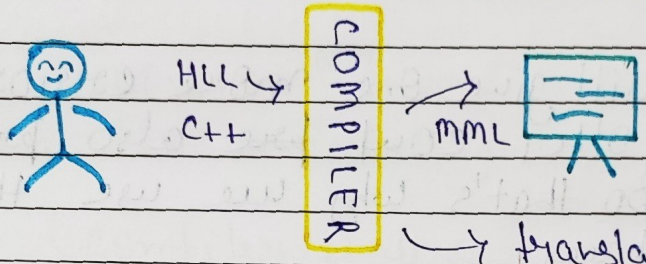
Date: / /

Programming Languages

A way to communicate with computer is called programming languages. It has fixed set of rules or standard. Every language has its own ~~set~~ compiler / interpreter.

Compilation Process

↳ works as a translator



↳ translator

Where to code?

code editor or IDE

↳ VS code, Sublime, code blocks

Let's write our first code

int main() → Starting point of our code

→ It is function

↳ It is a block of code. In which we can provide input and it gives some output. (Reuseable block of code)

return type ← int main() → Part of syntax

{ ~~~ ↳ function name

~~~

Syntax ↑ } → scope of function

cout << "I Love Babar"; → end of line

String

↳ it display's ~~s~~ on standard output (prints)

↓

on Standard display
~~~

papergrid

Date: / /

cout
↓
define in iostream header file. if we want to use cout in our code we need to import/include it in our code.

→ Part of syntax
#include <iostream>
↳ syntax

Now, It will give one more error because there are other cout pre also present in compiler. So that's why we use the concept of namespaces

↳ Portion of code it has additional information of different cout's

using namespace std;
↳ Here we are using standard namespace

Preprocessor directive
↑
→ importing this file

```
#include <iostream>
using namespace std;
int main(){
    cout << "Aman Grabbar";
}
```

- #include → Preprocessor directive; importing this file
- additional information → using namespace std;
- using namespace std; → we use standard namespace
- int main(){ → scope; entry point
- cout → identifier used to print on console.
- << → syntax
- "Aman Grabbar" → String
- ; → end of line

Date: / /

cout << endl; → It prints a new line
cout << '\n'; → It can also
↳ a new line character
Ex- cout << "Aman Grabbar" << endl;
↓
for next line

// → use for commenting.
if we do like this

// cout << "Aman" << endl; → then it is the non executable line (compiler ignore that line) the whole code.
cout << "Grabbar";

What is return 0 in main function?
In main function return 0 means that the program executed successfully. It is fully optional the compiler automatically adds return 0; if you don't add a return value.

```
1  #include<iostream>
2  using namespace std;
3  int main()
4  {
5      cout << "Aman";
6      cout << endl;
7
8      return 0;
9  }
```

→ we can create our namespaces (custom)

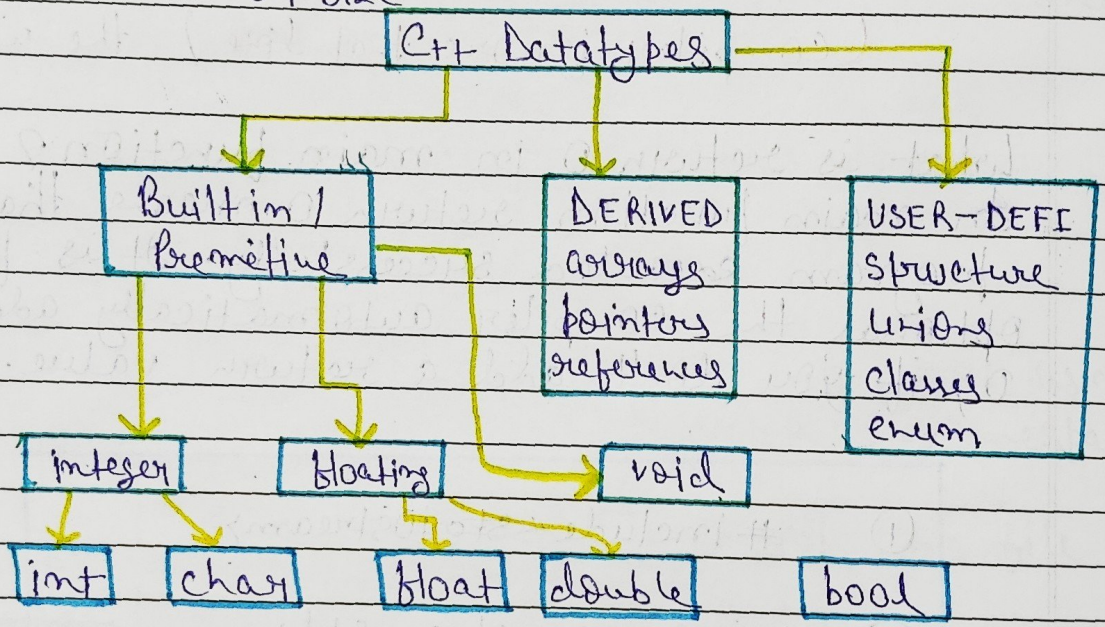
Big room → namespace → cout
Iostream → vault → money

Date: / /

What happens
cout << "Aman"; cout << "Grabbar";
→ It will be continue with out any error.
Output - AmanGrabbar

Input / Output in C++
↳ cout
↳ cin >> variable name to store;
cin >> a; → input from user.

Datatypes & Variables → named memory location
↳ types of data or size



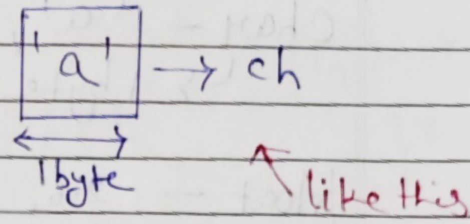
int marks = 90; → value
↳ name assign to data (variable)
↳ assignment Operator
↓
type of data

1 byte - 8 bit
1 bit → 0/1

int - 4 byte or 32 bits

int a = 5 → address 108.
int → data type
5 → 4byte
→ a name of variable

char ch = 'a';
character type ←



↳ It will make 1 byte block in memory according to datatype

Void — Empty / Null

Difference between int main & void main

```
int main () {
    // code
    // code
    return 0;
}
```

→ It is a int type function that's why its returning.

```
void main () {
    // code
    // code
}
```

→ this main function will return nothing because of void.

Derived Data type — This type of data is made by use of built-in / pripmitive datatypes.

User Defined — This data type is created by the programmer.

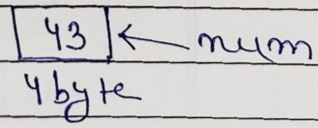
Variable — name given to a memory location.

Datatypes —

Integer — -3, -21, -99, 0, 1, 2, 100
↳ int
↳ int num = 43;

int num; ↳ declaration

int num = 50;
↳ initialization



Date: / /

Char – 'a', 'k', '$', '!'
↳ 1 byte

float – 2.6, 1.7, 99.871
↳ 4 byte

long – 8 bytes (but machine dependent)
long — in 32 bit → 4 bytes
long — in 64 bit → 8 bytes

Garbage Value

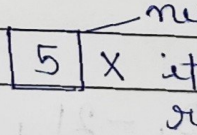
```
int num;
cout << num << endl;
```
↳ it prints garbage value.

Variable Naming Conventions

→ Use meaning full names.
→ Begin with a letter & underscore (_).
→ No keywords.
→ C++ is case sensitive Ex- student & Student different variables.
→ camelCase use → Ex- totalAmount
→ etc.

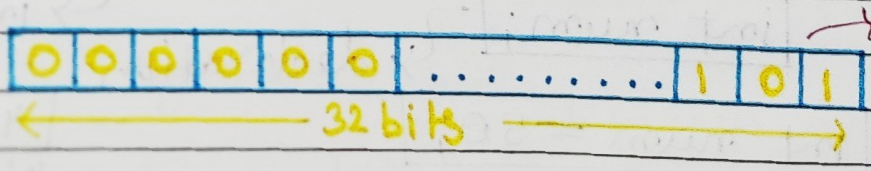
How data is Stored? +Ve or -Ve integers?

int num = 5;

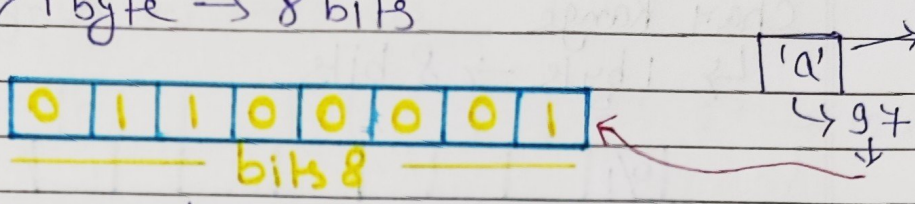


X it stores in binary representation.

int → 4 bytes → 32 bits



char → 1 byte → 8 bits



ASCII — every character mapped with different numerical ASCII value.

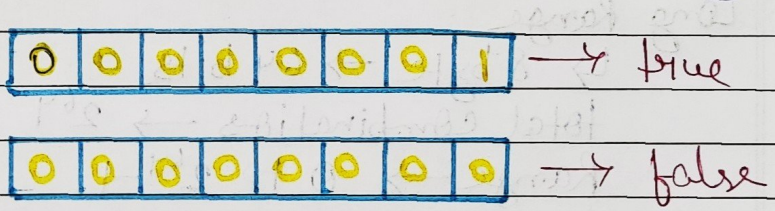
char ch = 'a'; → 97 ASCII value
↳ Binary stores
↳ 1100001

Boolean — 1 byte → 8 bits

bool flag = true;

true — 1
false — 0

it use only 1 bit, so why it waste 7 bits because smallest addressable memory is 1 byte.



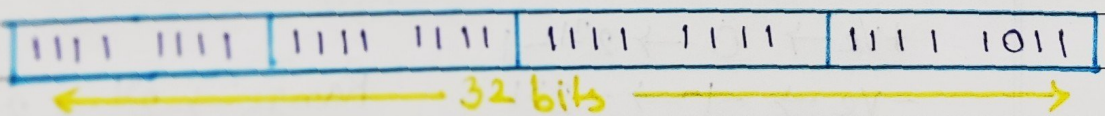
How -ve numbers stores in memory?

→ ignore -ve sign
→ find out binary equivalent
→ take 2's complement

int a = -5

1 → 5
2 → 00000000 00000000 00000000 00000101
3 → 2's complement

1's → 11111111 11111111 11111111 11111010
2's → + 1
11111111 11111111 11111111 11111011



it is the binary representation of -5

Date: / /

Char Range

↳ 1 byte → 8 bits

2 choices total combinations → 2^8

$= 256$

Range − 0 → 255

$0 \rightarrow 2^8 - 1$ $(0 \rightarrow 2^n - 1)$

Generic Formula

int Range

↳ 4 byte → 32 bits → n bit

total combinations → $2^{32} \rightarrow 2^n$

Range → $0 \rightarrow 2^{32} - 1 \rightarrow 2^n - 1$

Long Range

↳ 8 byte → 64 bits

total combinations → 2^{64}

Range → 0 to $2^{64} - 1$

Xyz Range

↳ 71 bytes → 568 bits

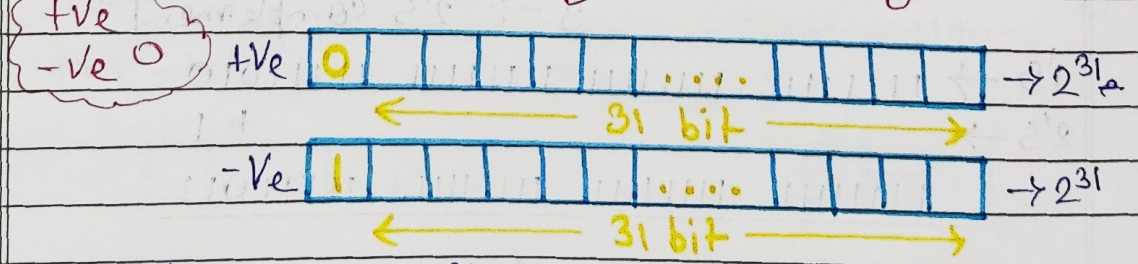
total combinations → 2^{568}

Range − 0 to $2^{568} - 1$

Signed vs Unsigned data

↳ +Ve

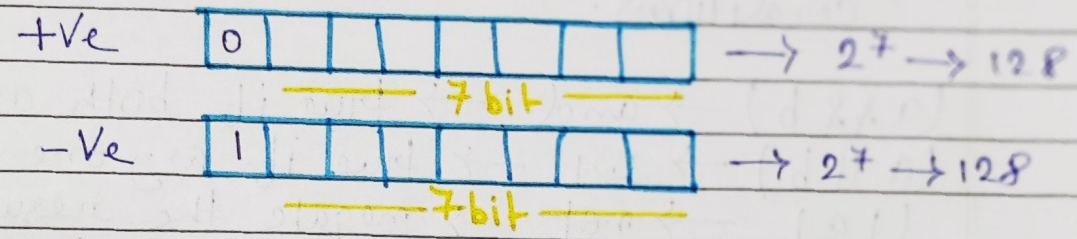
int a = 5; → by default signed



+Ve → 0 → $2^{31} - 1$

−Ve → −1 → -2^{31} Range → $-2^{31} \rightarrow 2^{31} - 1$

Char Signed vs Unsigned



+Ve → 0 → 128 → 0 → 127
-Ve → -0 | → -128 →
-128 → 127
Range

Operators

Arithmetic — +, -, *, /, %.
int op int → int
float op int → float → Higher data type will
int op float → float dominate always.
Same for double

3 → int
3.0 → float / double → by default

```
cout << sizeof(3.0);
```
↓
(8)

Relational — <, >, <=, >=, !=, ==
Output - 0 or 1
False ← → True

Assignment Operator — =
variable = value

+=, -=, *=, /=, %=
↳ This are compound assignment Operators.

Date: / /

Logical Operators - When you have multiple conditions.

(a && b) → and → true if both are true
(a || b) → or → true if any one is true
(!a) → not → negate the result.

Output - 0 or 1

false ↙ ↓ true

Conversion of !

0 — 1
1 — 0
any Non-Zero value — 0