Introduction

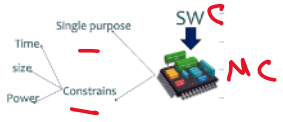
Sunday, August 6, 2023 4:36 PM

→ why Emb?

① Automotive ④ Health Care
② Automation
③ IoT

→ What Emb?

C → SW → SoC → NC → Special purpose → Constrains → cost ↓ time size ↓ Power ↓



History

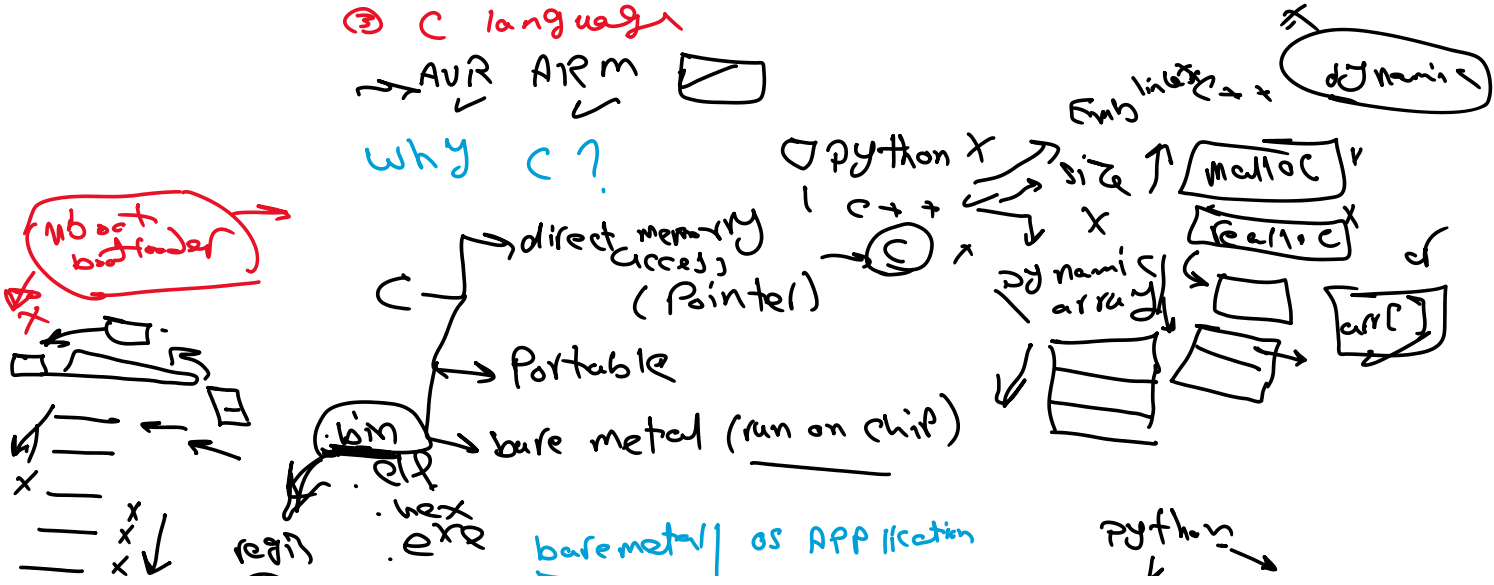
① Machine Code 0 & 1 → 011 01 11 (1+2)
② Assembly Code add(01,11)
↳ Machine Dependant

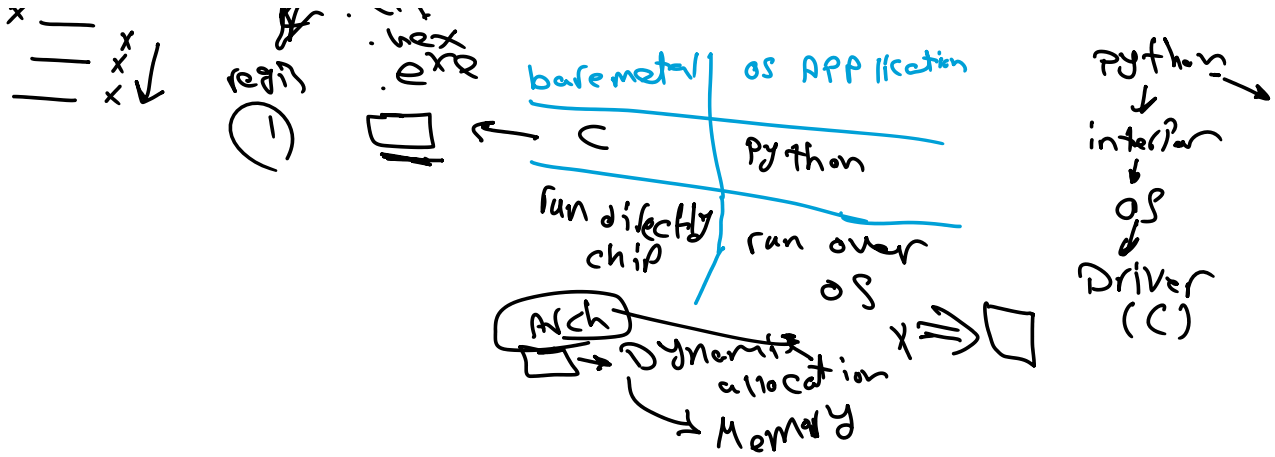
AVR ✓ ldr str ARM ✗

③ C language

→ AVR ARM

Why C?

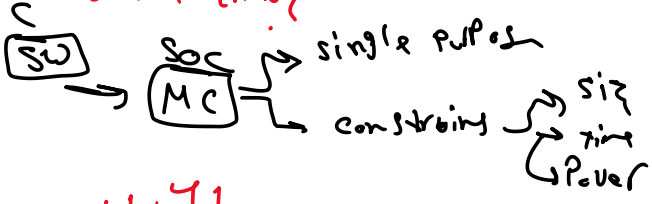




Summary

Why fmb?

What fmb?



History!

① Machine Code

② Assembly Code Machine Dependent

③ C language

Portable

Direct memory accell (Pointer)

bare metal

bare metal → Chip

OS APPlic → run over OS

C easy to read?

↳ NO

Lecture 1

Sunday, August 6, 2023 5:56 PM

→ What C ?

→ function

main fun → entry Point

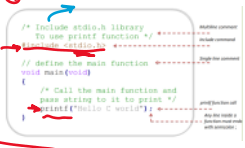
startup call main fun

Assembly C

Assem / C

Hello World

comments



stdio.h → library
→ printf / scanf

Comments

* will not be executed

/* */ → multi line

// → single line

Comments

① documentation
② Clear
③ 1 comment → 1 line Code

syntax:

printf("Hello World");

'A' → char

"A" → string

library fun <stdio.lib>

#include<stdio.h>

```
int main()
{
    printf("Hello");
    return 0;
}
```

printf not rec

1 X error

```
x = printf("Hello");
```

x = 5

Hello 5

\n → new line
\t → new tab
\" → "
\' → '
\\ → \
\r →

lab 1

lab 1

Variables:

syntax:

type name ;

↳ Declaration

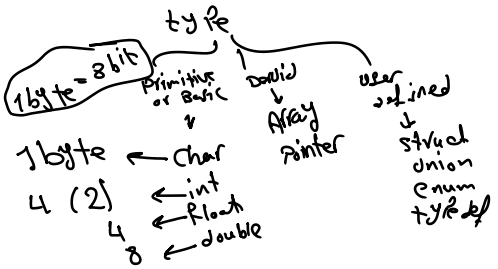
type name = value ;

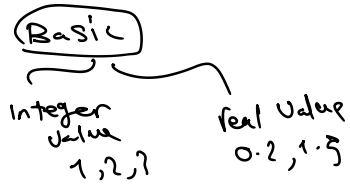
↳ Initialization

4 byte

int x = 10 ; 4 

x





type ⇒ size

syntax:

type name; → Declaration

type name = value; → initialization

name:

Rule):

A to Z, a to z → Ahmed, ahmed

0 to 9 → Not first char

_ underscore → ✓

special ch → X $X

Space → X

keyword → X

↳ int, void, main, float

int _ ; ✓

int 1a; X

int a1; ✓

int Int; ✓

int ___1; ✓

int Int;

int ___1;

printf("%. ", x); → 1 byte (~~character~~)

int ← d
char ← c
float ← f
double ← lf

char x = 48; 48 → ascii → '0' 97 → 'a'

printf("%c", x); →

printf("%d", x); → 48

scanf → library fun <stdio.lib>

↳ get value from user

syntax;

scanf("%d", &x); → Address of

%d
%c
%f
%lf

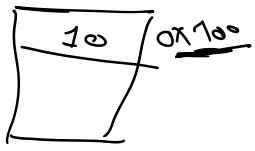
↳ MUST

(segmentation fault error

x = printf("Hello"); / x = scanf("%d", &y); 10

x = 5 ←

x = 1 y = 10



x = scanf("%d %d", &y, &z);

x = 2

int x = 0; → 0

scanf("%d", &x);

↳ Segemation fault

error

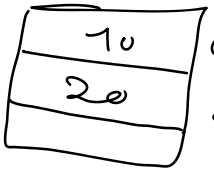
lab 2

int x, y;

z = scanf("%d %d", &x, &y);

10 20

z = 2



x = 10
y = 20

int

scanf

scanf

scanf

buffer

clean

fflush(stdin);

fflush(stdout);

Git bash

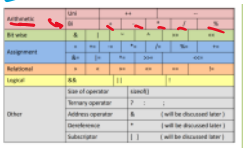
C operator

Address &x

① Uni: op □

② Bi: □ op □ 2+3

③ Tri: □ □ □



x=1 y=2

x+y=3

x-y=-1

x*y=2

x/y = 1/2=0

x%y = 1

int/int=int

int/float=float

sign(x) ÷

+

-

4+2%-8=2 X → 6

4+-2%8=2 ✓

4+-2%-8=6 X 2

4+2.0%8=6.0 X float % — ⇒

float k=0.0;

k=5/2+3/2+1

× /

+ —

sign(y) $y = \boxed{\text{sign}(x)\ z}$

k = 4.0

k = 2 + 1 + 1 = 4

float

4.0

?

5 → in

5.0 → flo

lab 3

++ , -- Increment/decrement

↳ x = 10

x++; → x = x+1 → x = 11

++x; → x = x+1 → x = 11

x++ → Postfix

++x → prefix

x = 10

y = x++; → x = 11 y = 10

y = ++x; → x = 11 y = 11

y++ = x++ Xeror

y = y+1 = x = x+1 X

==

a = b = 3 X

b + 3 = a X

a = 3 + b ✓

X

y++ = x++

x = 10

$y = x++ ; \rightarrow x = 11 \quad y = 10$

$y = ++x ; \rightarrow x = 11 \quad y = 11$

$y = x++$ { $x = x+1$ ✓ ; $y = 10$ ✓ }

$y = x++$ { $y = 10$; $x = x+1$ } ✗

$i = 3 ;$

$i = i++ ;$ → { $i = i+1$; $i = 3$ } $\quad i = 4$

$i = i++ ;$ → $i = 3$

$i = 3$ ✗ 4

$x = 10$

$x = x + 7 ;$

$x = 10$

$x = ++x ;$

$x = x + 1$

$x = 11$

$i = i++$

$i = 4$ ✗ $i = 3$

if(printf(———))

$y = x++$ → $x = x+1$; $y = 10$

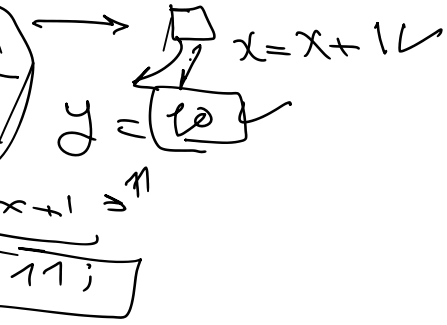
$y = ++x$, → $x = y$

Post → return then increment

Pre → increment then return

$\text{---} \rightarrow x = x - 1$

$y = 5++ ;$ → error ~~$5 = 6$~~ $\quad 5 = 5+1$


x=x+1
y=10
x+1
11;

y = 5 + + 1

y = 5

5 = 5 + 1

++ , -- ⇒ Variable

Binary: 0 & 1

10 → 0000 1010 (8 bit)

128 64 32 16 8 4 2 1

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

octa → 0000 1010

8

10 → 000 001 010

012

x = 012 → Octa

Hex → 0000 1010

16

0x 0A

10 → 0000 1010

8 → 3 octa

000 001 010

8 → 2^3

16 → 2^4

x = 012 ;

↳ Octa

0x0A → (10)10

Hex

(10)10 = (0x0A)16 = (012)8

Hex

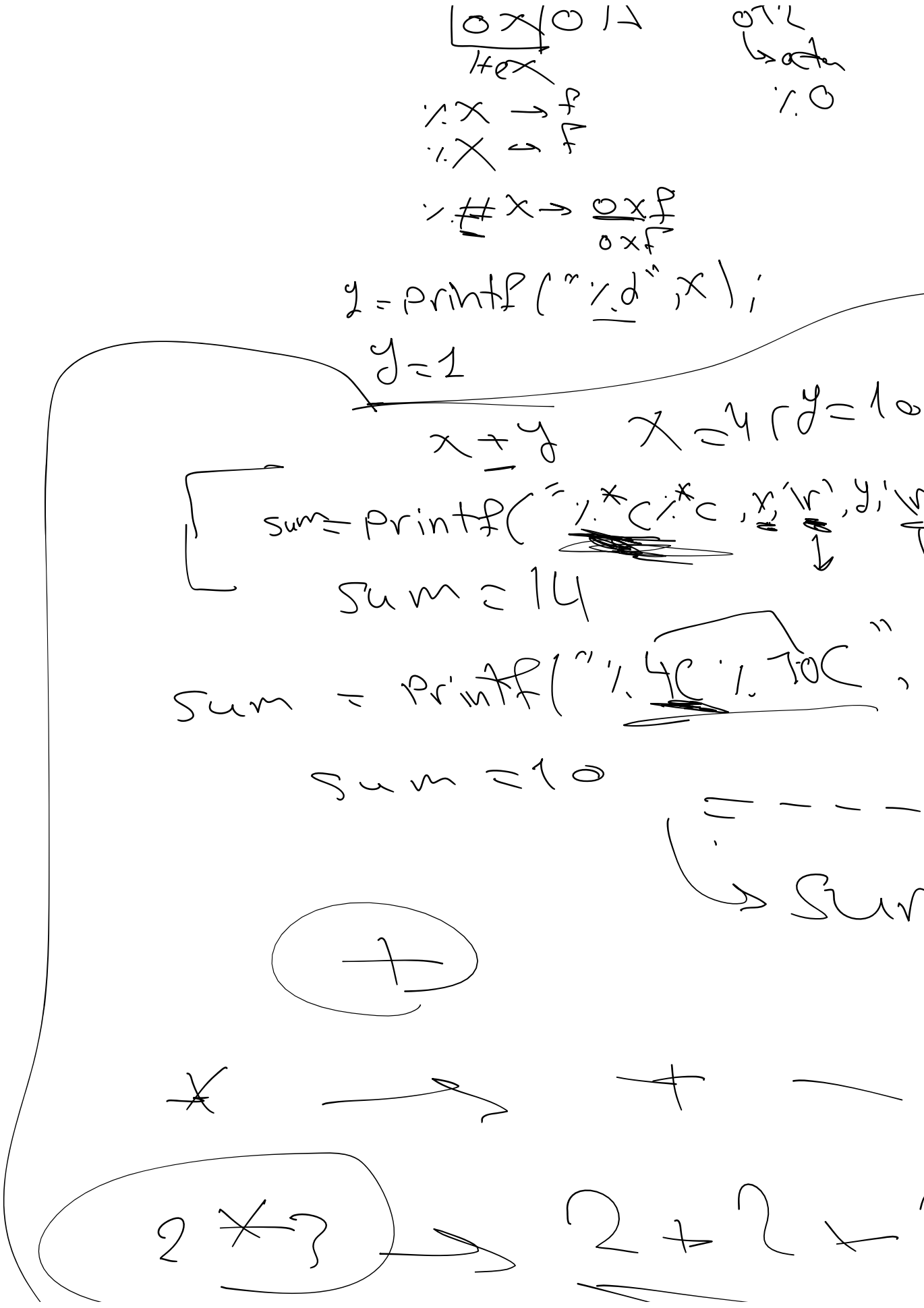
octa

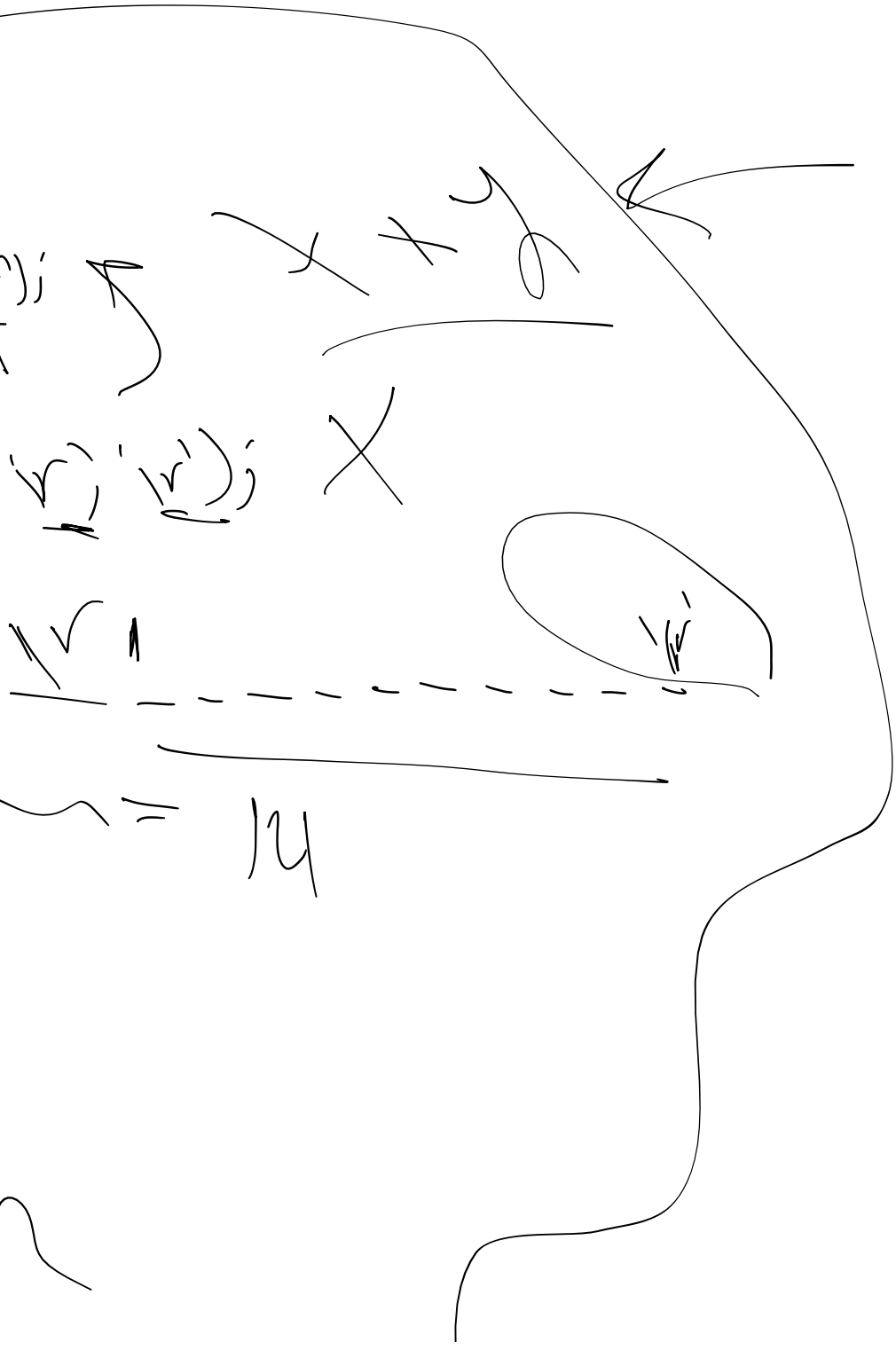
0x0A

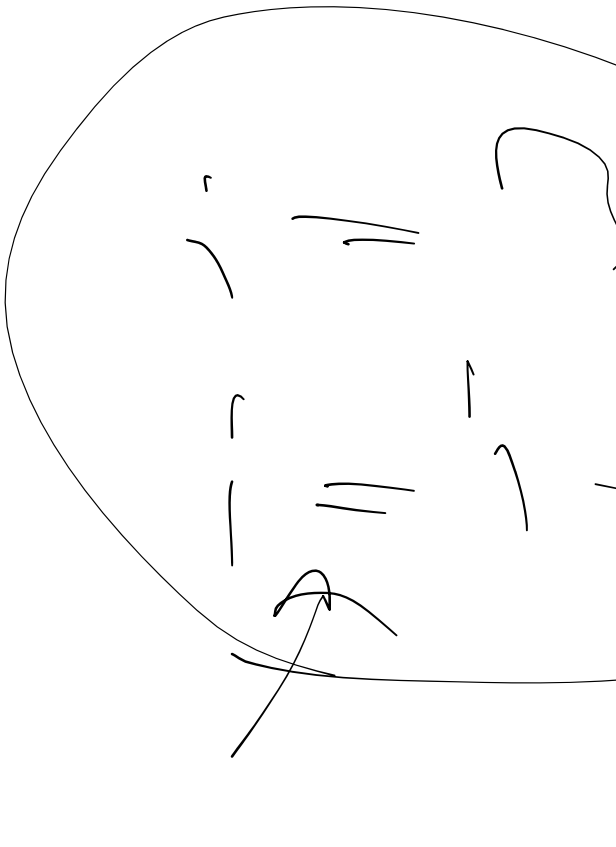
Hex

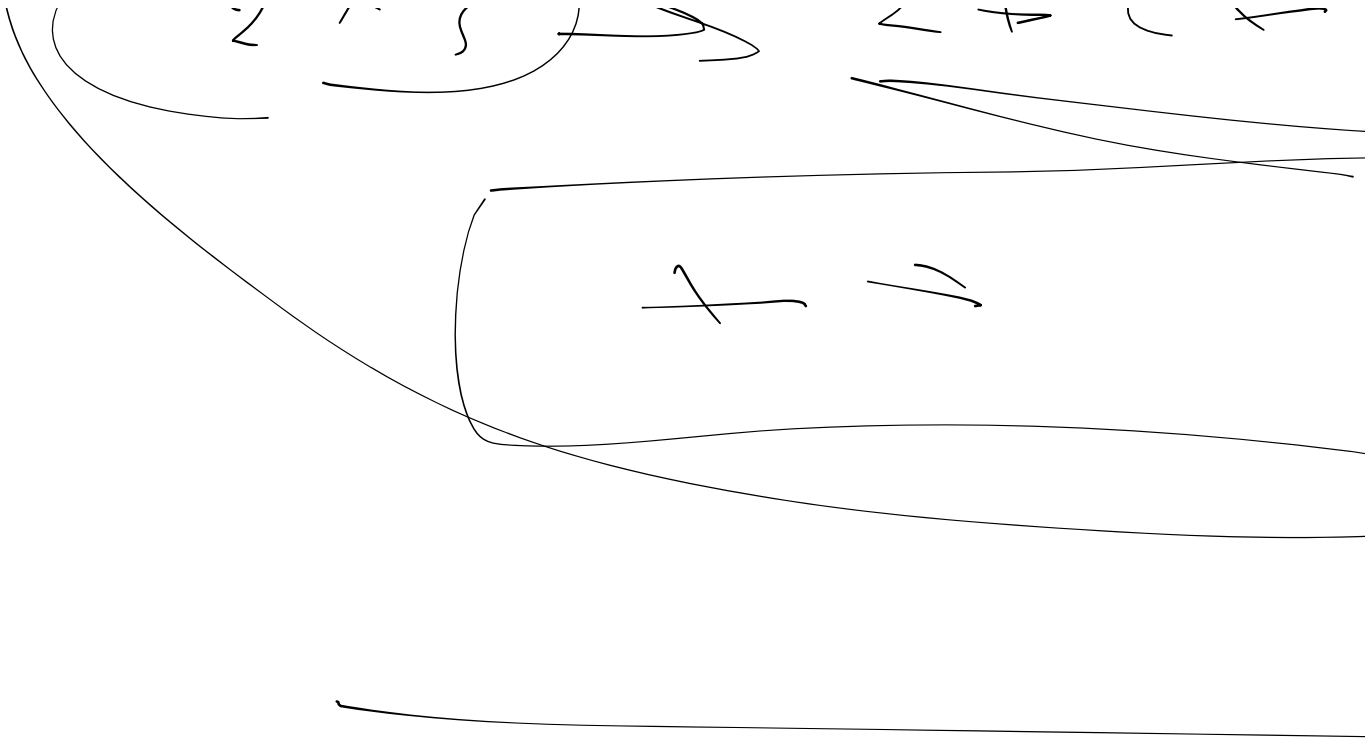
012

↳ octa


0x10
hex
%x → f
%X → F
%#x → 0xf
0xF
%d
y = printf("%d",x);
y=1
x=y
x=4 y=10
sum = printf("%c%c%c",x,'\n',y,'\n
sum = 14
sum = printf("%4c%10c",
sum = 10
+
X
2 X 7
2 + 2







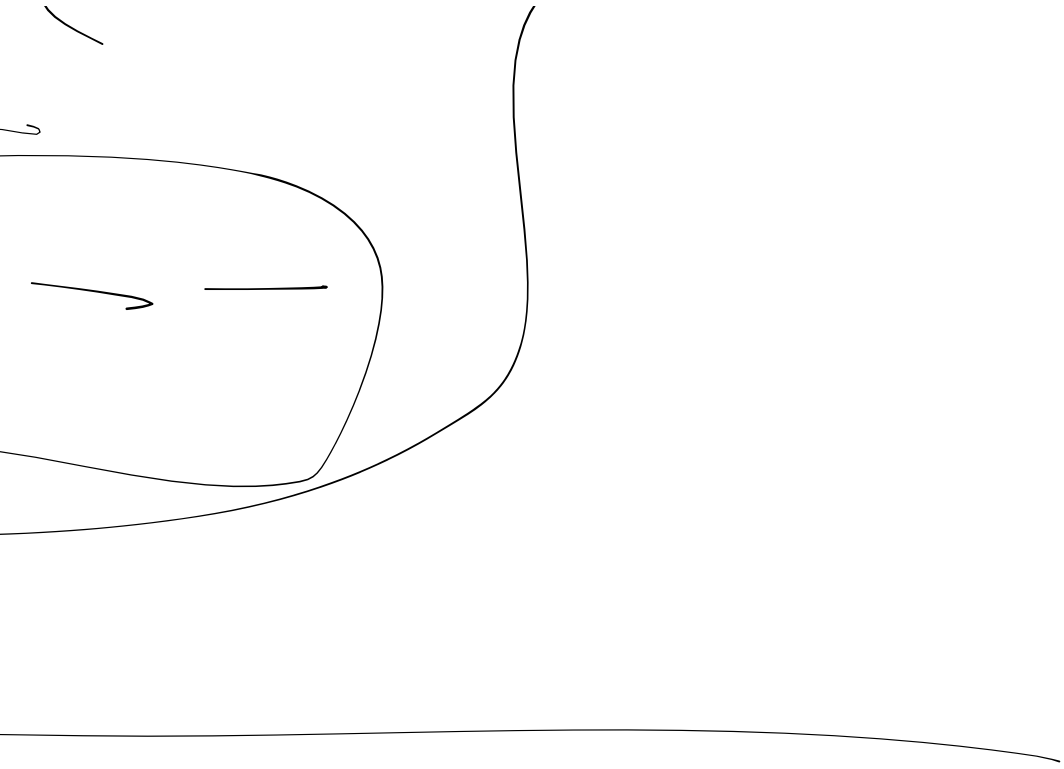
Bitwise

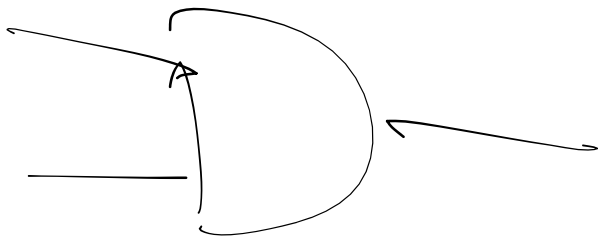
& | ^

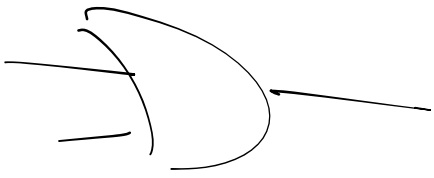
0 & 0 → 0

1 & 1 → 1

1 → 1 | 1 → 1







0 | 0 → 0

^ → 1 ^ 1 → 0
0 0

1 ^ 0 → 1

~ → 0 → 1
1 → 0

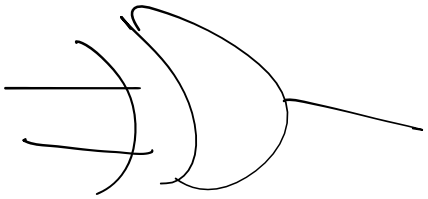
y = 5 → 0101

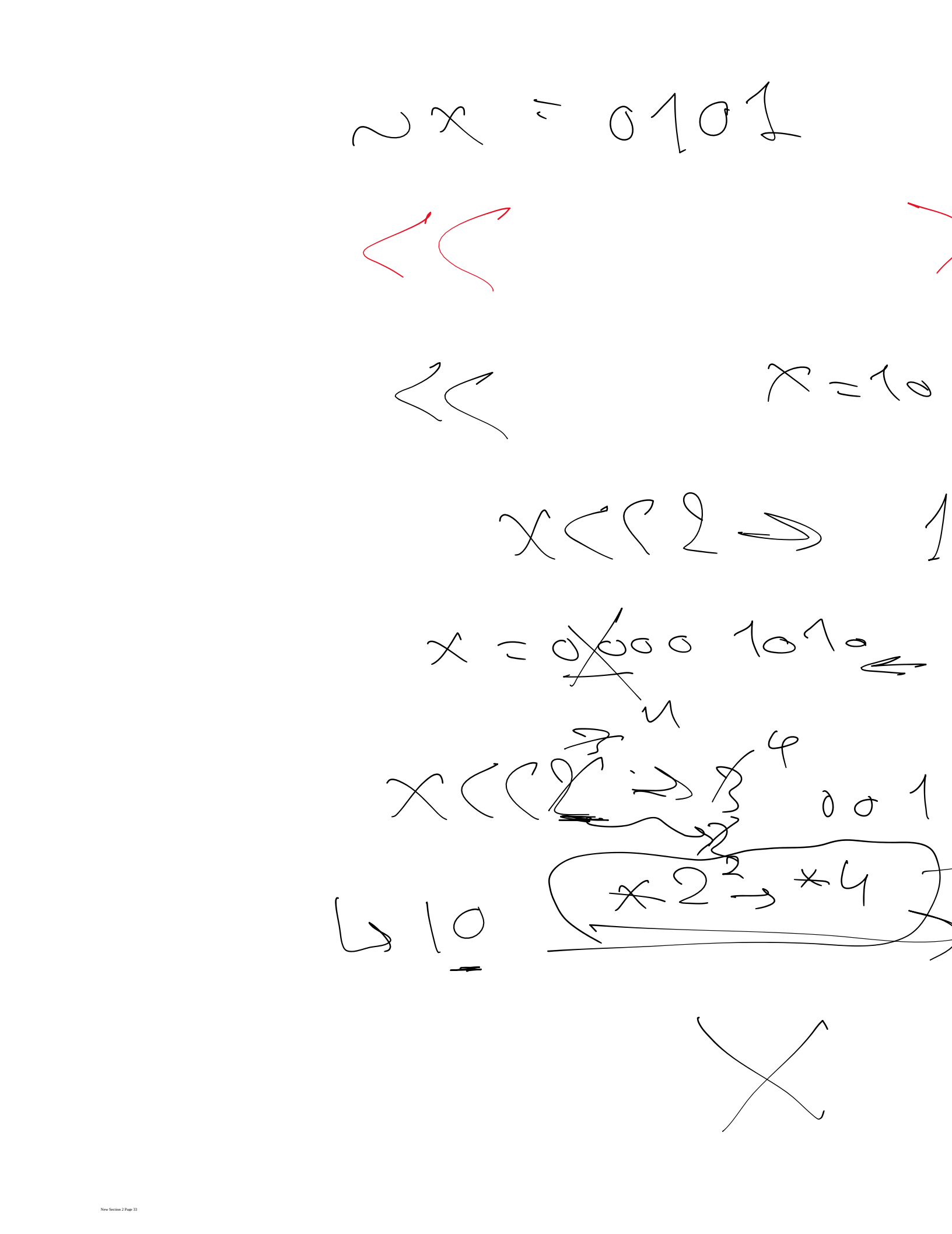
x = 10 → 1010

x & y = 0000

x | y = 1111

x ^ y = 1111




x = 0101
x << 2 → 1
x = 0000 1010
x << 2 → 0 0 1
x 2² → x 4
↳ 10

4 bit

= 10

000

8 bit

0 1000

40

8 bit

>>

x = 10   4 bi

~~1010~~ →

x >> 2 → 0010

0000 ~~1010~~

x >> 2 → 0000 0 ...

$/2^2$

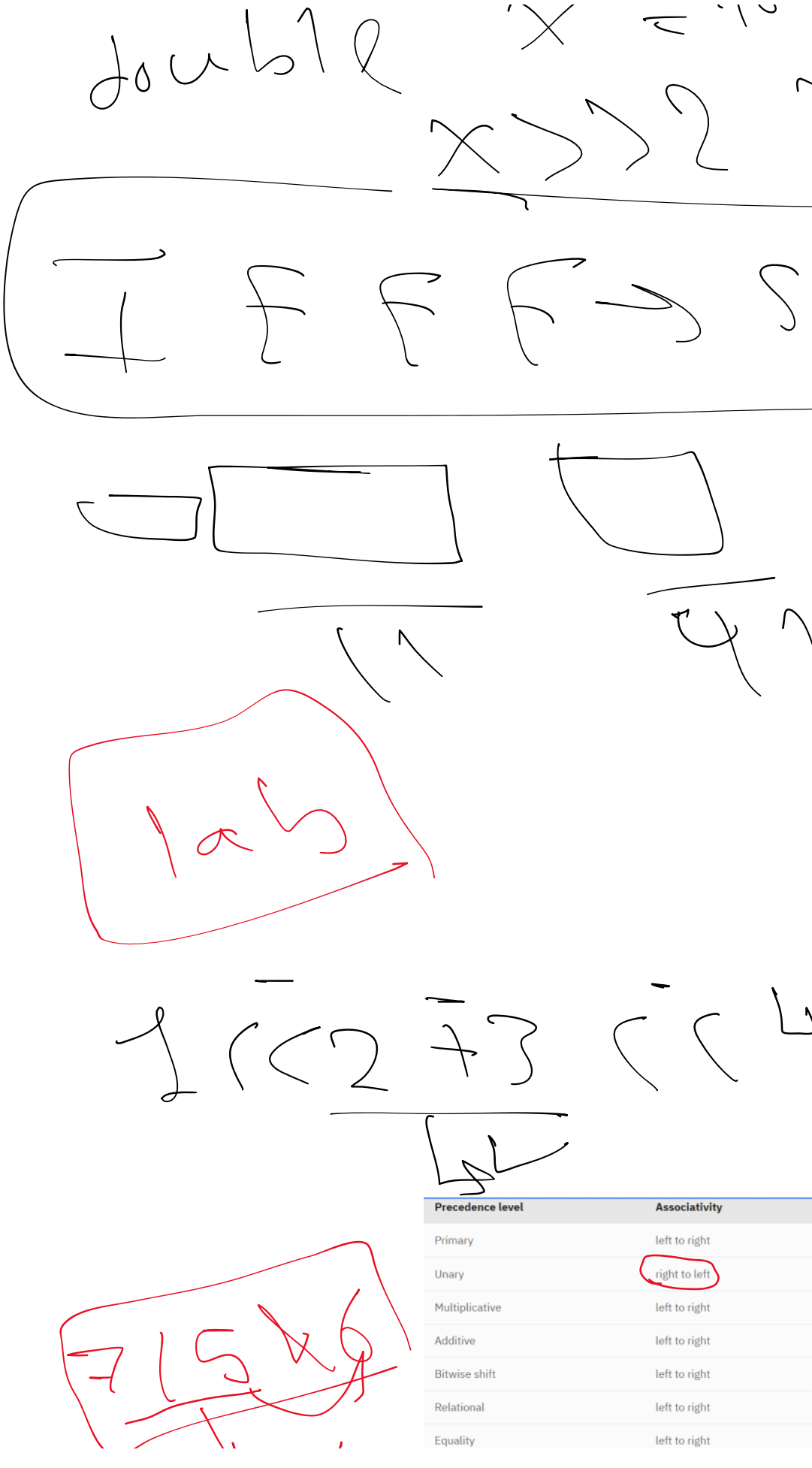
10 —$/2^2$→ 2

^   & | ~   >>

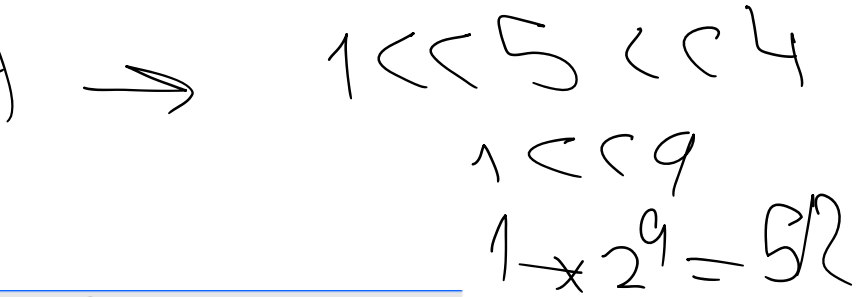
---

integer

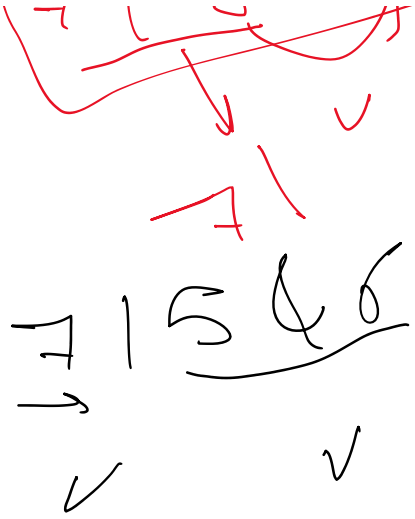
double x = 10



Precedence level	Associativity
Primary	left to right
Unary	right to left
Multiplicative	left to right
Additive	left to right
Bitwise shift	left to right
Relational	left to right
Equality	left to right



	Operators
→	() [] . ->
	++ -- - + ! ~ & * (typename) sizeof
	* / %
	+ -
	<< >>
	< > <= >=
	++ !=



Relational	left to right
Equality	left to right
Bitwise logical AND	left to right
Bitwise exclusive OR	left to right
Bitwise inclusive OR	left to right
Logical AND	left to right
Logical OR	left to right
Assignment	right to left
Comma	left to right

<< >>

< > <= >=

++ !=

& ↓

^ or ¬

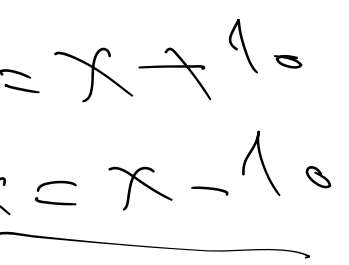
| ↑

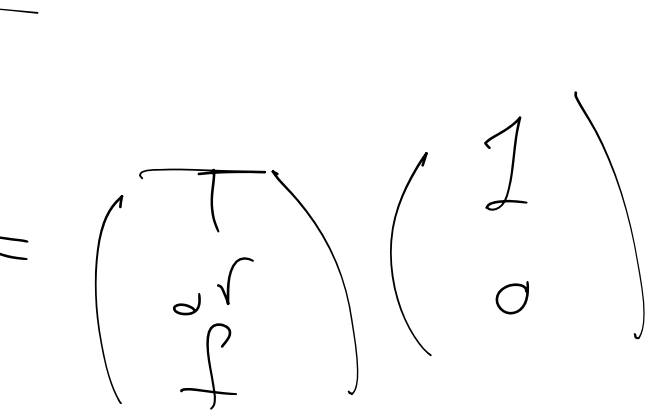
&&

||

= += -= *= /= <<= >>= %=
&= ^= |=

,





$x = 10$ y

$x > y \rightarrow 1$

$y > x \rightarrow 0$

$x == y \rightarrow$

$x != y$

↳ True

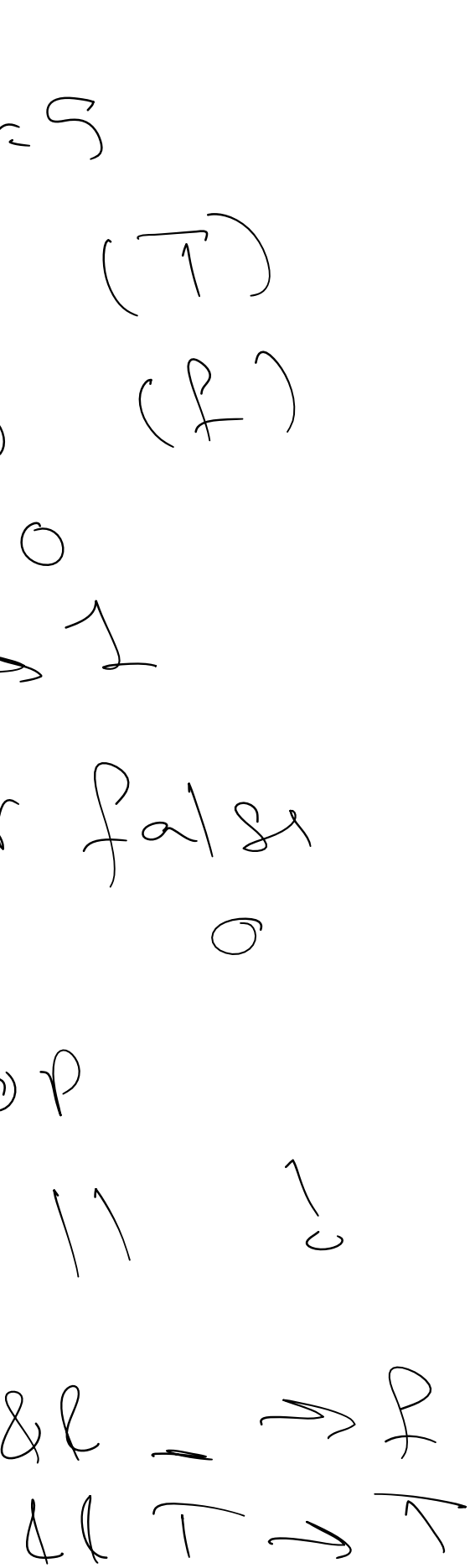
1

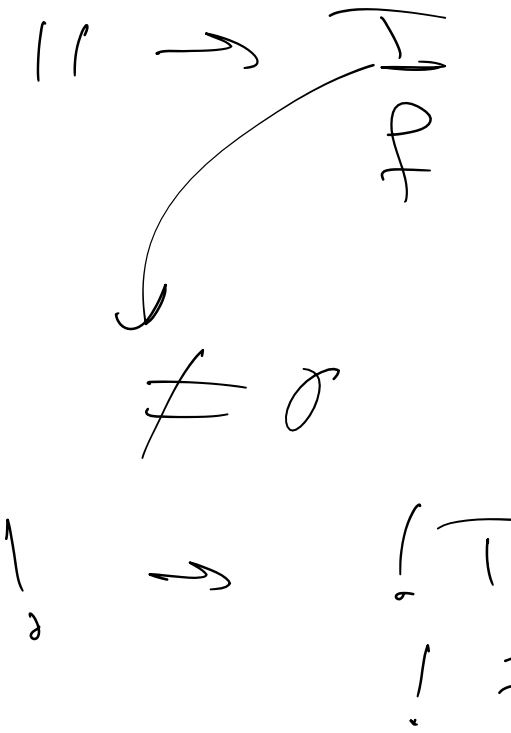
Logical

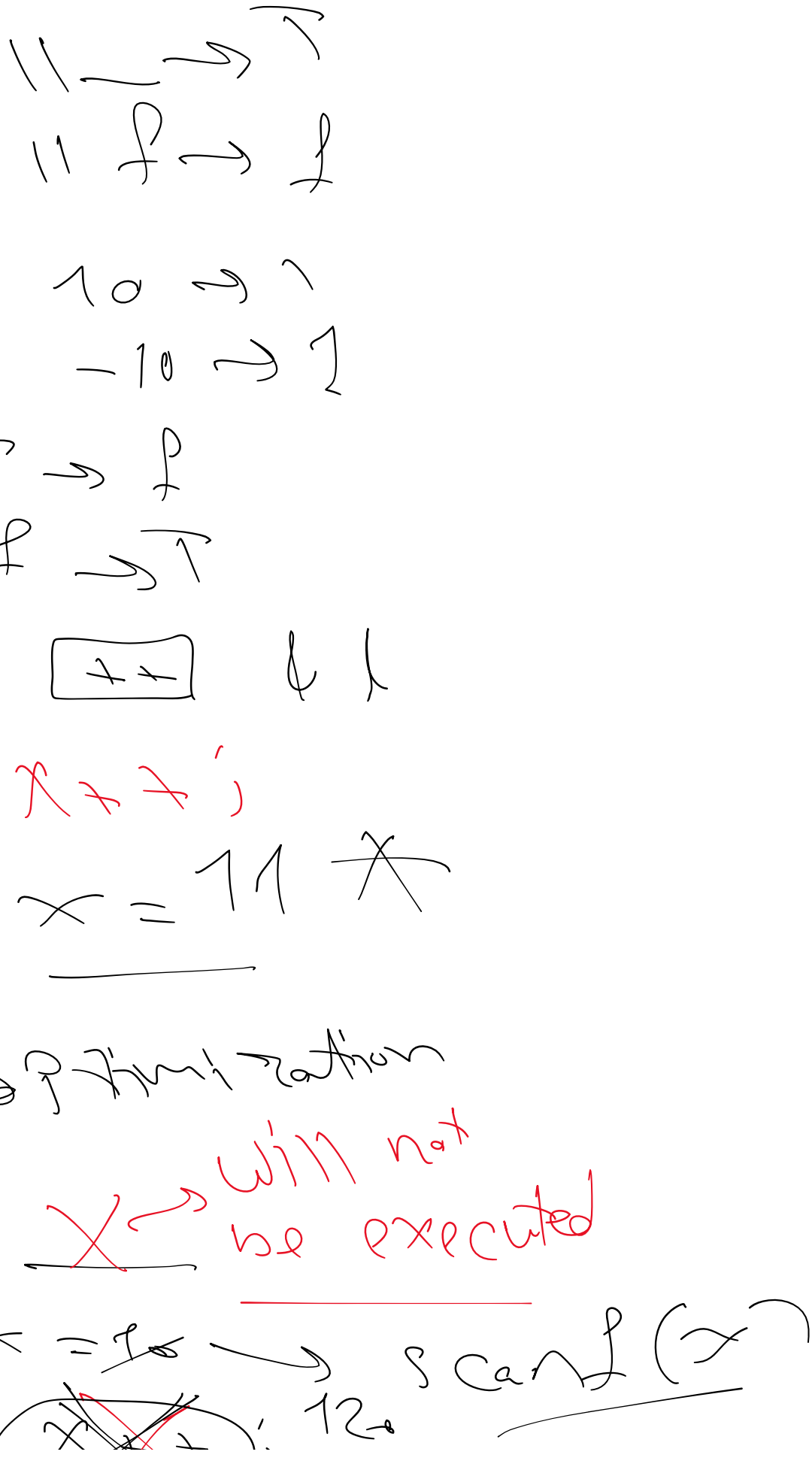
&&

$0 \& \rightarrow F$

F






x++;
x = 11
will not
be executed
scanf(

y = 0 &&

y = 0

x =

y = 1 ||

y = 1

&&

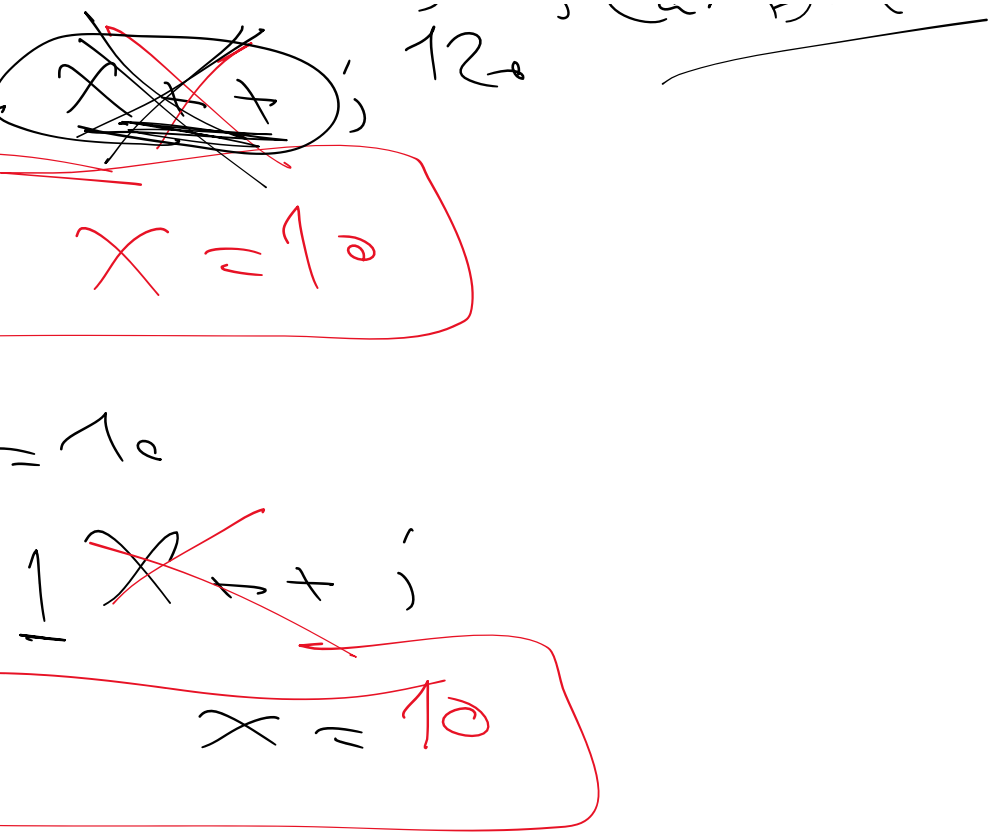
opt

sizeof

sizeof(

sizeof(

int x



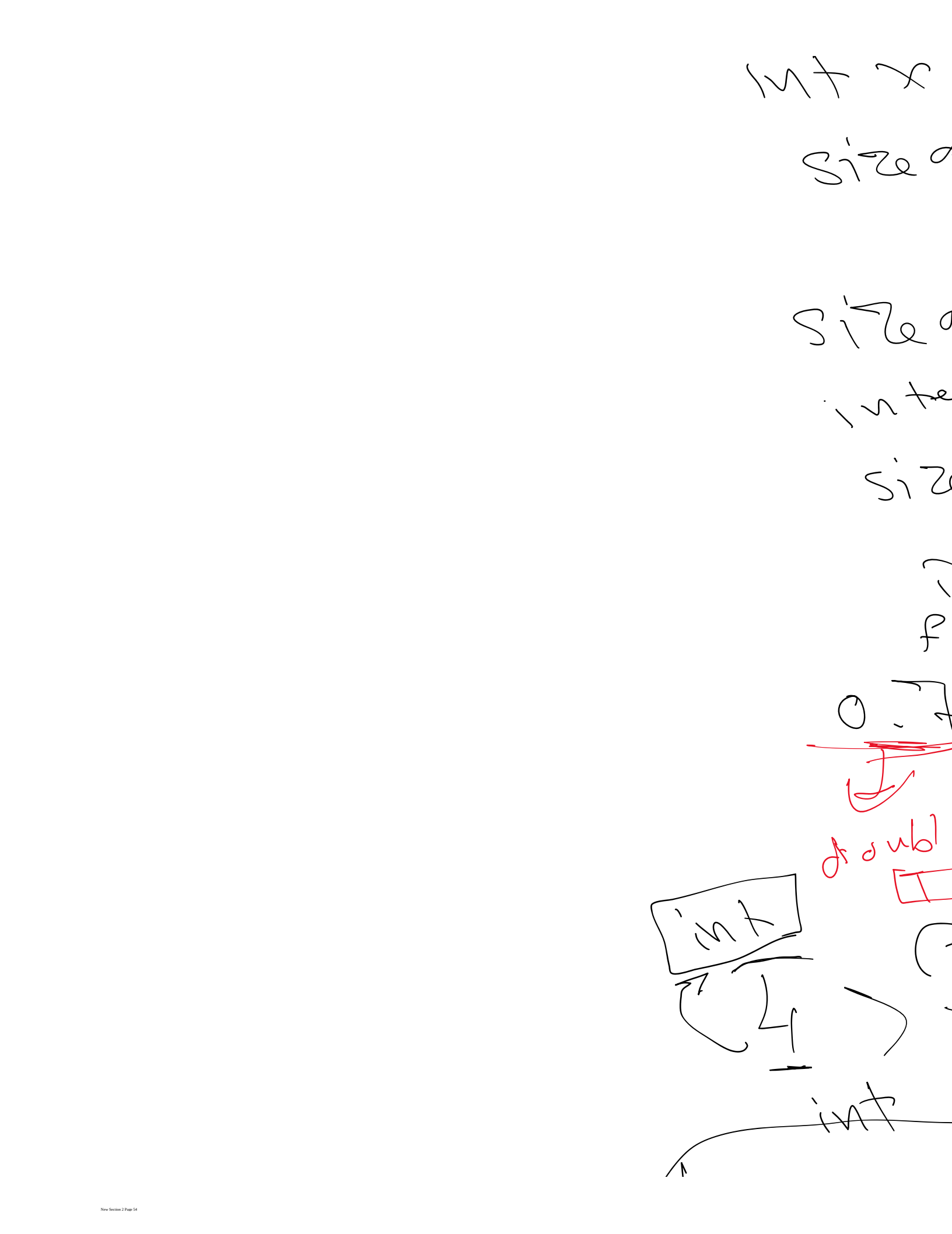
11

return size in bytes

int) = 4 byte

char) = 1 byte

= 10


int x
size o
size o
inte
siz
O . 7
double
int
int

= 10

f(x) = 4 byte

f(10) = 4 byte

ger

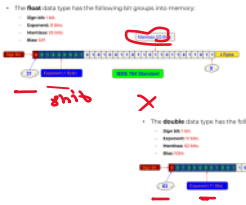
f(0.1) = 8 byte

ouble

loat a = 0.1;

> a → 1

float



8 bits

x

int

1 → 1

double

int / int = int

int / float = float

1 → 0

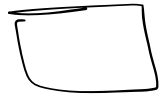
will be idle

sizeof(

unsign

Summ

wh



main

str

(int) > = 1 → 0

ed int

ary:

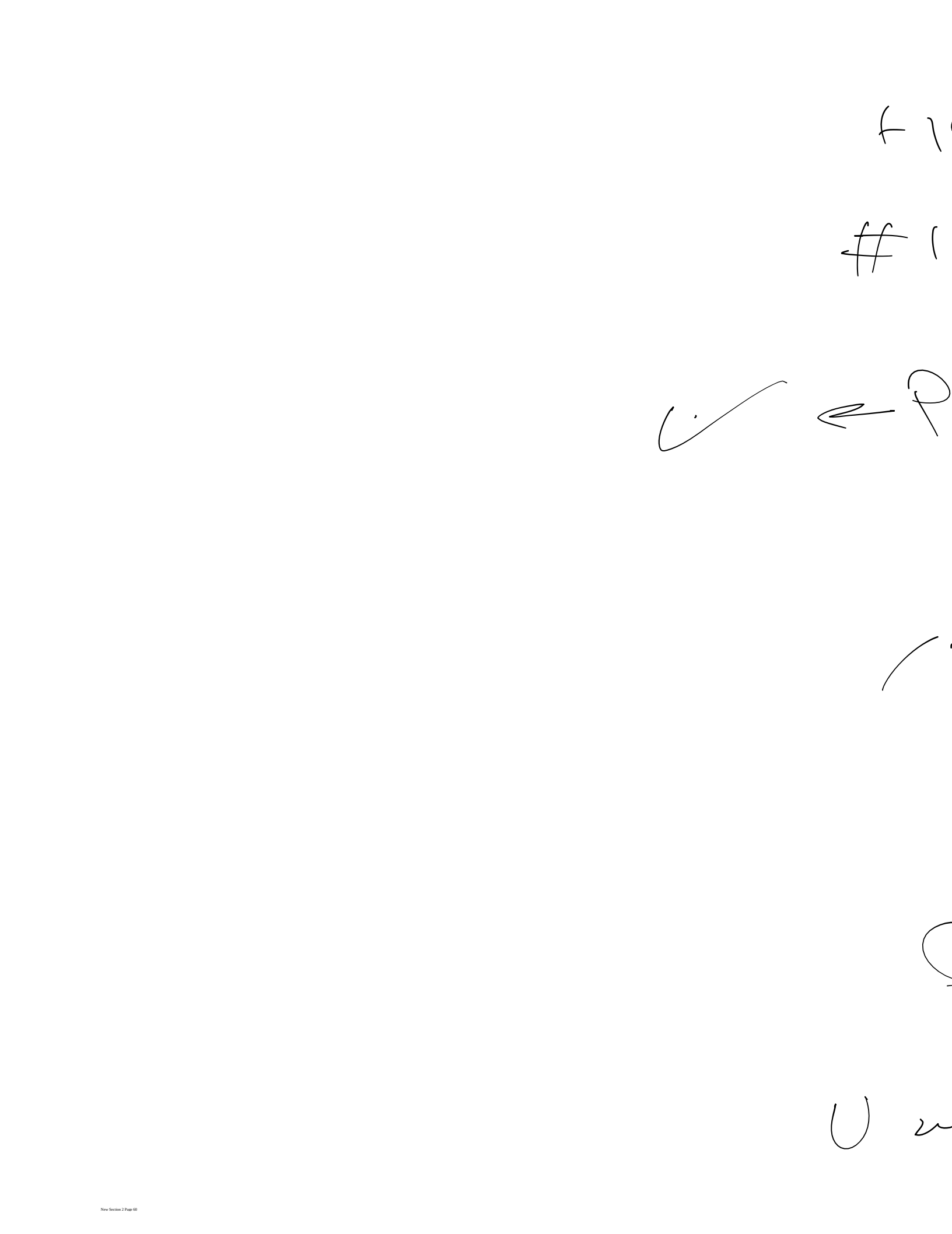
ct c?

→ fanatic

fun → entry point

rup call → main fun

Assembly



ello world

nclude (stdio.h)

rintf("Hello world");

string

* */ → multiline

// → single

clear direction

riable

' → Char

String

garbage

type

type

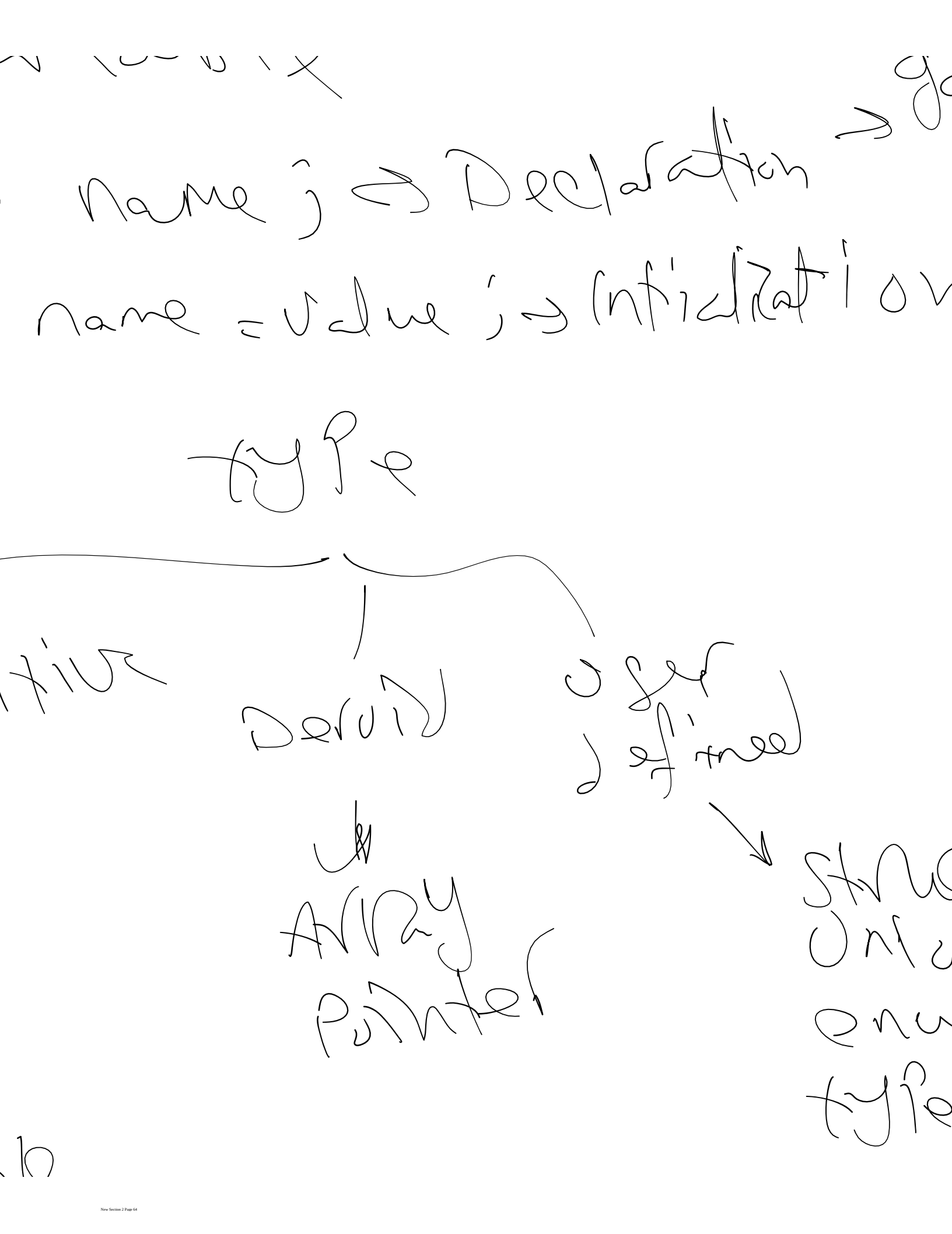
1 byte = 8 bit

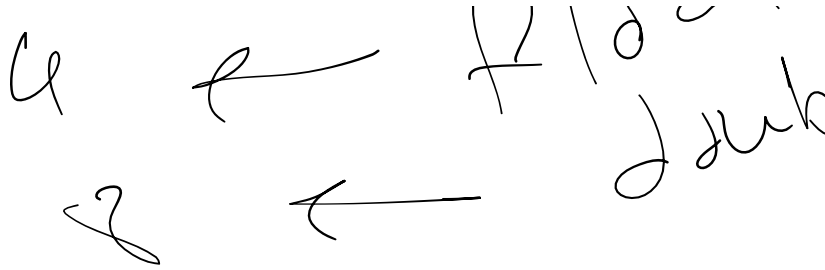
byte

1 ⟷ char

(2) 4 ⟵ int

4 ⟵ float


name ; => Declaration
name = value ; => Initialization
type
Derived
User defined
Array
Pointer



sb

variable name:

A to Z or

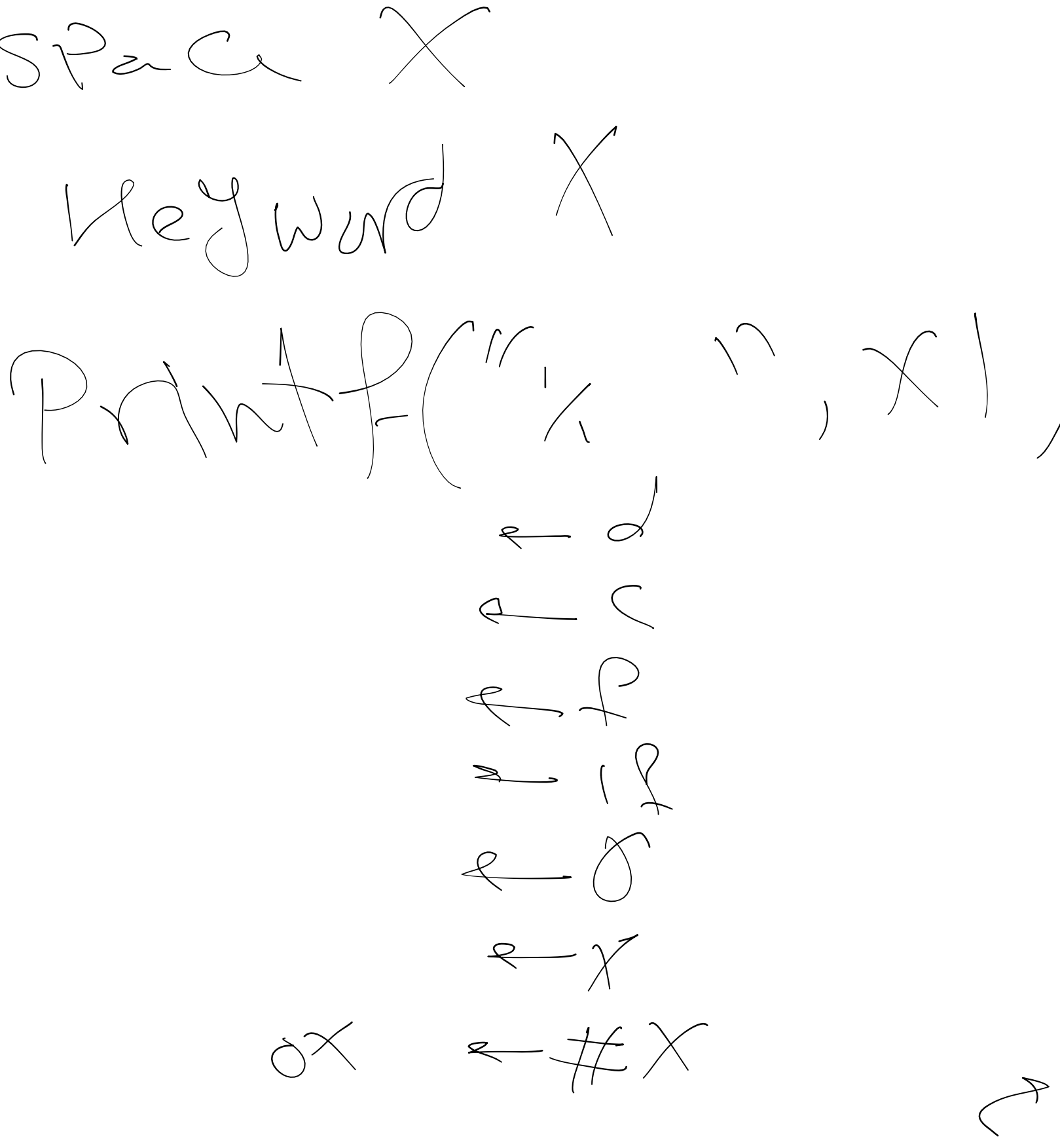
a to z

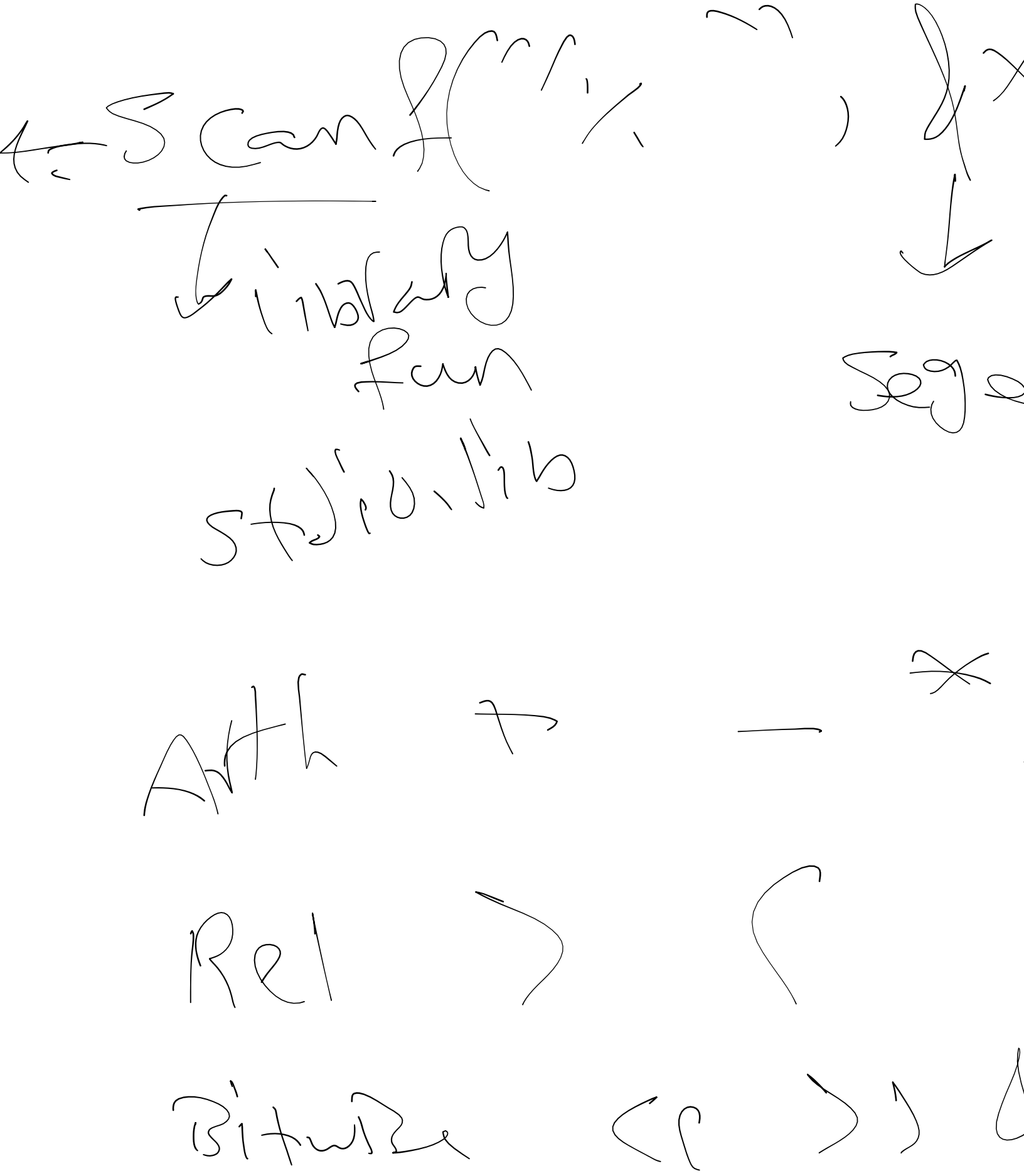
0 to 9 → Not fir

cha

_ underscore →

Special char X


space x
keyword x
printf("%d\n", x);
%d
%c
%f
%lf
%o
%x
%#x



Logical && ||

optimization

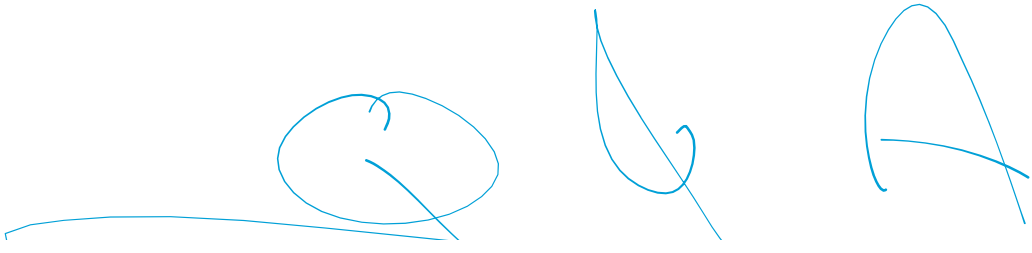
x = 5

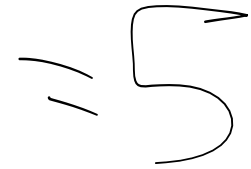
y = x++ → x=6 y

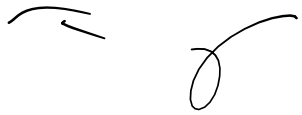
y = ++x → x=6 y

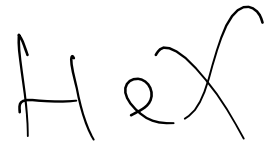
Binary → octa →

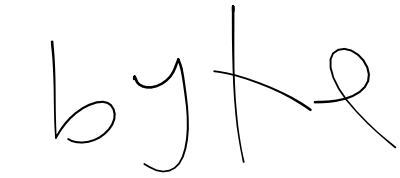
sizeof(int) → 4











int i = 0 , j = 1 , k = 2

m = i++ || j++ || k++

i = 1 j = 2 k = 2 m

m = i++ || j++ || k++

i = 1

m = 0 || j++ || k++ ;

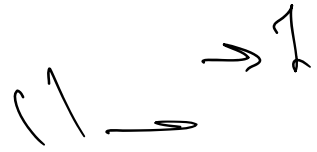
j = 2

m = 0 || 1 || k++ ; 1

m = 1 || k++ ; k =

m = 1

k = 2



i = 1 j = 2 k = 2

int i = 0 , j = 1 , k = 2 , m ;

m = i ++ || j ++ || k ++ ;

m = 0 || j ++ || k ++