* Skeleton of c++ program:-

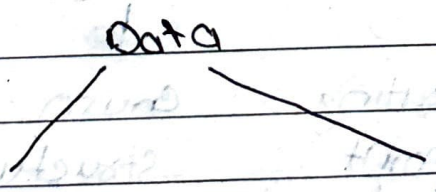
```
#include <iostream>
using namespace std;
int main() {
 std::cout << "Helo world";
}   return 0;
}
```

- main() - Starting point of the program is main function.
- int - int here is return type of the function.
- return 0;:- means the program ended successfully.

- Any non-zero value (eg. return 1;) indicates that the program encountered an error or abnormal termination.
- cout - cout is a Object used to print anything on the monitor.
- << - insertion operator.
- #include <iostream> - Header file.
- iostream - library
- :: - scope resolution operator
- std :- all the built in things available in the library are grouped under one name i.e std

* Data and data types :-

A program is a set of two ingredients
i) data
ii) operations performed on that data



Numeric	character/Alphabetic
12	
25	128-A
15.35	

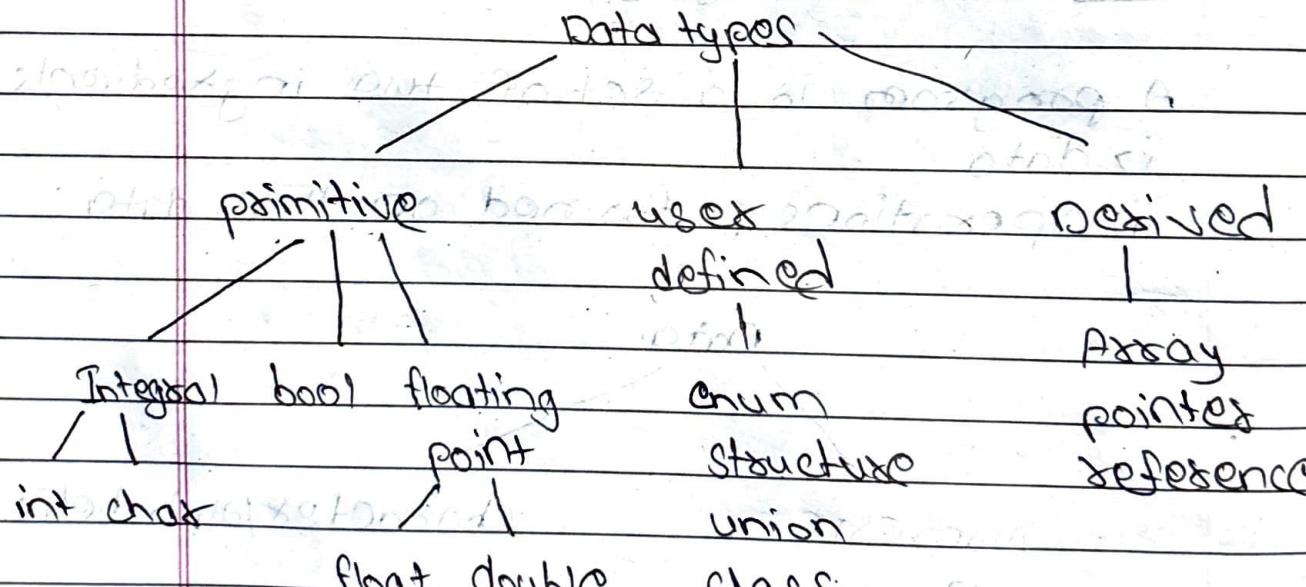
- All electronic devices where numbers are ~~use~~ used, treat numbers with

point and numbers without decimal point differently because there are extra efforts needed for representing decimal point number.

i) integer (without decimal point)
ii) float (with decimal point)

- A alphabet is a character whereas a set of alphabets/characters forming on single entity is string.

* Data types and variables

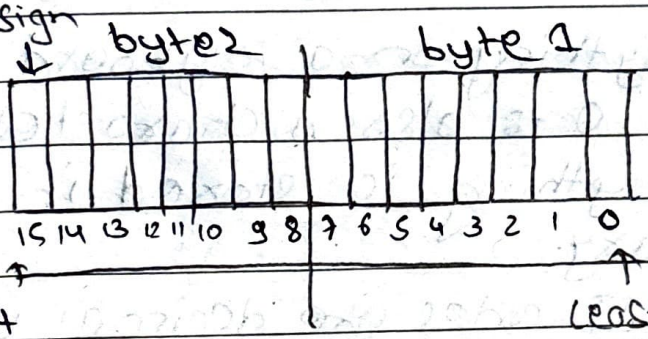


- Integral means not having a decimal point.

(in bytes)

Data type	Size	Range
int	2 or 4	-32768 to 32767
float	4	-3.4×10^{-38} to 3.4×10^{38}
double	8	-1.7×10^{-308} to 1.7×10^{308}
char	1	-128 to 127
bool	undefined	true / false

* Int of 2 byte :-



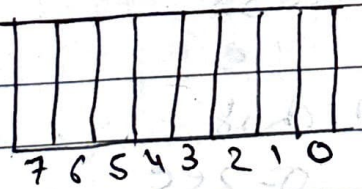
$2^{15} = 32768$

0 - 32767

-32768 0 32767

- If most significant bit is 0, the no. is positive.
- If MSB is 1, the number is negative.
- In negative it is -32768 because 0 doesn't exist.
- In positive it starts from 0 - 32767.

* Char of 1 byte :-



$2^7 = 128$

0 - 127

-128 0 127

- For characters, ASCII codes are given

A-65	a-97	0-48
B-66	b-98	1-49
⋮	⋮	⋮
Z-90	z-122	9-57

- Everything on a keyboard is a character so 0 is also a character.
- Everything is stored in the form of binary.
- ASCII codes are decimal values which are converted to binary while storing.
- ASCII - American standard code for information interchange.

* modifiers:-

i) unsigned
ii) long

- unsigned int - takes only positive values
  0-65535

- unsigned char
  0-255
- long int - 4 bytes
  8 bytes

- long double - 16 bytes

* Difference between float and double:-

1) Size and precision:-

- float : 32 bits (4 bytes) with 7 decimal digits of precision.
- double : 64 bits (8 bytes) with 15-16 decimal digits of precision.

2) memory usage:-

- float uses less memory, but its less precise.
- double uses more memory, and hence it is more precise.

- Why use the 'f' suffix (eg. 3.14f)?

In C++, floating point literals like 3.14 are treated as double by default.

To explicitly specify that a literal is a float, you use the f suffix:

```
float a=3.14f;
```

- without the f suffix, like:

```
float a=3.14;
```

The literal 3.14 is treated as a double and the compiler will have to convert it to float, which is less efficient.

* Arithmetic operators and expressions:-

Arithmetic - +, -, *, /, %

Relational - <, <=, >, >=, ==

Logical - &&, ||, !

Bitwise - &, |, ~, ^

Increment/Decrement - ++, --

Assignment - =

- If we want quotient then we '/' (divide) sign.

```
int a=10/5;
```

5) 10 (2 → quotient
-10
0

- If we want remainder, then use '%' sign.

int a = 13%5

$$5\overline{)13}\ \ 2,\quad -10,\quad \boxed{3}$$ → remainder.

- Even if we use float or double while variable declaration, then too we will not get floating point value.

```
float a = 13/5;
  output → 2
    or
double a = 13/5;
  output → 2
```

- To get floating point value, we should typecast it.

```
float a = (float) 13/5;
  output → 2.6
    or
double a = (double) 13/5;
  output → 2.6
```

- In java, it will give ans with exact precision points.

#IMP - If both values in a operation are int, then compiler will assume ans will also in int, so we dont get ans in float for float a = 13/2 (int/int)

Imp

- If any one value is float, then the output will also be float.

float a = 13/2.0 = 6.5

Date
Page

- If both the values are float then we get ans as float.

float c = ~~(float)~~ 13.5/5.5;
output → 2.454545

- '%' operation can only be done in int and char.

* operator precedence and expressions:-

- precedence - priority or weightage.

operator	Assumed precedence
( )	3
*, /, %	2
+, -	1

- $x = \underset{3}{a} + \underset{1}{b} * \underset{4}{c} - \underset{2}{d} / e$

- Here the precedence sequence will go from left to right
- Associativity is left to right.

- If we want to increase precedence of some, then we use brackets.

$x = \underset{1}{(a+b)} * \underset{4}{c} - \underset{3}{d} / e$

Date
Page

* Some common formulas and how to write them in a program :-

i) Area of triangle $a = \frac{1}{2} bh$ — a = b * h / 2

ii) ~~Area~~ perimeter of rectangle $P = 2(l+b)$

— P = 2 * (l+b)

iii) Sum of n terms

$S = \frac{n(n+1)}{2}$ — S = n * (n+1) / 2 ;

iv) n^{th} term of A.P series $t = a+(n-1)d$

t = a+(n-1) * d

v) Quadratic equation $x = \frac{-b + \sqrt{b^2-4ac}}{2a}$

x = -b + b*b - 4 * a * c

By using header file #include <cmath>
or #include <math.h>

x = (-b + sqrt(b*b - 4*a*c))/(2*a)

vi) speed = $\frac{v^2-u^2}{2a}$ - speed=(v*v)-(u*u)/(2*a);

OR

speed=(pow(v,2)-pow(u,2))/(2*a);,

* find sum of first n natural numbers:-

Sum = n*(n+1)/2

• natural numbers start from 1.

Sum = 1+2+3+4+5 = 15
= 5*(5+1)/2
= 5*(6)/2
= 15

* Compound assignment:-

+=
-=
*=
/=
%=

&=
|=
<<=
>>=

Date
Page

* Increment and decrement operators:-

- pre increment
  ++x;

- post increment
  x++;

- pre decrement
  --x;

- post decrement
  x--;

• In pre-increment

```
int x=5, y;

y = ++x;      x = 6
(2) (1)       y = 6
```

~~Here, the precedence sequence is~~
- Here, first the value of x will be incremented and then it will be stored in y.
- "++" is done first and "=" is done after it i.e assignment is done after incrementing in pre-increment.

• In post increment

```
int x = 5, y;

y = x++;   y = 5
①  ②       x = 6
```

• Here "=" will be done first and "++" is done after it i.e assignment is done first and increment is done after it.

- ```
  int x = 5, y = 10, z;

  z = x++ * y;
  ②   ③  ①
  z = 5 * 10
  z = 50 , x = 6
  ```

- ```
  int x = 5, y = 10, z;

  z = ++x * y;
  ③  ①   ②
  z = 6 * 10
  z = 60 , x = 6.
  ```

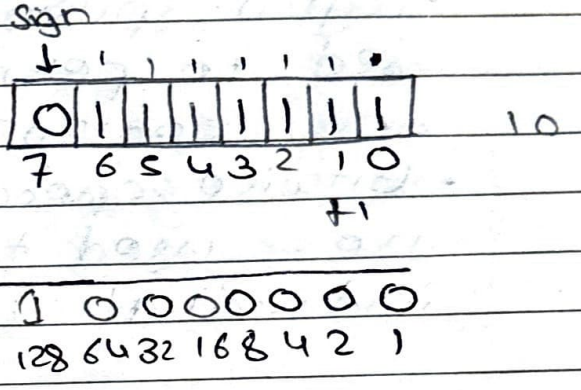
* overflow :-

Range of char

-128 to 127

2	127	
2	63	-1
2	31	-1
2	15	-1
2	7	-1
2	3	-1
2	1	-1
	0	-1

- Sign bit = 0 : number is +ve
  sign bit = 1 : number is -ve



char x = 127

- Now, if we do ++x; then this will overflow as range for a char is only from -128 to 127.
- when the value is more than the capacity, so it will take values again from the beginning.
- So adding or incrementing x will return it to the beginning of range i.e -128
- To print this, we should typecast it

~~char x = +~~

cout << int(x);

Date
Page

* Bitwise operators:-

• Logical vs bitwise operators:-

i> Purpose:-

- Logical operators: this operators are used to perform logical operations on boolean values (true/false).
- Bitwise operators: these operators are used to perform operations on individual bits of integers.

ii> operands:-

- Logical operators: they work with boolean values.
- Bitwise operators: they work with integers (binary representation numbers).

• Bitwise operators:-

& and
| or
^ X-OR
~ NOT
<< left shift
>> right shift

i> & and

bit 1	bit 2	bit 1 & bit 2
0	0	0
0	1	0
1	0	0
1	1	1

ii> ! or

bit 1	bit 2	bit 1 ! bit 2
0	0	0
1	0	1
0	1	1
1	1	1

iii> ^ x-or (exclusive -OR)

bit 1	bit 2	bit 1 ^ bit 2
0	0	0
1	0	1
0	1	1
1	1	0

a> int x = 11 , y = 5 , z

x = 00001011 y = 00000101

z = x & y

$x = 00001011$

$y = 00000101$

00000001

$\therefore z = 1$

for $x=11, y=7$:-

- if we print cout<<(x&y);, the output will be 3 because '&' is bitwise &.
- whereas if we print cout<<(x&&y);, the output will be 1 because '&&' is logical &.
- logical and gives 1 for both values positive.
- logical and gives 0 if any one or both values are zero.
- Bitwise & gives 0 if any one or both values are zero.

b) int x=11, y=7, z;

$x \wedge y =$ 00001011

00000111

00001100

$2^3 2^2 2^1 2^0$

$\therefore 2^3 \times 1 + 2^2 \times 1 + 2^1 \times 0 + 2^0 \times 0$

$= 8 + 4$

$= 12$ [cout<<(x^y)]

c) char x=5, y;

x → 00000101

y=~x → 11111010

-ve

-6

- Negative numbers in C++ are represented in 2s complement form
- If we want a positive value then again you have to take 2's complement to get a +ve value.

2's complement

```
 11111010
 00000101
        +1
 --------
 00000110
 --------
    6
```

d) int x=5, y;

x = 00000101

y = x<<1 00001010

$2^3 2^2 2^1 2^0$

= 10

- ~~If~~ left shifts, ~~if we perform~~ left shift ~~with any value, the original value is raise to any value.~~

- if $x = 5$
  $y = x << 1$
  $\therefore x * 2^1 = \underline{\underline{10}}$

- if $x = 5$
  $y = x << 2$
  $\therefore x * 2^2 = \underline{\underline{20}}$

Imp
- $x << i$ [left shift]
  $x * 2^i$

Imp
- $x >> i$ [Right shift]
  $\frac{x}{2^i}$

- int x=5, y;
  y = x >> 1
  $\frac{x}{2^1} = \frac{5}{2} = 2$ [int].

- Sign bit is not disturbed in left shift and right shift operations.
- i.e negative numbers remains -ve and positive remains +ve.