Data types in C++:

primitive:

- integer, 4, -35
- float - 5.6, 8.9, -2.1
- character - c, f, @, %
- Boolean - 0, 1

Derived

- Function
- Arrays
- pointer
- Reference

User-defined

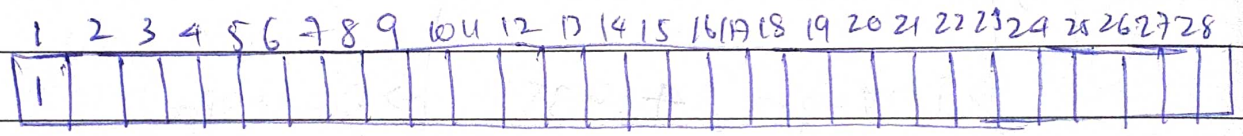
- Class
- Structure
- Union
- Enum

1. Integers: are the whole numbers which are either positive or negative. It requires 4 bytes of memory. 1 Byte = 8 bits

Range of Integers:

*. Unsigned: 0 to $2^{32}-1$.

To store negative numbers the MSB is set to 1 or vice-versa.



MSB

Range (Signed): -2^{31} to $2^{31}-1$.

2. Float: are used to stored real numbers. Its size is 4 bytes. It can store upto 7 decimal digits

3. Double: It can store upto 15 decimal digits Its size is 8 bytes

ASCII char of a is 97. similarly, A is 65

4. char : it requires 1 Byte of memory.

5. Bool : stores two values either true or false
size is 1 byte

-> Type Modifiers : A modifier is used to alter the meaning of the base type so that it more precisely fits the needs of various situations.

Signed	4 bytes	-2,147,483,648 to 2,147,483,647
Unsigned	4 bytes	0 to 4,294,967,295
Long	8 bytes	-9,223,372,036,854,775,808 to +9,223,372,036,854,775,807
short	2 bytes	-32,768 to 32767

preprocessor directive
used to include directive

Header file for taking i/p and printing output

```
#include<iostream>
```

The execution of code begins from main function

```
int main(){
```

Used to display output in Quotation marks

```
std::cout<<"Hello world\n";
```

marks the end of a statement

Adds line break;

```
return 0;
```

exit status of a function

```
}
```

Indicates the start and end of a function

(1) continue statement: is used to skip the current loop iteration. when the continue statement is executed in the loop, the code inside the loop following the continue statement will be skipped & the next iteration of the loop will begin.

(2) break statement: is used to terminate the loop. As soon as break statement is encountered from within a loop, the loop iterations stops there and control returns from the loop immediately to the first statement after the loop.

Operators:

1. Arithmetic operators:
   - → Binary operators (+, -, *, /, %)
   - → Unary operators (++, --)

* in Binary operator, they requires two operands to operate on. Similarly, Unary operators works only on a single operand.

⇒ pre-increment: it first increments the values & then assigns / use that value

ex: ① i = 1;
i = i++ + ++i;
i = 1 + 3 = 4 //

② i = 1;
j = 1;
1 + 1 + 1 + 1 + 3 + 3
k = i + j + i++ + j++ + ++i + ++j;

i = ? ; j = ? ; k = ?

k = 10 ; i = 3 & j = 3 //

③ i = 0
i = i++ - --i + ++i - i--;
0 - 0 + 1 - 1
i = 0 //

④ i = 1 ; j = 2 ; k = 3

m = i-- - j-- - k--;
= 1 - 2 - 3
= -4 //

i = 0
j = 1
k = 2

(5) i=10; j=20
K= i-- - -i++ + --j - ++j + --i - j-- + +++i - j++;
= 10 - 9 + 19 - 20 + 9 - 20 + 10 - 19
j=20; i=10; K=-20 //

(6)

2. Relational operators: also known as comparison operators. Defines a relation between 2 operands It returns a boolean value

* == : returns true if both operands have equal value.
* != :
* >
* <
* <=
* >=

3. Logical operators: used to connect multiple conditions/expressions together.

* && : AND gives true if both operands are true
* || : OR gives true if atleast one of the operand are true.
* ! : NOT gives the opposite logical value.

4. Bitwise operator:

i. & : AND operator

```
  0101
& 0110
------
  0100
```

ii. | : OR operator

```
  0101
| 0110
------
  0111
```

iii. ^ : XOR operator

```
  0101
^ 0110
------
  0011
```

iv: ~ : ones complement

```
~0110        ~0101
------       ------
 1001         1010
```

v. << left shift operator

```
4<<1
(0100)<<1

(1000)
```

vi. >> Right shift operator

```
4>>1
(0100)>>1

(0010)
```

a<<n	a*2^n
a>>n	a/2^n.

5. Assignment operators: =, +=, -=, *=, /=, %=

6. Miscellanous operators:

* sizeof(): Returns the size of a variable.
* Ternary or conditional operator.
  condition ? X : Y
  The above equation returns the value of X if the condition is true or else the value of Y
* cast: converts one data type to another.
* &: Address or Reference operator. Returns the address of a variable.
* *: Pointer variable.

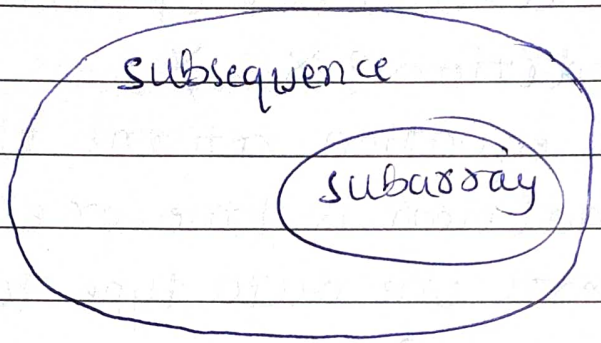
Subarray: continuous part or sequence of the array.

* Number of subarrays of an array with n elements = $n*(n+1)/2$ or nC_2+n

Subsequences: is a sequence that can be derived from an array by selecting zero or more elements, without changing the order of the remaining elements.

* Number of subsequences of an array with n elements = 2^n.

Note: Every subarray is a subsequence but every subsequence is not a subarray.

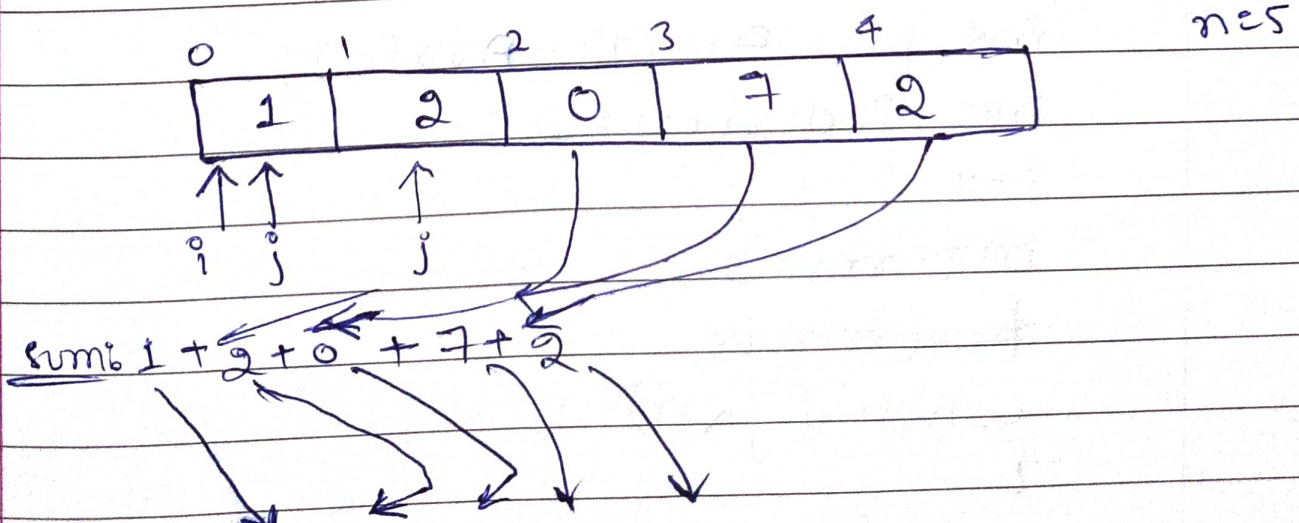


Constraints:

* In 1 sec = 10^8 operation (Approx) can be executed.

Sum of all subarrays:

```
for (int i=0; i<n; i++)
{
    sum=0;
    for (int j=i; j<n; j++)
    {
        sum += Arr[j];
        cout << sum << "\t";
    }
}
```



Output: 1 3 3 10 12

→ Longest Arithmetic Subarray: An arithmetic array is an array that contains atleast two integers and the differences between consecutive integers are equal.

```
int curr_max = INT_MIN;
for (int i=0; i<n; i++)
{
    if (Arr[i] > A[i+1] && Arr[i] > curr_max)
    {
        res++;
    }
    curr_max = max(curr_max, Arr[i]);
}

int pd = Arr[1] - Arr[0];
int indexLen = 2;
int j = 2;
int res = 2;
while (j<n)
{
    if (pd == Arr[j] - Arr[j-1])
    {
```

```
        indexlen++;
      }
      else
      {
        pd = Arr[j] - Arr[j-1];
        indexlen = 2;
      }
      res = max(indexlen, res);
      j++;
    }
    cout << res << endl;
}
```

-> Remove Duplicates from an array Without extra space or in constant space

```
j=0;
for (int i=0; i<n-1; i++)
{
    if (Arr[i] != Arr[i+1])
        Arr[j++] = Arr[i];
}
A[j++] = A[n-1];
```