

if ($a > b$) & ($a > c$), print a
else if ($b > a$) & ($b > c$), print b
else print c

28/01/2023

Programming language

Language in which we are giving instructions to our computer. Also note that every programming language will have its own compiler / interpreter.
Ex - C++, C, Java, Python are programming language.

Compilation process

We will be writing the code in the high level language but computer won't be able to understand it & we need a translator & here compiler will act as a translator. Compiler will convert the high level language to machine understandable language. Machine language means binary language i.e. Os & Is.

Compiler A compiler takes a program as whole and generates intermediate machine codes. This translates the entire source code in a single run.

Interpreter The interpreter will translate the program line by line & also it never generates any intermediate machine codes.

IDE The full form is Integrated Development Environment. It is a tool on which we can write the code. It provides the environment, also provides various features & hence it increases our productivity.

Ex → Codeblocks, VS code, Sublime, Xcode

Writing our first program

→ Scope of int main()

1) `int main() { }`

This is similar to start component we have done in the flowchart. Compiler will always search for this & from here the execution of the program will start.

2) `cout << "Namaste Bharat";`

This is used to print on the console/screen but it gives an error that is the use of undeclared identifier. This means that compiler does not know what is cout. Before we use cout, we need to import its code so that we can use this functionality.

3) `#include <iostream>` cout

This has the code of implementation of cout but again it gives error that is, do you mean std::cout?

We use `std::cout << "Hello";`

↳ This is a namespace

Namespace

This is basically some area and implementation of cout will be different in different namespaces. But we have to use the standard implementation & hence we wrote `std::cout`

↳ standard namespace

Namespace is a region where the scope of identifiers have been defined. For example → scope of cout is defined in std namespace

But this is a cumbersome task of writing std::cout again & again & hence we can define / declare which namespace to use.

using namespace std ; → Now we can use functionality of print simply by cout <<

cout << "Hi" ; → Terminator
↓

For printing → Part of syntax & these symbols are used to print on standard output display.

"Hi" → Hi is a text (string) written inside double inverted commas

Note → cout << 2 ; } These will be printing
cout << "2" ; } 2 on the screen
cout << '2' ;

Also note that << is known as insertion operator.

Single inverted commas means character.

Double inverted commas means string.

Next line

To go to the next line, we use keyword endl.

```
cout << "Hi"; } HiHello
```

```
cout << "Hello"; } Hello
```

```
cout << "Hi" << endl; } Hi
```

```
Cout << "Hello"; } Hello
```

Note → There is one more way to add new line i.e new line character \n.

```
cout << "Hi \n"; } Hi
```

```
cout << "Hello"; } Hello
```

Logically there is no difference b/w endl & \n.

Taking input from user

For printing we use cout, but for taking input from the user, cin is used.

But for taking input, we have to store it in memory & that memory space is given a name

int a; → Taking integer memory space & give it name a.

↳ To take input & send it to memory space a.

Note → To complete giving input, we need to hit enter key.

Comments in C++

These lines are ignored by compiler & hence

are not executed.

→ We have to add this to make line comment.

1/ This is a comment

way 2-

anything written inside /* */ is also termed as comment

e.g. /*this is a comment*/

Comments can be used to make notes. Also there is a shortcut to make a line comment is
ctrl + / Used for better understanding of programmer

Variable

Variables can be defined as named memory location so that we can refer it in future.

int a = 7; →

a cout << a ; → 7

Data type

This tells what type of data is stored.

Char, short, int, float, double are the data types.

variable

int a = 7;

data type value assigned to a

There can be various types of data types

(i) Built-in → Already defined. Ex - int, float, void etc.

for decimals

means empty & no data value

Note →

i/p → function

→ o/p

return type

int main () {

means return nothing

void main () {

Bytes

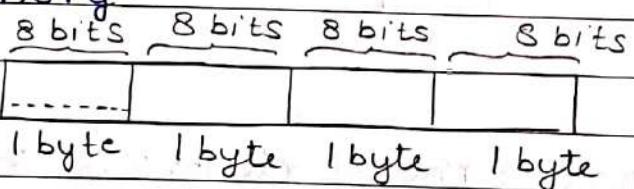
(Also depends on machine)

	Size	Range
char	1	-128 - 127
short	2	-32768 - 32767
int	4	-2147483648 - 2147483647
long	4	-2147483648 - 2147483647
long long	8	-9223372036854775808 - 9223372036854775807
float	4	3.4 e +1 - 38
double	8	1.7 e +1 - 308

32 bit CPU → can be different for
64 bit CPU

int datatype

Takes up 4 byte of memory i.e $4 \times 8 = 32$ bits of memory.



As each bit can have either 0 or 1 value, then there are 2 combination possible for each bit & hence 2^{32} are possible combinations.

char datatype

char ch = 'a';

↪ 1 byte
8 bits

1 byte

2^8 combination are possible as each bit have either 0 or 1 & hence total 8 bits.

$$2 \times 2 \times 2 \times 2 = 2^8 \\ \text{C 8 times}$$

Now question comes that whether character a is stored or not. But a number is stored instead & all the characters are mapped to ASCII value i.e a numeric value & this numeric value is converted into 0s & 1s & then stored.

`char ASCII val.`

`a → 97 → converted to 0s & 1s and then stored in memory`

boolean datatype

Can have only 2 values i.e either true or false.

→ data type

`bool flag = true;`

`cout << flag;` → gives o/p. 1.

`bool flag = false;`

`cout << flag;` → gives o/p 0.

Note → true → 1 } similar
false → 0 }

** The space taken by boolean is 1 byte but it is sufficient to represent it in 1 bit but still 1 byte is taken because smallest / minimum memory that can be used is 1 byte. There is memory wastage also.

Smallest addressable unit is 1 byte.

float & double data type

These are used for representing decimal numbers.

`float f = 1.2;`
`double f = 1.2;`

But which one to use? Double is more precise than float & also float takes up less memory space. The one taking more space can represent much bigger number.
 → (0 to 255)

Ques → As we know that 256 characters can only be stored in char but what if we do `char ch = 256;` ?

This is basically the overflow condition & error comes i.e overflow in conversion from int to char changes value from "256" to "\000". Also we can say last 8 bits will be copied. Depends on compiler to compiler

(ii) Derived data types

Primitive data type are used to make these. Ex → Arrays, pointer etc.

(iii) User defined data type

Make our own data type. Ex → structures, union, classes & enumerations.

Variable naming conventions

- 1) Can contain letters, digit & underscores
- 2) Names must begin with letter or underscore
- 3) Names are case sensitive
- 4) Names can not have spaces & no special character other than underscore can be used.

sizeof()

This returns the size of data type in bytes

`sizeof (int); → 4 } in bytes`
`sizeof (char); → 1 }`

Storage of data

(i) Positive numbers

`int a = 5;`

0	0	0	1	0	1
---	---	---	-------	---	---	---

32 bits

`char ch = 'a';` → Just convert the ASCII value to binary & then it will be similar to that of integer.

0	1	1	0	0	0	1
---	---	---	---	---	---	---

8 bits

It is important to note that first bit from left will be 0 if +ve number is there. It will be 1 if -ve number is there.

(ii) Negative number

Will be stored in form of 2's complement.

Way to find 2's complement

↳ Is complement + 1

Flip 0 to 1 & 1 to 0

7 → 00000111 } Just for understanding 8
 11111000 → Is complement bits.

11111000

1

11111001 → 2's complement

Note → In binary $1+1=10$.

`int a = -5`

Step 1 Ignore negative sign.

5

Step 2 Find binary equivalent of 5

000....0101

Step 3 Find 2s complement

111...1010

-ve no. $\leftarrow \text{111...1011} \rightarrow$ 2s complement

This way -ve number is stored in memory.

How to read from memory?

Just find 2s complement

000...0100

1

000...0101 \rightarrow 5

But we have already seen that first bit in memory representation was 1 & hence it is a negative number

Ans \rightarrow -5

Interesting problem

22	31	43	7
----	----	----	---

\rightarrow each of 1 byte

Q.) How we get to know whether we have to read all the 4 bytes or just 1 byte.

Sol.) This is known by the data type used.

Signed vs unsigned data type

In signed we can store all the numbers whether +ve, -ve or 0.

In unsigned we can not store the negative values.

For example → unsigned int will have range of 0 to $2^{32} - 1$

Note By default, data type will be signed.

Range of signed int will be -2^{31} to $2^{31} - 1$.

Unsigned int

2^{32}

0 to $2^{32} - 1$

Divide into 2 half

2^{31}

2^{31}

That's why taking 2^{31} in the range.

Generic for n bits

Signed → -2^{n-1} to $2^{n-1} - 1$

Unsigned → 0 to $2^n - 1$

Typecasting

Converting one type of data type to another data type is known as typecasting.

char ch = 97 ; → Integer to character

cout << ch ; → gives a as output

int num = 'b' ; → Character to integer

cout << num ; → gives 98 as output

Implicit Type conversion

This is automatic typecasting. The above examples were of implicit type conversion.

Explicit Type conversion

This is done manually. This uses () operator for typecasting

```
float a = (float) 2 ;
```

↳ here data type will be mentioned.

Operators in C++

1) Arithmetic operators

+, -, *, /, % are arithmetic operators.

a = 5

b = 3

cout << (a/b) << endl; → This gives 0.666666 as output.

Note →

int = int, float = float, double = double

int int

int



Bigger data type will always dominate.

int = float, float = float

float

float

2) Relational operators

>, <, >=, <=, !=, ==

↓

equal to

not equal to

This will give the boolean output.

3) Assignment operator

= int a = 3 ; → 3 has been assigned to a

4) Logical operator

Used when there are multiple conditions.

&& → Logical AND (AND gate)

|| → Logical OR (OR gate)

! → Logical NOT (NOT gate)

(i) AND

$$0 \& 0 \rightarrow 0$$

$$0 \& 1 \rightarrow 0$$

$$1 \& 0 \rightarrow 0$$

$$1 \& 1 \rightarrow 1$$

(ii) OR

$$0 | 0 = 0$$

$$1 | 0 = 1$$

$$0 | 1 = 1$$

$$1 | 1 = 1$$

(iii) NOT

$$! 0 = 1$$

$$! 1 = 0$$

Note →

cond1 && cond2 && cond3

↳ If it becomes false, then no need to check other conditions.

*** **cond1 || cond2 || cond3**

↳ if cond1 becomes true(1), then no need to check other

This is termed as **SHORT CIRCUITING**