << "Neither were selected" << endl;
    }
}
```

```cpp
void loops()
{
    int arr[5];
    //for loop
    for(int i = 0; i < 5; i++)
    {
        arr[i] = i;
    }
    cout << endl;
    int i = 0;

    //while loop
    while(i < 5)
    {
        cout << i << " ";
        i++;
    }
    cout << endl;

    //do while loop
    do
    {
        cout << i << " ";
        i--;
    } while (i > 0);
    cout << endl;

    //for each loop
    for(int a: arr)
    {
        cout << a << "  ";
    }
}
```

String Functions

- length(): - Returns the length of the string
- find(): - Returns the position of the first occurrence of a substring in a string.
- rfind(): - Returns the position of the last occurrence of a substring in a string.
- replace(): - Replaces all occurrences of a substring with another substring in a string.
- erase(): - Erases a specified number of characters from a string, starting at a specified position.
- insert(): - Inserts a specified number of characters into a string, starting at a specified position.

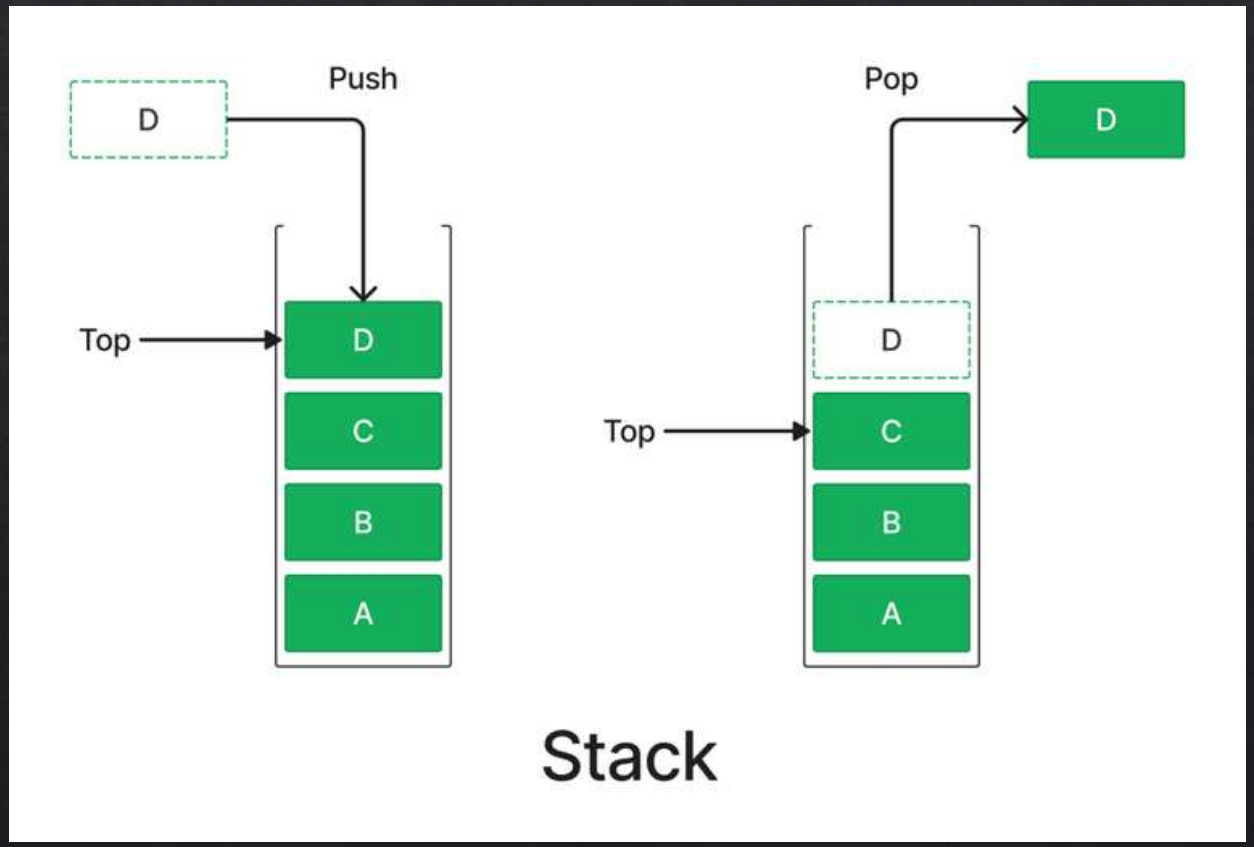
```
void strings()
{
    string s = "hello";
    for(int i = 0; i < s.length(); i++)
    {
        s[i]++;
    }
    cout << s << endl;
    s = "say";
    s.pop_back();
    s.push_back('t');
    cout << s << endl;
    for(auto a = s.rbegin(); a < s.rend(); a++)
    {
        cout << *a;
    }
```

```
    reverse(s.begin(), s.end());
    cout << endl;
    cout << s;
    cout << endl;
    s = "the quick brown fox jumped over the lazy dog";
    cout << s.find("the") << endl;
    cout << s.rfind("the") << endl;
    string str1 = "Let us learn C language";
    string str2 = "C++";
    cout << "The string before replacement is: "<<str1<<'\n';
    str1.replace(13,1,str2);      //13 => 13th index, 1 => 1 char(s) from 13th (inclusive)
                                                replace with str2
    cout << "The string after replacement is: "<<str1<<'\n';
}
```

STL – Standard Template Library

Stack

- LIFO – Last In First Out

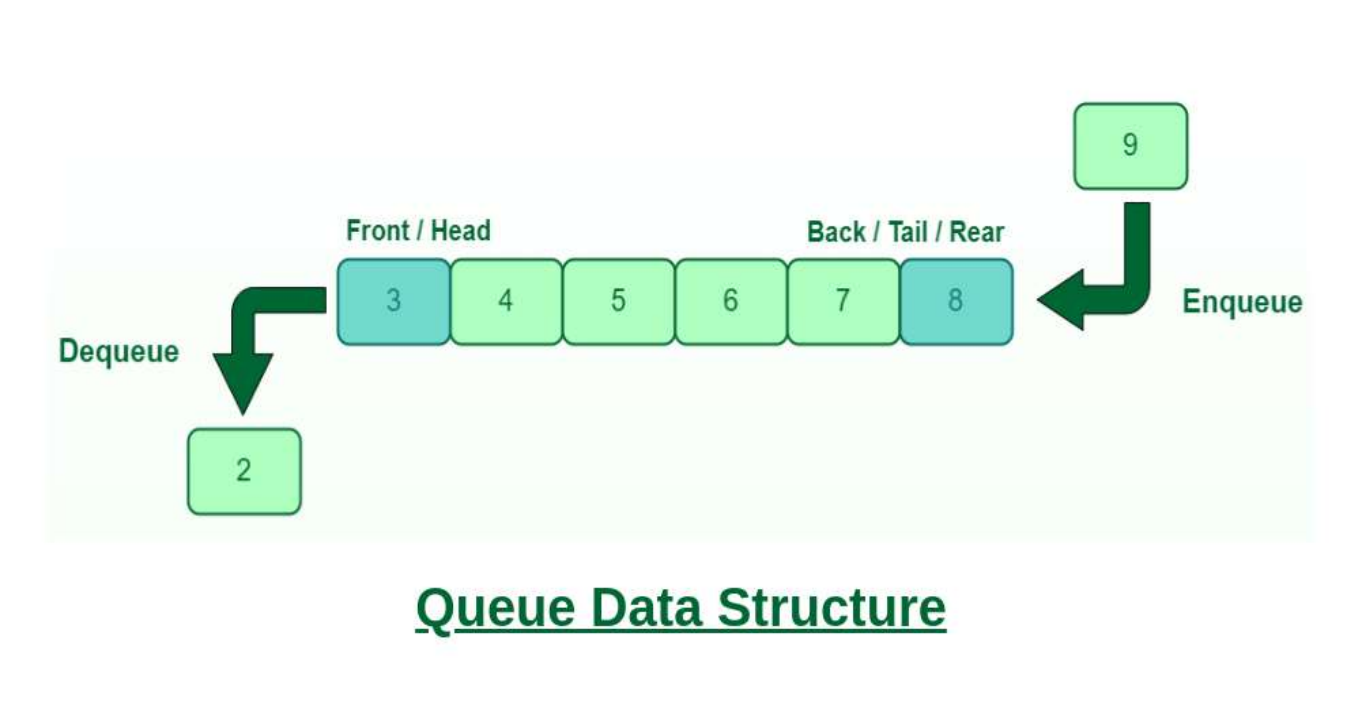



```
struct stack
{
    int top;
    int arr[];
};

void push(int x)

int top()

void pop()
```

```cpp
#include <bits/stdc++.h>      //Used for STL

void explainStack()
{
    stack<int> st;
    st.push(1);
    st.push(2);
    st.push(3);
    st.push(4);
    st.push(5);
    cout << "\n" << st.top() << endl;
    st.pop();
    //also st.size() and st.empty()
}
```

Queue

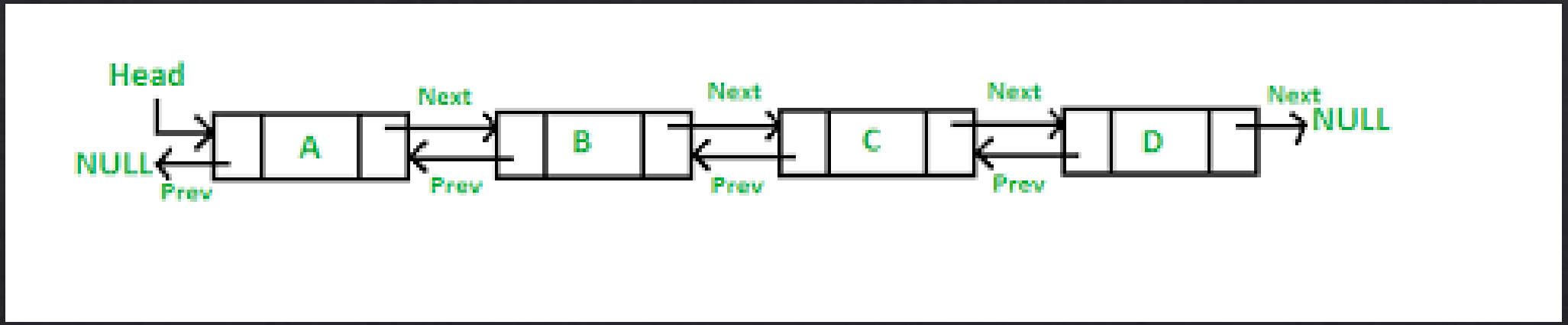
- FIFO – First In First Out




```
struct queue
{
    int front;
    int rear;
    int arr[];
};

void enqueue(); //used as push

void dequeue(); //uses as pop

int front();
int back();
```

```cpp
void explainQueue()
{
    queue<int> q;
    q.push(1);
    q.push(2);
    q.push(3);
    cout << "Front: " << q.front() << endl;
    cout << "Back: " << q.back() << endl;
    // q.front() is the first element and q.back() is the last element
    q.pop();    // pop the first element because FIFO
    cout << "Front: " << q.front() << endl;
    cout << "Back: " << q.back() << endl;
}
```

Lists




```
struct node
{
    node* next;
    node* prev;
    int val;
};
```

```
struct list
{
    node* head;
    int size;
};
```

```cpp
void explainList()
{
    list<int> ls = {1,2,3,4,5};
    ls.push_back(6);    //or emplace
    ls.push_front(0);   //or emplace
    cout << endl;   //doubly linked list to reduced time complexity when compared to insert in vector
    for(int i : ls)
    cout << i;
    cout << endl;
    cout << "Front: " << ls.front() << "\nBack: " << ls.back() << endl;
    //rest similar to vectors
}
```

Vectors

Iterators:

begin(), end(), rbegin(), rend()

Attributes:

size(),front(),back(), []/at()

Modifiers:

push_back(), pop_back(), insert(), erase()

```
void vec()
{
    vector<int> v;
    v.push_back(1);
    v.push_back(2);
    v.push_back(3);
    vector<int>::iterator itn = v.end();
    cout << *(itn-1) << endl;
    vector<int>::reverse_iterator it = v.rbegin();
    cout << *(it) << endl;
    for(int i: v)
    {
        cout << i;
    }
    cout << endl;
    for(vector<int>::reverse_iterator i = v.rbegin(); i < v.rend(); i++)
    {
        cout << *i;
    }
```

```
cout << endl;
    for(int i = 0; i < v.size(); i++)
    cout << v[i];
    cout << endl;

    auto i = find(v.begin(), v.end(), 2);
    cout << i-v.begin();
    cout << "\n" << endl;
    vector<int> v2(10,5);
    for(auto i: v2) cout << i;
    vector<int> v3 = {6,4,2,7,3,1,6,4,0};
    sort(v3.begin(),v3.end(),greater<int>());
    cout << endl;
    for(auto i: v3) cout << i;
    cout << endl;
    v3.insert(v3.begin()+4,9);
    for(auto i: v3) cout << i;
    v3.erase(v3.begin()+2,v3.end()-1);
    cout << endl;
    for(auto i: v3) cout << i;
}
```

Common Algorithms

sort(first_iterator, last_iterator, *, function[if need be]) – Ascending

sort(first_iterator, last_iterator, greater<int>()) - Descending

find(first_iterator, last_iterator, value) – Iterator to first position

reverse(first_iterator, last_iterator) – Reverse

*max_element (first_iterator, last_iterator) – Max Element

*min_element (first_iterator, last_iterator) – Min Element

count(first_iterator, last_iterator,value) – Count occurances of value