ALPHA INTERN

BAAP OF ALL DSA SERIES

NOTES on FUNDAMENTAL OF C++

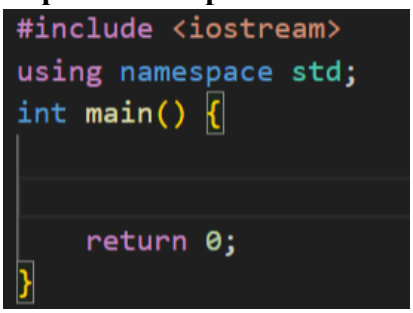
alphaintern.in

Boiler Plate in C++

Boiler Plate in C++?

The term boilerplate refers to sections of code that are typically repeated with little or no modification across many parts of a program. These sections include necessary imports, standard declarations, and function structures that all programs tend to include.

Sample of boilerplate



Key Components of Boilerplate in C++

1) Include Directives: These are used to import standard or custom libraries into your program so that you can use predefined functions, classes, and objects.
   Example: #include <iostream> includes the standard input/output stream library to allow the program to use cin (input) and cout (output).

2) Namespace Std : In C++, the std namespace is a collection of standard library components, such as functions, data types, and objects.

3) Main Function: This is the entry point of any C++ program. When the program starts, execution begins from the **main()** function.

Variables: A variable is like a container that holds a value. The value stored in a variable can change (or vary) as the program runs.

Example: int age = 25;

Explanation:

- int: This tells the computer that the type of data we want to store is an **integer** (a whole number).
- age: This is the **name** of the variable. You can name a variable almost anything you like, like age, height, or score.
- = 25: This assigns the value **25** to the variable age. This means that age now holds the number 25.

Constants:

A constant is similar to a variable, but once a constant is given a value, it cannot be changed during the program.

```
const float pi = 3.14;
```

Explanation:

- const: This keyword tells the computer that the value of this variable cannot be changed.
- float: This tells the computer that the type of data we're storing is a floating-point number (a number with a decimal point).
- pi: This is the name of the constant.
- = 3.14: This assigns the value 3.14 to pi. Once this is done, you cannot change pi to any other value in the program.

Rules for variable nomenclature:

- * Variable Names Must Start with a Letter or Underscore (_)

Ex: Correct

```
int age;
int _score;
```

* It cannot start with a number.

```
int 1age;
```

* You can’t use a keyword as a variable name.

```
int if;
int for;
```

* Names cannot contain whitespaces or special characters like !, #, %, etc. except underScore(_).

```
int %weight;
int #avg;
```

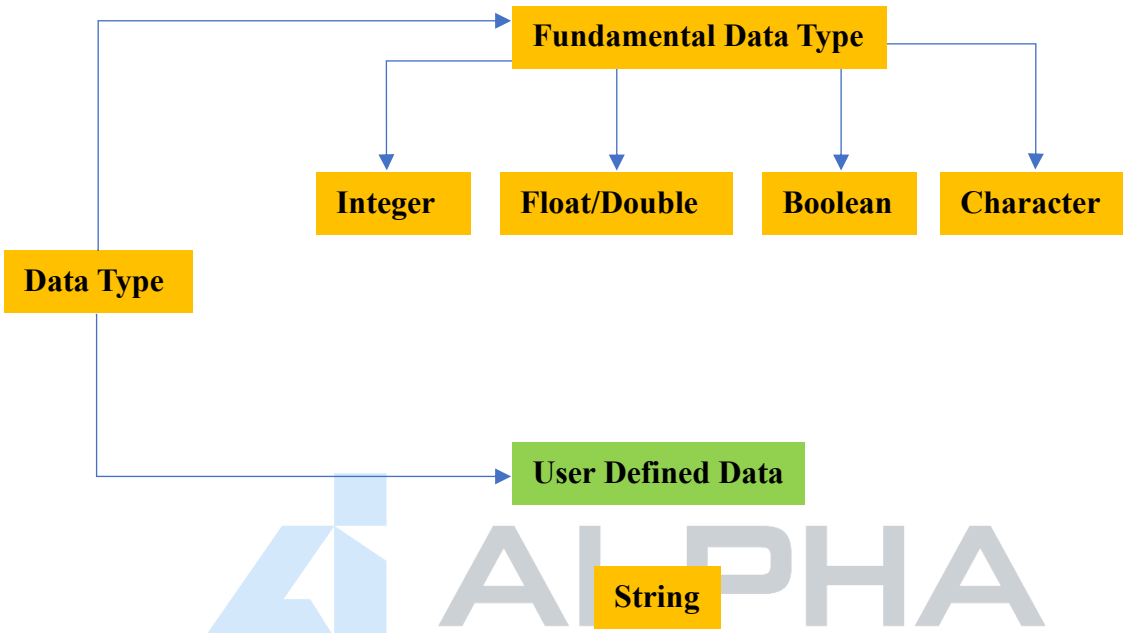
* White Space is not allowed between the variable name.

```
int avg of five;
```

Data Types in C++

Data Types in C++:

Data types define what kind of data a variable can hold. They specify the size and type of data that can be stored in a variable



Range and size of a dataTypes

Type	Bits	Range
int	16	-32768 to -32767
unsigned int	16	0 to 65535
signed int	16	-31768 to 32767
short int	16	-31768 to 32767
unsigned short int	16	0 to 65535
signed short int	16	-32768 to -32767
long int	32	-2147483648 to 2147483647
unsigned long int	32	-2147483648 to 2147483647
signed long int	32	0 to 4294967295
float	32	3.4E-38 to 3.4E+38
double	64	1.7E-308 to 1.7E+308
long double	80	3.4E-4932 to 3.4E+4932
char	8	-128 to 127
unsigned char	8	0 to 255
signed char	8	-128 to 127

1. **Integer Types**

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

int main() {
    int num1 = 10;
    short num2 = 5;
    long num3 = 100000;
    long long num4 = 1000000000;

    cout << "int: " << num1 << endl;
    cout << "short: " << num2 << endl;
    cout << "long: " << num3 << endl;
    cout << "long long: " << num4 << endl;

    return 0;
}
```

Output:

```
int: 10
short: 5
long: 100000
long long: 1000000000
```

2. **Floating-Point Types**

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

int main() {
    float f = 5.75;
    double d = 3.14159;
    long double ld = 1.6180339887;

    cout << "float: " << f << endl;
    cout << "double: " << d << endl;
    cout << "long double: " << ld << endl;

    return 0;
}
```

Output:

```
float: 5.75
double: 3.14159
long double: 1.61803
```

3. **Character Types**

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

int main() {
    char ch = 'A';
    wchar_t wch = L'我';

    cout << "char: " << ch << endl;
    cout << "wchar_t: " << wch << endl;

    return 0;
}
```

Output:

```
char: A
wchar_t: 25105
```


4. **Boolean Type**

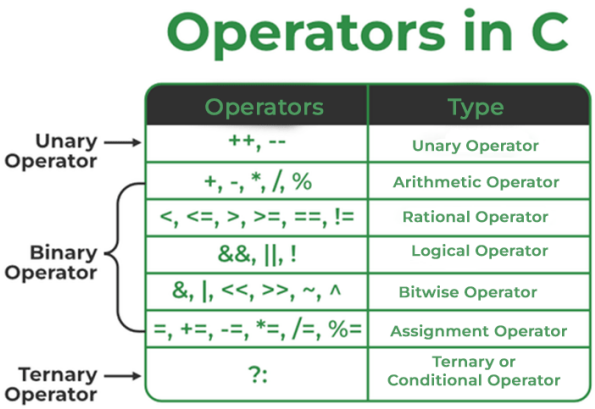
```cpp
#include <iostream>
using namespace std;

int main() {
    bool isTrue = true;
    bool isFalse = false;

    cout << "isTrue: " << isTrue << endl;
    cout << "isFalse: " << isFalse << endl;

    return 0;
}
```

Output:

```
isTrue: 1
isFalse: 0
```

OPERATORS in C++

OPERATORS: Operators are symbols that perform operations on variables and values.



- **Unary operator:** the operator that works on single operands.
- **Binary Operator: the** operator that works on two operands.
- **Ternary operator:** the operator that works on three operands.
    - **Operands:** the data, the expression,the value on which operator act or work.

1. Arithmetic Operators

These operators are used to perform basic mathematical operations like addition, subtraction, multiplication, etc.

- **+ (Addition)**
- **- (Subtraction)**
- **(Multiplication)**
- **/ (Division)**
- **% (Modulus - returns the remainder of a division)**

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

int main() {
    int a = 10, b = 5;

    cout << "Addition: " << a + b << endl;  // 10 + 5 = 15
    cout << "Subtraction: " << a - b << endl;  // 10 - 5 = 5
    cout << "Multiplication: " << a * b << endl;  // 10 * 5 = 50
    cout << "Division: " << a / b << endl;  // 10 / 5 = 2
    cout << "Modulus: " << a % b << endl;  // 10 % 5 = 0 (remainder)

    return 0;
}
```

Output:

```
Addition: 15
Subtraction: 5
Multiplication: 50
Division: 2
Modulus: 0
```

2. **Relational (Comparison) Operators**

These operators are used to compare two values and *return* true or *false.*

- == (Equal to)
- != (Not equal to)
- > (Greater than)
- < (Less than)
- >= (Greater than or equal to)
- <= (Less than or equal to)

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

int main() {
    int x = 10, y = 5;

    cout << (x == y) << endl;  // false (0)
    cout << (x != y) << endl;  // true (1)
    cout << (x > y) << endl;   // true (1)
    cout << (x < y) << endl;   // false (0)
    cout << (x >= y) << endl;  // true (1)
    cout << (x <= y) << endl;  // false (0)

    return 0;
}
```

Output:

```
0
1
1
0
1
0
```

3. Logical Operators

These operators are used to perform logical operations between two or more conditions.

- **&& (Logical AND)** - Returns true if both conditions are true.
- **|| (Logical OR)** - Returns true if at least one condition is true.
- **! (Logical NOT)** - Reverses the truth value of a condition.

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

int main() {
    bool a = true, b = false;

    cout << (a && b) << endl;  // false (0)
    cout << (a || b) << endl;  // true (1)
    cout << !a << endl;        // false (0)

    return 0;
}
```

Output:

```
0
1
0
```

4. Assignment Operators

These operators are used to assign values to variables.

- **= (Simple assignment)**
- **+= (Add and assign)**
- **-= (Subtract and assign)**
- ***= (Multiply and assign)**
- **/= (Divide and assign)**
- **%= (Modulus and assign)**

Example:

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

int main() {
    int a = 10, b = 5;

    a += b;  // a = a + b
    cout << "a += b: " << a << endl;  // 15

    a -= b;  // a = a - b
    cout << "a -= b: " << a << endl;  // 10

    a *= b;  // a = a * b
    cout << "a *= b: " << a << endl;  // 50

    a /= b;  // a = a / b
    cout << "a /= b: " << a << endl;  // 10

    a %= b;  // a = a % b
    cout << "a %= b: " << a << endl;  // 0

    return 0;
```

Output:

```
a += b: 15
a -= b: 10
a *= b: 50
a /= b: 10
a %= b: 0
```

5. **Increment and Decrement Operators**

These operators are used to increase or decrease a value by 1.

- **++ (Increment): Increases the value by 1.**
- **-- (Decrement): Decrease the value by 1.**

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

int main() {
    int a = 10;

    cout << "a++: " << a++ << endl;  // Output 10, but a becomes 11 after this statement
    cout << "a: " << a << endl;     // a is now 11

    cout << "--a: " << --a << endl;  // Output 10, a is now 10

    return 0;
}
```

Output:

```
a++: 10
a: 11
--a: 10
```

6. **Bitwise Operators**

These operators perform operations on the individual bits of data.

- **& (Bitwise AND):** Compares each bit of two numbers, returns 1 if both bits are 1.
- **| (Bitwise OR):** Compares each bit of two numbers, returns 1 if at least one bit is 1.
- **^ (Bitwise XOR):** Compares each bit of two numbers, returns 1 if the bits are different.
- **~ (Bitwise NOT):** Reverses all bits (1s become 0s and 0s become 1s).
- **<< (Left Shift):** Shifts bits to the left by the specified number of positions.
- **>> (Right Shift):** Shifts bits to the right by the specified number of positions.

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

int main() {
    int a = 5, b = 3;  // Binary: 0101, 0011

    cout << "a & b: " << (a & b) << endl;  // 1 (0001)
    cout << "a | b: " << (a | b) << endl;  // 7 (0111)
    cout << "a ^ b: " << (a ^ b) << endl;  // 6 (0110)
    cout << "~a: " << (~a) << endl;        // -6 (inverse of 0101)
    cout << "a << 1: " << (a << 1) << endl;  // 10 (1010)
    cout << "a >> 1: " << (a >> 1) << endl;  // 2 (0010)

    return 0;
}
```

Output:

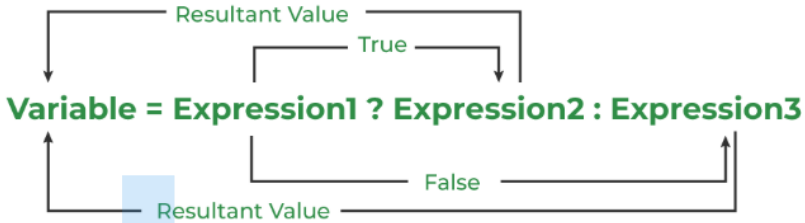
```
a & b: 1
a | b: 7
a ^ b: 6
~a: -6
a << 1: 10
a >> 1: 2
```

7. **Conditional (Ternary) Operator**

This operator is used for making simple decisions.

- ? : (Ternary Operator): A short way of writing if-else.

Syntax:




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

int main() {
    int a = 10, b = 5;

    int maxVal = (a > b) ? a : b;  // If a is greater than b, assign a, else assign b
    cout << "Max Value: " << maxVal << endl;  // 10

    return 0;
}
```

Output:

```
Max Value: 10
```

8. Sizeof Operator

This operator is used to determine the size (in bytes) of a data type or variable.

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

int main() {
    int a = 10;
    double b = 3.14;

    cout << "Size of a: " << sizeof(a) << " bytes" << endl;  // 4 bytes
    cout << "Size of b: " << sizeof(b) << " bytes" << endl;  // 8 bytes

    return 0;
}
```

Output:

```
Size of a: 4 bytes
Size of b: 8 bytes
```

ASCII TABLE

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	|
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	