C++ Data Types

DataType	Meaning	Size (in Bytes)
int	integer	2 or 4
float	floating-pt	4
double	double FP	8
char	character	1
wchar_t	wide char	2
boolean	bool	1
void	empty	0

Integer

~~int~~ = 4 bytes
1 byte = 8 bits
4 bytes = 32 bits

from 0th bit to 31st bit (right to left)

31st bit = sign (+ or −)
0th – 30th bits = number

IN comp, you write with binary
no. of ways one can fill these 31 bits to make different numbers = 2^{31} ways
in the 31st bit, 0 = '+' and 1 = '-ve'
the range will be from most -ve no to most +ve

most negative num is 1 (31st bit) followed by 31 zeros = −2147483548

most positive num is 0 (31st bit) followed by 31 ones = +2147483548

∴ the range of values goes from −2147483548 to +2147483548
Therefore, 4 BYTES CAN STORE VALUES FROM THIS RANGE

C++ Float and Double

- float = 4 bytes
- double = 8 bytes
  ⇒ double has 2x precision for float

```
// creating float type variables
float num1 = 3.0f;
float num3 = 3E-5f; // 3*10^-5

// creating double type
double num4 = 3.0;
double num6 = 3E-5;
```

NOTE:
we must add the suffix f or F at the end of a float value, else compiler interprets it as a 'double'.

ex: float a = 5.6;
we have assigned a double value to a float variable.

NOTE:
4 Byte = 32 bits → 1 bit for sign
→ 8 bits for exponent
→ 23 bits for fraction/actual digits

each bit = binary digit
~~each binary digit~~ = 0.3 decimal digits of info
∴ 23 bits × 0.3 ≈ 6.9 decimal digits
≈ 7 decimal digit of precision

∴ A float can store any number, big or small, but it can only store the first 7 decimal digits accurately

NOTE:
Always use double instead of float as float is prone to inaccuracy errors when working with large numbers

setprecision()

- under iomanip header file for i/o manipulation
- specify no. of decimal places to print in cout

```
#include <iostream>
#include <iomanip>
using namespace std;
int main() {
    double a = 3.19234823993;
    float b = 3.19234823993F;

    // set precision
    cout << setprecision(13);
    cout << "Double" << a << endl;
    cout << "Float" << b << endl;
    return 0;
}
```

output: Double 3.19234823993
Float 3.192378227046

since float has a precision of upto 7 digits, it shows garbage values after its precision is exceeded.

long double = 12 bytes

C++ char:

char = 1 byte

```
char ch = 'h';
cout << 'ASCII value' << int(ch) ; #104
char ch= 104;
cout << 'character' << ch ;  # h
```

NOTE:
++i and i++ is the same in a for loop

C++ Constants

NOTE: Constants must be in uppercase.

```
const double PI = 3.14;
const int MAGIC_NUM = 42;
```

note: const double PI; not allowed as a value is not assigned.

C++ Input/Output

```
int num;
cout << "Enter num : ";
cin >> num;
cout << "The num is : " << num;
return 0;
```

multiple inputs

```
char a;
int num;
cout << "Enter a char & an int : ";
cin >> a >> num;

cout << "char" << a << endl;
cout << "num" << num;
```

NOTE:

7/2 = 3
7.0/2 = 3.5
7/2.0 = 3.5
7.0/2.0 = 3.5

} if either dividend or divisor is a floating point number, we get the result in decimals.

Other C++ Operators

sizeof	returns the sizeof data type	sizeof(int);
?:	returns value based on cond"	(5>0)? "even" : "odd";
&	represent memory address	#
.	access members of struct variable or class objects	s1.marks = 92;
->	used with pointers to access the class or struct variable	ptr -> marks = 92;
<<	print to the o/p value	cout << 5;
>>	gets the i/p value	cin >> num;

GoTo Statement

```
int main()
{   float num, avg, sum=0.0;
    int i,n;

    cout << "Max no. of inputs : ";
    cin >> n;

    for (i=1; i<=n; ++i)
    {   cout << "Enter n" << i << " : ";
        cin >> num;

        if (num <0.0)
        {
            goto jump; // control of the program move to jump
        }
        sum += num;
    }

    jump:
        avg = sum /(i-1);
        cout << "\nAverage = " << avg ;
        return 0;
}
```

Switchcase:

```
#include <iostream>
using namespace std;

int main()
{   char oper;
    float num1, num2;
    cout << "Enter operator (+,-,*,/): ";
    cin >> oper;
    cout << "Enter 2 num: " << endl;
    cin >> num1 >> num2 ;

    switch (oper)
    {   case '+':
            cout << num1 + num2;
            break;

        case '-':
            cout << num1-num2;
            break;

        case '*':
            cout << num1 * num2;
            break;

        case '/':
            cout << num1/num2;
            break;

        default:   // doesn't match any case
            cout << "No";
            break;
    }
    return 0;
}
```

C++ Ternary Operators

```
string result = (marks >=40) ? "passed" : "failed";

string result;
result = (number ==0) ? "Zero" : ((number >0) ? "+ve" : "-ve");
```

C++ Functions

```
void greet() {
    cout << "Hello World";
}

void displayNum ( int n1 , float n2)
{     cout << "The int num is = " << n1;
      cout << "The double num is = " << n2;
}
int main()
{     int num1 = 5;
      double num2 = 5.5;
      displayNum (num1, num2);
      return 0;
}
```

Function Prototype :

If func" difinition is written after the func" call, use function prototype

```
ex: #include <iostream>
    using namespace std;

    // function prototype
    void add (int, int);

    int main()
    {   add (5,3);  # func" call before declaration
        return 0;
    }

    void add (int a, int b)  # func difinition
    {   cout << (a+b); }
```

C++ Libraries :

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{     double number, squareRoot;
      number = 25.0;
      squareRoot = sqrt (number);
      cout << squareRoot ;
      return 0;
}
```

C++ Default Arguments / Parameters

case 1 : no arg is passed

```
void temp ( int = 10, float = 8.8);
int main()
{   temp();
}
void temp (int i , float f){ //code }
```

case 2 : first argument is passed

```
void temp ( int =10 , float = 8.8);
int main() {
    temp(6);
}
void temp ( int i, float f){ //code }
```

case 3 : all arguments are passed

```
void temp ( int = 10, float = 8.8);
int main() {
    temp (6, -2.3);
}
void temp ( int i, float f) { //code }
```

case 4 : second arg is passed

```
void temp ( int =10, float = 8.8);
int main() {
    temp (-2.3);
}
void temp ( int i, float f) { //code }
```

error

C++ Function Overloading

In C++, overloaded func^ are func^ which have the same name but different number / type of arguments

ex:
```
int test () {}
int test ( int a) {}
float test ( double a) {}
int test ( int a, double b) {}
```

NOTE:
overloaded func^ may or may not have diff return types but MUST HAVE DIFFERENT arguments

```
// error code
int test ( int a) {}
double test ( int b) {}
```

C++ Operator Overloading

similar to function overloading, except with the addition of the operator keyword followed by the operator symbol

Syntax:
```
returnType operator symbol (arguments) {
    ...
}
```

[NOTE: read later after learning abt classes n stuff]

C++ Inline Functions

- This copies the func^ to the location of the func^ call in compile-time and may make the program execution faster

syntax:
```
inline returnType functionName (parameters) {
    // code
}
```

how this program works:

```
inline void displayNum ( int num) {
    cout << num << endl;
}
```

```
int main()
{
    displayNum (5);
    displayNum (8);
    displayNum (666);
}
```

compilation →

```
int main()
{
    cout << 5 << endl;
    cout << 8 << endl;
    cout << 666 << endl;
}
```