Now Let’s come back to C programming

32 bits

```
int y = -9;
```

y | 111....1 | 10111 |

9 = 00....001001

-9 = 1111....1 10111

```
printf("%d", y);

printf("%u", y);
```

32 bits

signed int y = -9;

y | 111 - - - - 1 1011 1 |

```
printf("%d", y);
```

⟹ -9

```
printf("%u", y);
```

⟹ huge No.

```c
int y = -9;
```

y 

```c
printf("%d", y);

printf("%u", y);
```

printf doesn't use the information about type of variable

```
unsigned int y = -9;
```

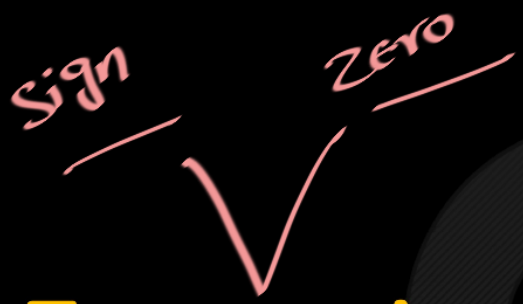
y | 111 - - - - | 1 | 1011 1 |

```
printf("%d", y);
```

→ -9

```
printf("%u", y);
```

→ huge no.



Extension and Truncation

Extension :- copying a lower bit number to higher bit. Eg.- short to int

Truncation :- copying a higher bit number to lower bit. Eg.- int to short

Extension

16 bits

```
short int x = 9;

int ix = x;
```

short x = 9;

short signed int x = 9

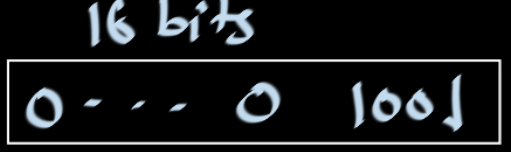
Extension

16 bits

```
short int x = 9;

int ix = x;
```

32 bits

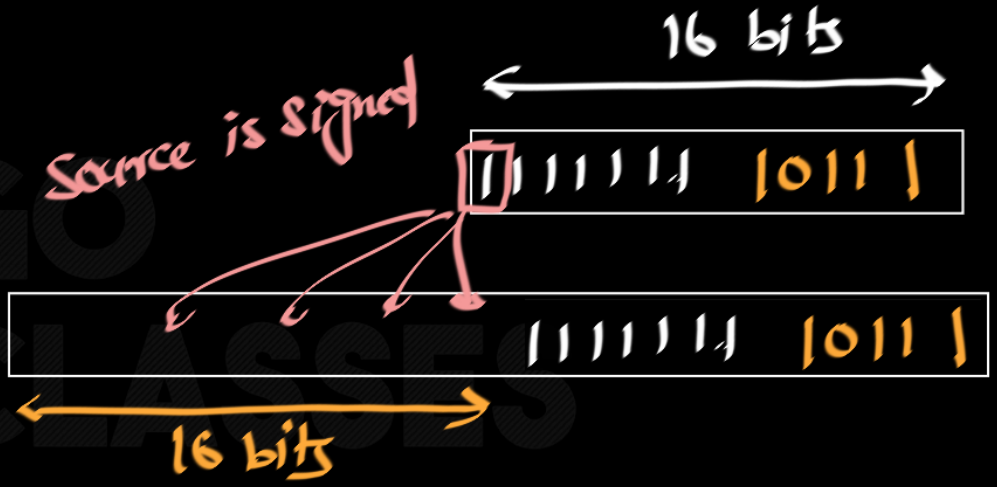
x



ix

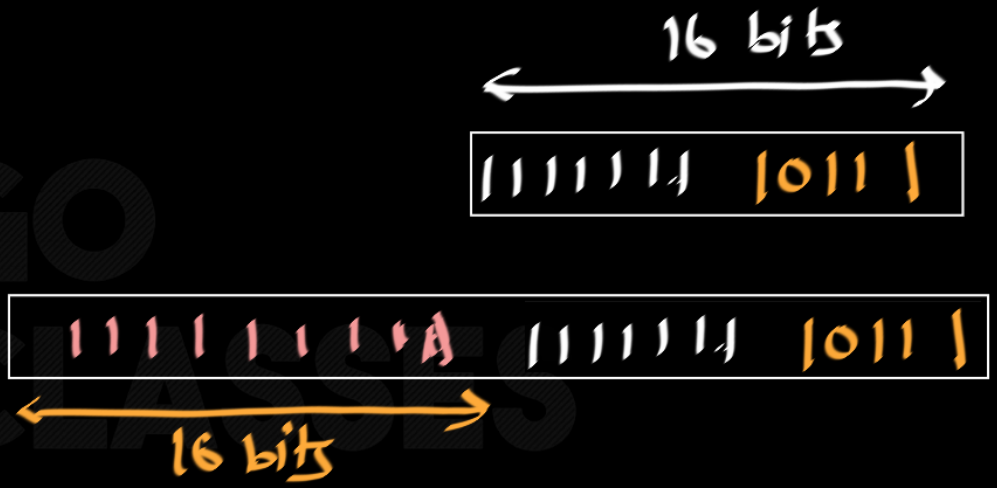
Extension

```
short int y = -9;
int iy = y;
```



Extension

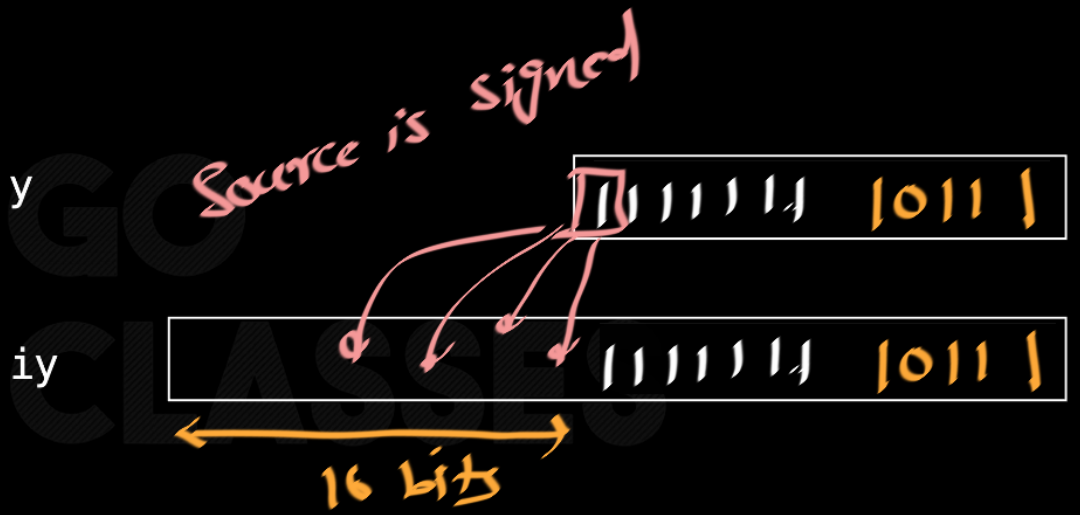
```
short int y = -9;
int iy = y;
```



```
short int x = 9;
int ix = x;
short int y = -9;
int iy = y;
```

	Decimal	Binary
x	9	00000000 00001001
ix	9	00000000 00000000 00000000 00001001
y	-9	11111111 11110111
iy	-9	11111111 11111111 11111111 11110111

```
short int y = -9;

unsigned int iy = y;
```



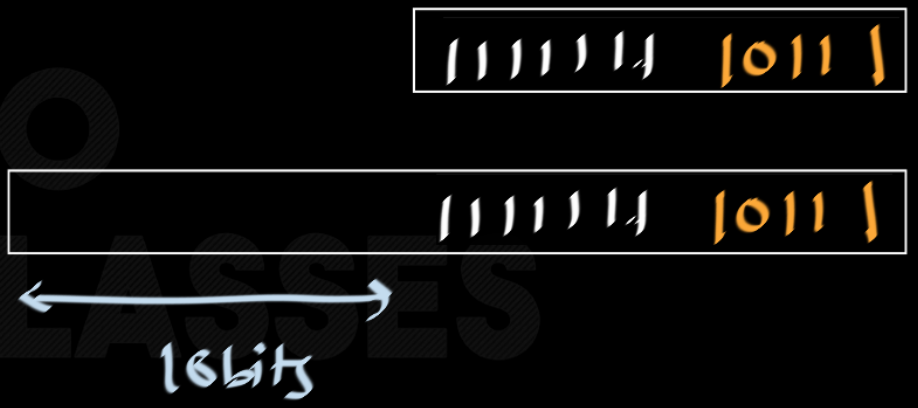
```
short int y = -9;
```

```
unsigned int iy = y;
```

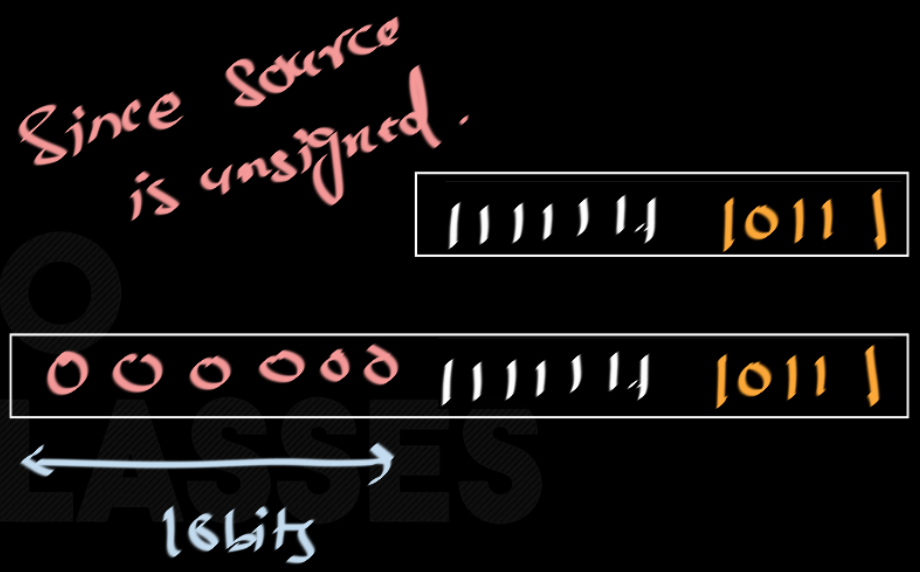
y

iy

```
unsigned short int y = -9;
int iy = y;
```



```
unsigned short int y = -9;
int iy = y;
```



```
unsigned short int y = -9;
unsigned int iy = y;
```

y

Since source is unsigned

iy

16 bit

Extension Depends on RHS (Source)

imp

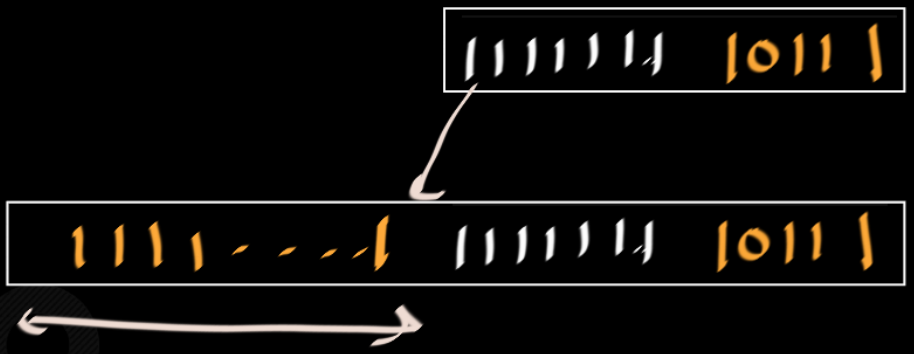
- Promotion always happens according to the *source* variable's type
  - Signed: **"sign extension"** (copy MSB—0 or 1—to fill new space)
  - Unsigned: **"zero fill"** (copy 0's to fill new space)

Stanford University

https://web.stanford.edu/class/archive/cs/cs107/cs107.1174/lect8.pdf

```
short int y = -9;
```

y



```
unsigned int iy = y;
```

iy

```
printf("%d", y);
printf("%u", y);
printf("%d", iy);
printf("%u", iy);
```

-9

huge no.

-9

huge no.

```
unsigned short int y = -9;
int iy = y;
```

y

iy

16 bits

```
printf("%d", y);
printf("%u", y);
printf("%d", iy);
printf("%u", iy);
```

huge

huge

huge

huge

```
int x = -1;
unsigned int u = x;
```


```
printf("%d", x);
printf("%u", x);
printf("%d", u);
printf("%u", u);
```

-1
Huge
-1
Huge

GO CLASSES

Optional

2.2.6 Expanding the Bit Representation of a Number

One common operation is to convert between integers having different word sizes while retaining the same numeric value. Of course, this may not be possible when the destination data type is too small to represent the desired value. Converting from a smaller to a larger data type, however, should always be possible. To convert

an unsigned number to a larger data type, we can simply add leading zeros to the representation; this operation is known as *zero extension*. For converting a two's-complement number to a larger data type, the rule is to perform a *sign extension*, adding copies of the most significant bit to the representation. Thus, if our original value has bit representation $[x_{w-1}, x_{w-2}, \ldots, x_0]$, the expanded representation is $[x_{w-1}, \ldots, x_{w-1}, x_{w-1}, x_{w-2}, \ldots, x_0]$. (We show the sign bit x_{w-1} in blue to highlight its role in sign extension.)

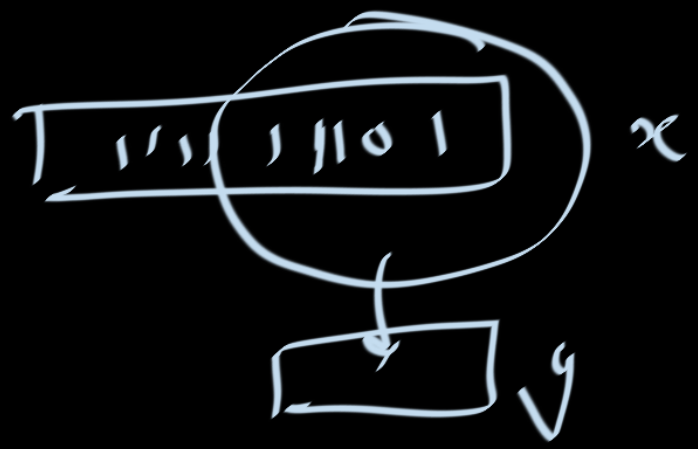
Truncation (higher bits to lower bits)

- Regardless of source or destination signed/unsigned type, truncation always just truncates
- This can cause the number to **change drastically in sign and value**

Stanford University

GO CLASSES

```
int   x =  -9
short y =  x;
```



- How to represent a number in binary
- How to copy a number from one to another.

```
int x = -9;
```

The integer promotions

Whenever a small integer type (char or short) is used in an **expression**, it is implicitly converted to int .

https://stackoverflow.com/a/46073296/3699192

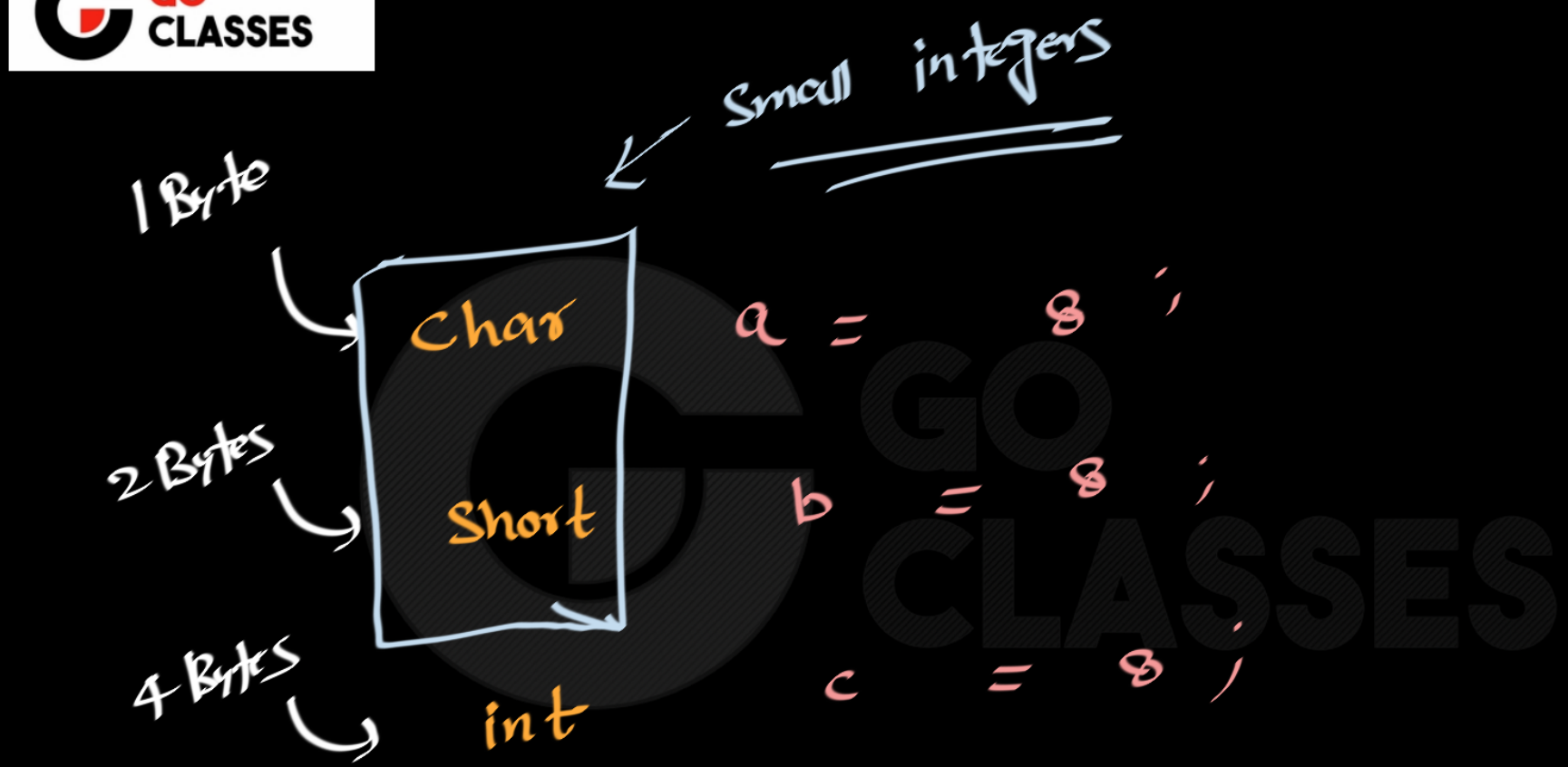
GO CLASSES

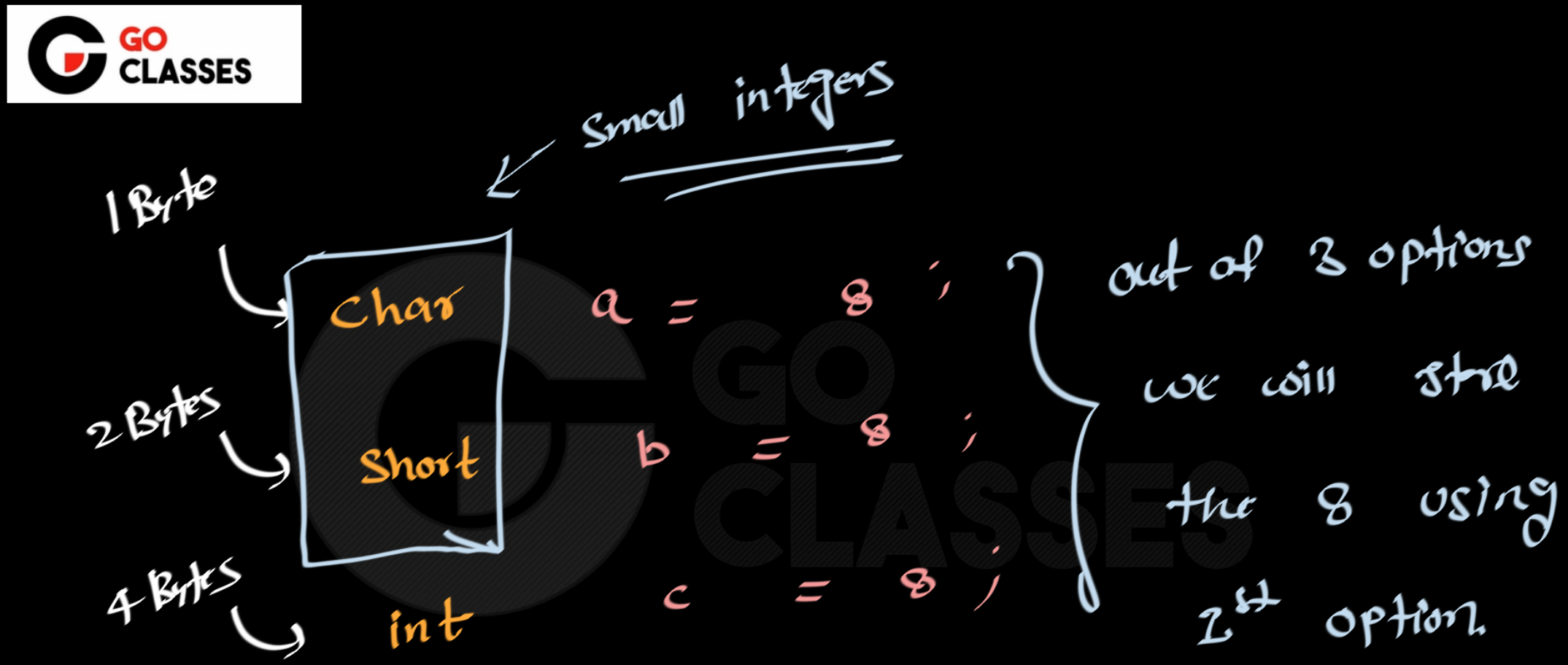
all of these are family of integers

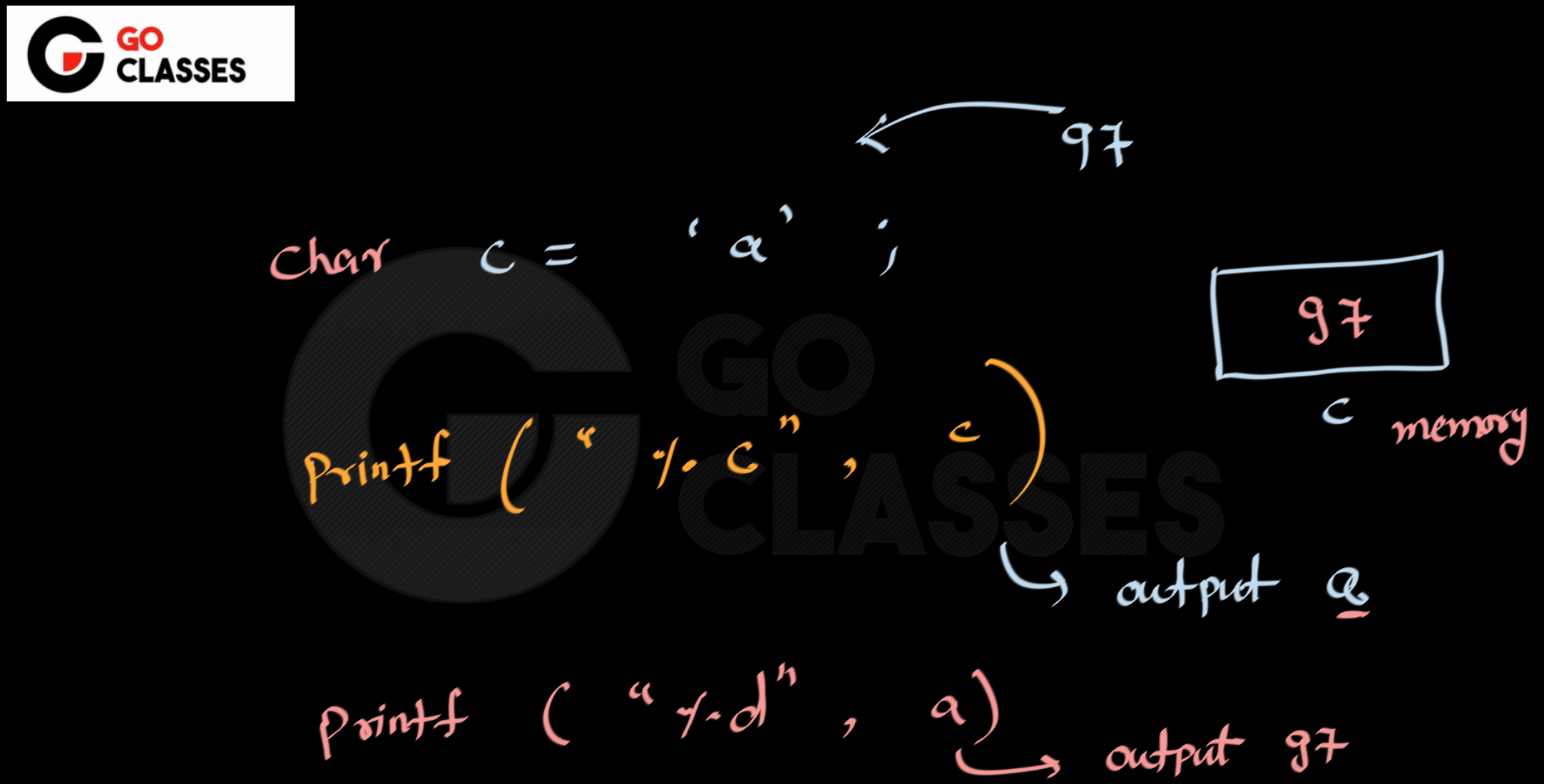
1 Byte → Char a = 8 ;

2 Bytes → Short b = 8 ;

4 Bytes → int c = 8 ;


Small integers
1 Byte
Char
a = 8 ;
2 Bytes
Short
b = 8 ;
4 Bytes
int
c = 8 ;


GO CLASSES
Small integers
1 Byte
2 Bytes
4 Bytes
Char
Short
int
a = 8 ;
b = 8 ;
c = 8 ;
out of 3 options
we will store
the 8 using
2nd option

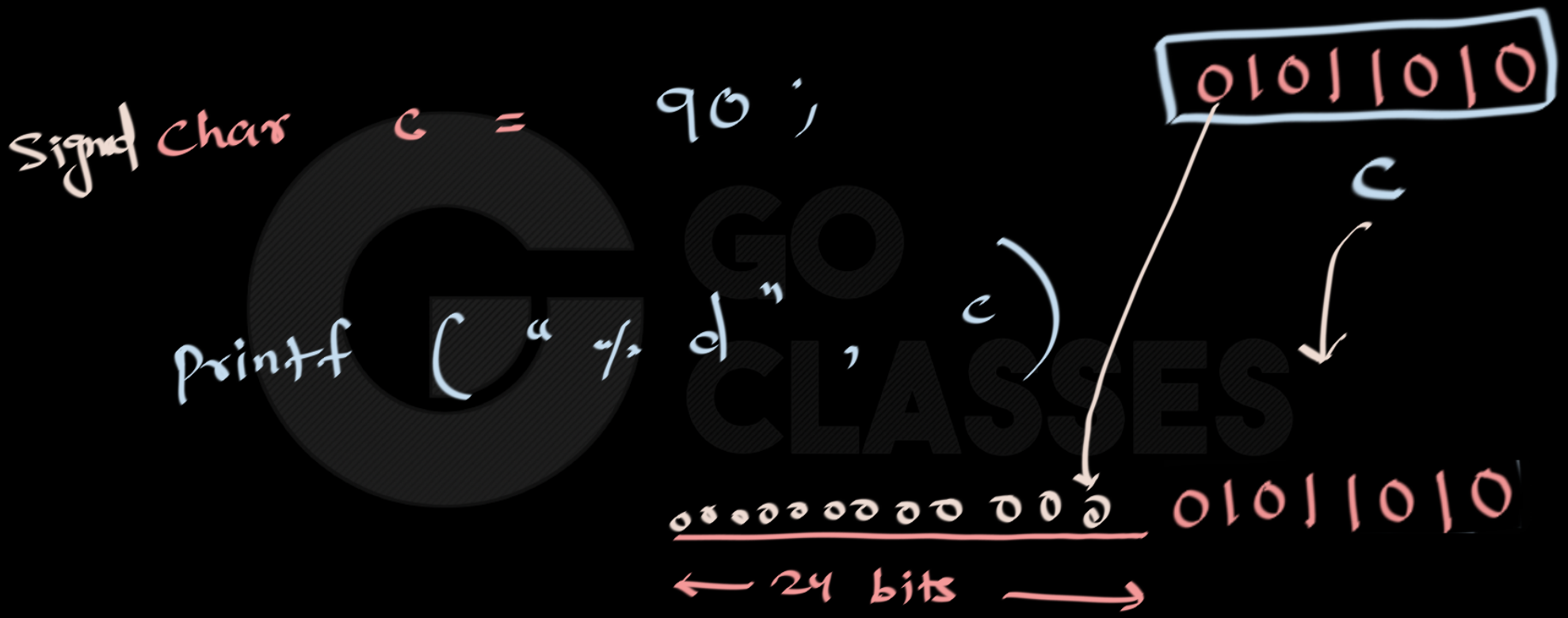

GO CLASSES
97
Char c = 'a' ;
97
c memory
Printf (" %c " , c)
output a
Printf (" %d " , a)
output 97

97

Char c = 'a' ;

c+1 = 97+1 = 98

c^2-1 = 97^2-1


Signed Char c = 90 ;
printf (" %d " , c)
01011010
c
00000000 00000000 01011010
← 24 bits →

GO CLASSES

signed char c = 90 ;

0	1	0	1	1	0	1	0

c

printf ("%d", c)

↳ 90

GO CLASSES

Signed char x = 130 ;

printf (" %d " , x)

GO CLASSES

Signed char x = 130 ;

printf (" %d " , x)

→ -126

GO CLASSES

Signed char x = 130 ;

1	0	0	0	0	0	1	0

x

printf ("%u", x)

↳ a huge no.

1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 0

24 bits

GO CLASSES

unsigned char x = 130 ;

1	0	0	0	0	0	1	0

x

printf (" %u " , x)

↳ 130

0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0

24 bits

GO CLASSES

C a

Selection

```c
#include<stdio.h>

int main(){


    unsigned char x = 130;
    printf("%d\n",x);

    printf("%u",x);


}
```

Desktop — -zsh — 80×24

```
(base) sachinmittal@Sachins-MacBook-Pro Desktop % gcc a.c
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out
130
130%
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
```

GO CLASSES

Signed char a = 258;

258

1 0 0 0 0 0 0 1 0

printf ("%d", a) → 2

printf ("%u", a) → 2

printf ("%c", a)

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

a

GO CLASSES

Signed char a = 258 + 128;

258

1 1 0 0 0 0 0 1 0

Printf ("%d", a) → -126

Printf ("%u", a) → huge no

1 0 0 0 0 0 1 0

a

11111111 (24 bits)

1 0 0 0 0 0 1 0

Printf ("%c", a)

GO CLASSES

unsigned char a = 258 + 128;

258

1 1 0 0 0 0 0 1 0

1 0 0 0 0 0 1 0

a

printf ("%d", a) → 130

printf ("%u", a) → 130

printf ("%c", a)

0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0

24 bits

```
signed char a = 30, b = 40;
signed char d = a * b;

printf ("%d ", d);
printf ("%d ", a * b);
```

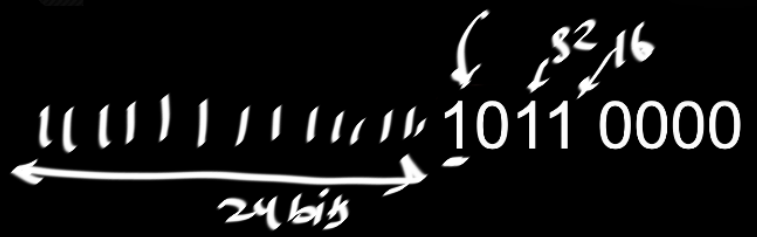
-80

1200

1200 in binary is
0000 0100 1011 0000

1011 0000

-120



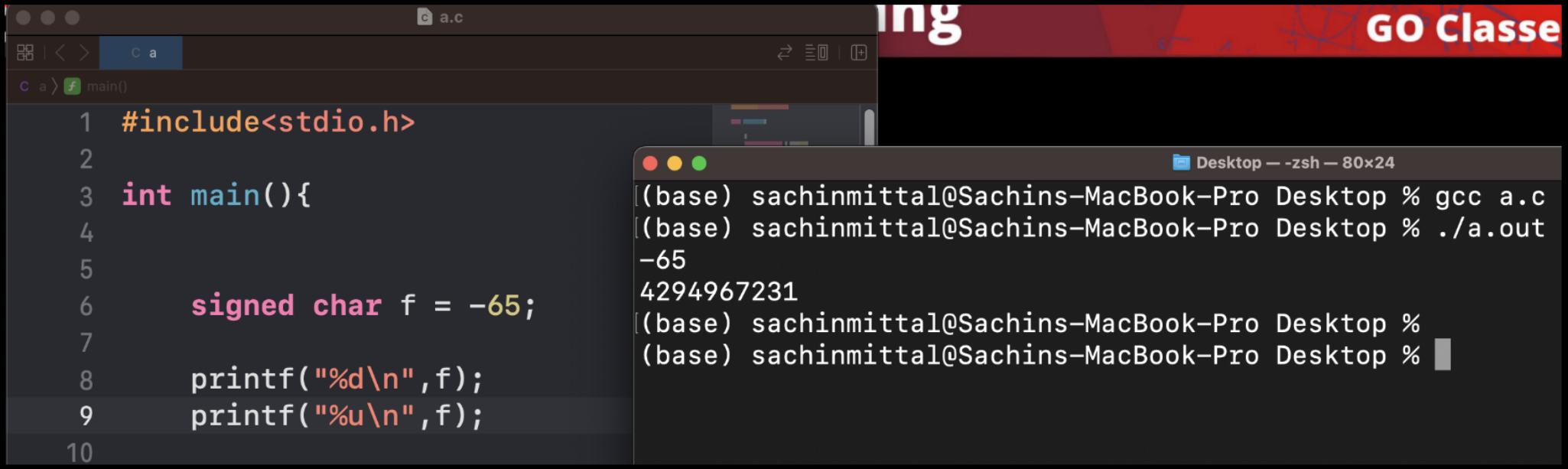
GO CLASSES

```c
#include<stdio.h>

int main(){


    signed char a = 30, b = 40;
    signed char d = a*b;

    printf("%d\n",d);
    printf("%d\n",a*b);
```

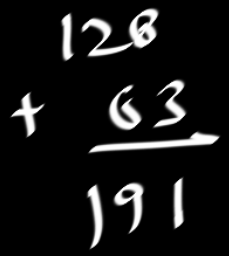
```
Desktop — -zsh — 80×24
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
-80
1200
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
```

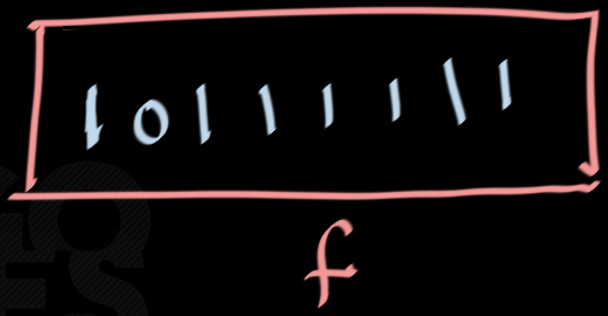
```
signed char f =-65;

printf("%d", f);

printf("%u", f);
```

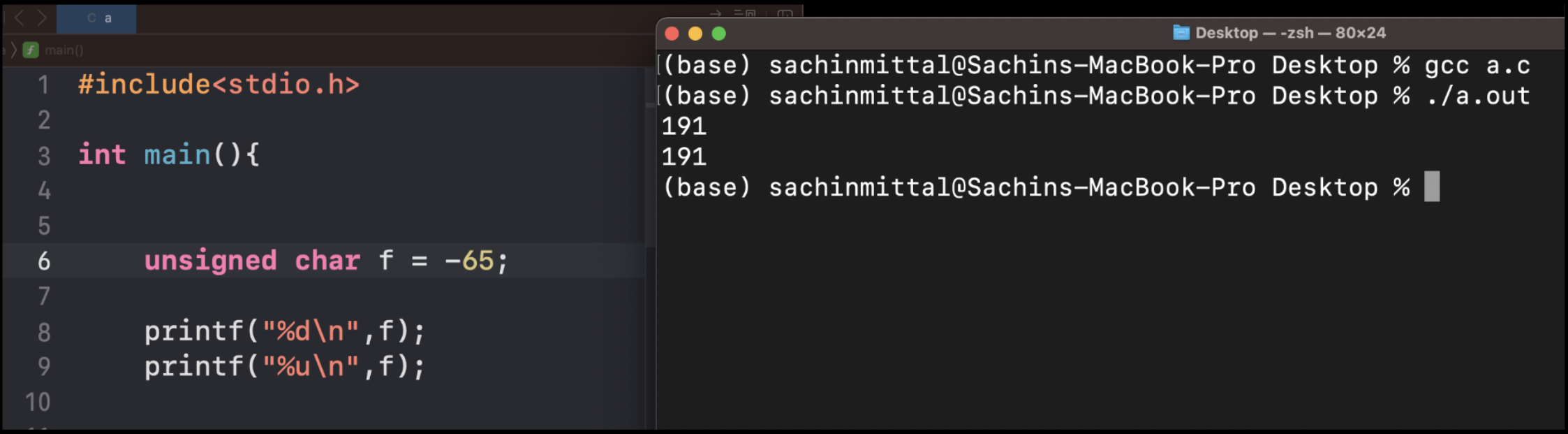
GO CLASSES


a.c
c a
c a > main()
1 #include<stdio.h>
2
3 int main(){
4
5
6 signed char f = -65;
7
8 printf("%d\n",f);
9 printf("%u\n",f);
10
ing
GO Classe
Desktop — -zsh — 80×24
(base) sachinmittal@Sachins-MacBook-Pro Desktop % gcc a.c
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out
-65
4294967231
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
(base) sachinmittal@Sachins-MacBook-Pro Desktop %

```
unsigned char f =-65;

printf("%d", f);
printf("%u", f);
```






a
main()
1 #include<stdio.h>
2
3 int main(){
4
5
6 unsigned char f = -65;
7
8 printf("%d\n",f);
9 printf("%u\n",f);
10
Desktop — -zsh — 80×24
(base) sachinmittal@Sachins-MacBook-Pro Desktop % gcc a.c
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out
191
191
(base) sachinmittal@Sachins-MacBook-Pro Desktop %

GO CLASSES

The integer promotions

Whenever a small integer type (char or short) is used in an **expression**, it is implicitly converted to int .

https://stackoverflow.com/a/46073296/3699192

GO CLASSES

Summary

1) Numbers ⇒ -3 , 3 have fixed repres.

2) printf doesn't make use of the informatn of data type

3) Extension ↔ (truncation ↔)

4) integer promotion.

Optional : 2'S Complement

$$\begin{array}{r} 0101 \\ +\,???? \\ \hline 0000 \end{array}$$

$$\begin{array}{r} {\scriptstyle 1\ \ 1} \\ 0101 \\ +\,1011 \\ \hline 0000 \end{array}$$

Answer

$$\begin{array}{r} 0101 \\ +1011 \\ \hline 0000 \end{array}$$

$$\begin{array}{r} 0011 \\ +\,???? \\ \hline 0000 \end{array}$$

$$\begin{array}{r} {\scriptstyle 1}\quad {\scriptstyle 1} \\ 00\overset{1}{1}1 \\ +\;1\;1\;0\;1 \\ \hline 0000 \end{array}$$

$$\begin{array}{r} 0011 \\ +1101 \\ \hline 0000 \end{array}$$

$$\begin{array}{r} 0000 \\ +\,???? \\ \hline 0000 \end{array}$$

Answer

$$\begin{array}{r} 0000 \\ +0000 \\ \hline 0000 \end{array}$$

There Seems Like a Pattern Here...

0101	0011	0000
+1011	+1101	+0000
0000	0000	0000

GO CLASSES

Why adding

with

2's Complement gives zeros ?

Why adding with 2's Complement gives zeros ?

$$\begin{array}{r} 0101 \\ +1011 \\ \hline 0000 \end{array}$$

Why adding with 2's Complement gives zeros ?

$$\begin{array}{r} 0101 \\ +1011 \\ \hline 0000 \end{array}$$

$$\begin{array}{r} 0101 \\ +1010 \\ \hline 0001 \\ \hline 0000 \end{array} \qquad 1\ 1\ 1\ 1$$

Any number added with One's complement gives all One's.

Why adding with 2's Complement gives zeros ?

$$\begin{array}{r} 0101 \\ +1011 \\ \hline 0000 \end{array}$$

$$\begin{array}{r} 0101 \\ +\ 1010 \\ \quad 0001 \\ \hline 0000 \end{array}$$

$$\begin{array}{r} 1111 \\ +\ 0001 \\ \hline 0000 \end{array}$$

One's Complement

A binary number plus its inverse is all 1s.

$$\begin{array}{r} 0101 \\ +1010 \\ \hline 1111 \end{array}$$

Add 1 to this to carry over all 1s and get 0!

$$\begin{array}{r} 1111 \\ +0001 \\ \hline 0000 \end{array}$$

GO CLASSES

Trick

to find 2's Complement of a number

Trick

To find the negative equivalent of a number, work right-to-left and write down all digits through when you reach a 1. Then, invert the rest of the digits.

$$\begin{array}{r} 100100 \\ +\,?????? \\ \hline 000000 \end{array}$$


1
100100
+ 01100
000000

Trick

To find the negative equivalent of a number, work right-to-left and write down all digits through when you reach a 1. Then, invert the rest of the digits.

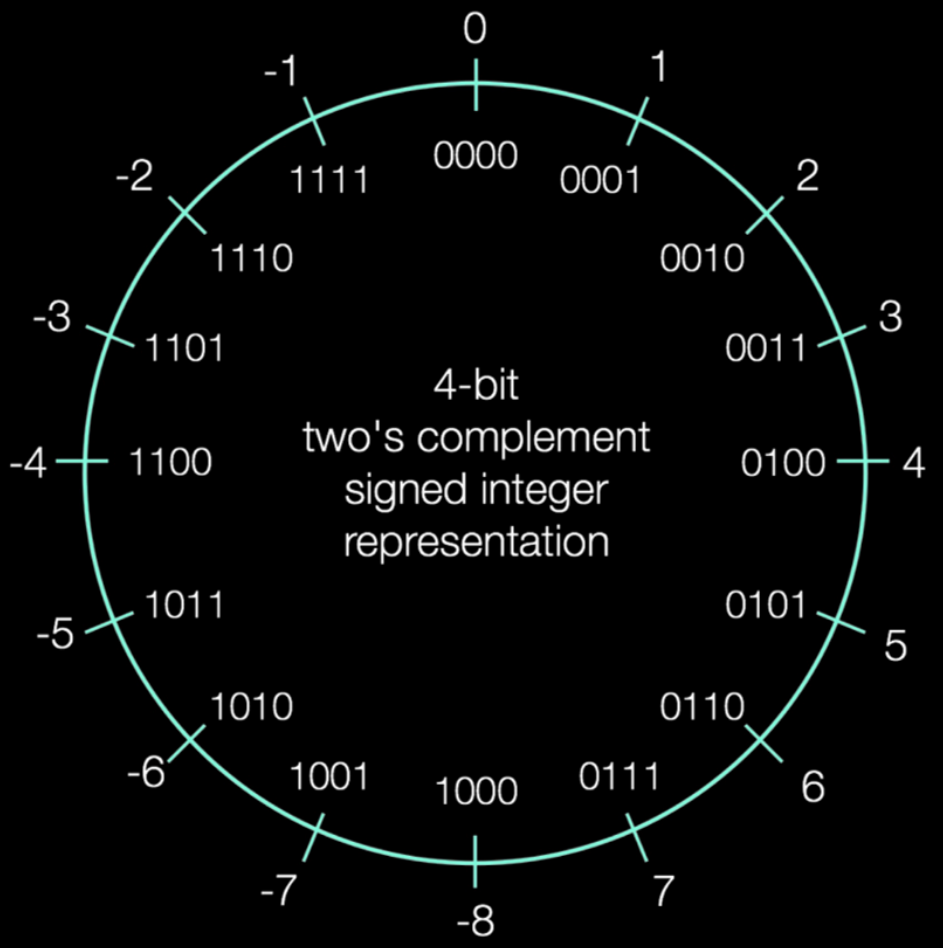
$$\begin{array}{r} 100100 \\ +\,???100 \\ \hline 000000 \end{array}$$

Trick

To find the negative equivalent of a number, work right-to-left and write down all digits through when you reach a 1. Then, invert the rest of the digits.

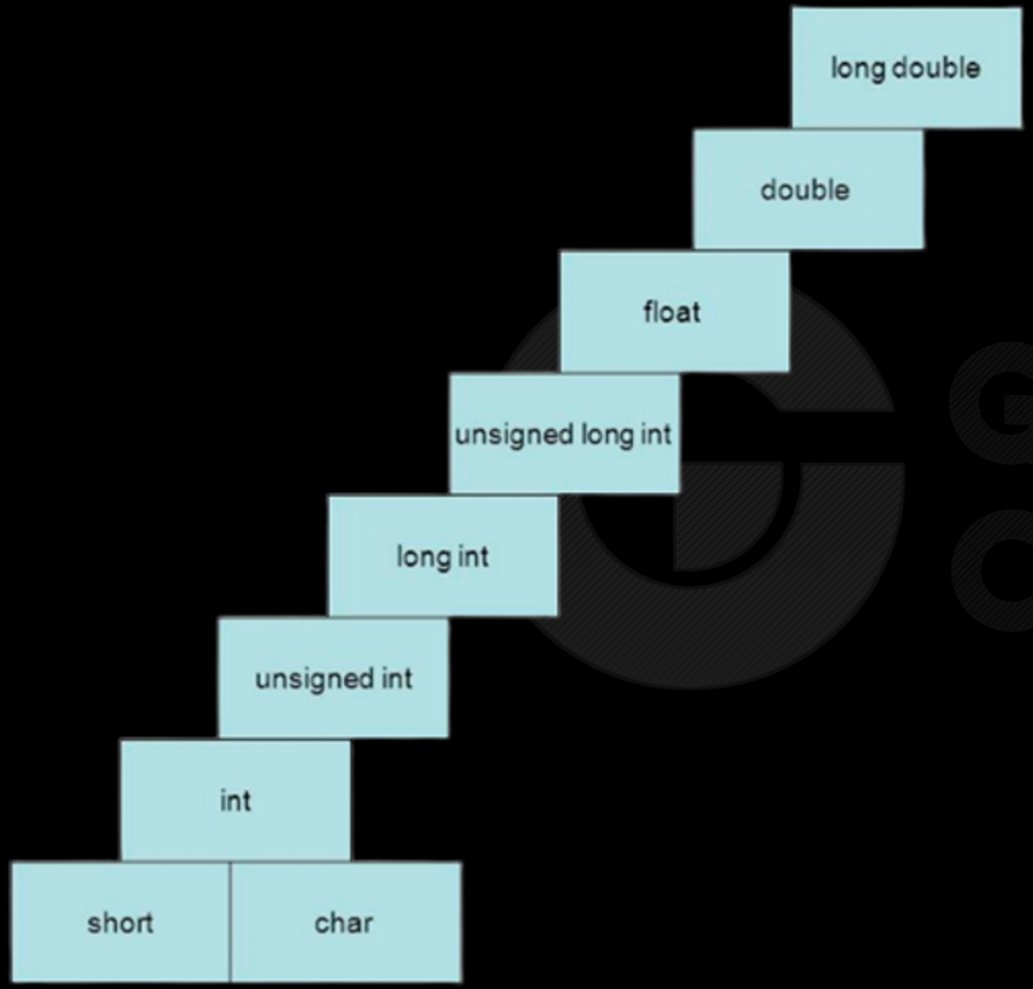
$$\begin{array}{r} 100100 \\ +\,011100 \\ \hline 000000 \end{array}$$


0101100
2's comp.
1010100


0
1
2
3
4
5
6
7
-8
-7
-6
-5
-4
-3
-2
-1
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
4-bit
two's complement
signed integer
representation

Type Conversion

Type Conversion


long double
double
float
unsigned long int
long int
unsigned int
int
short
char

Question 1

```
int value1 = 10, value2 = 3;
float result;

result = value1/value2;

printf("%f", result);
```

10/3 = 3

→ 3

Question 2

```
int a = 10;

float b = 3;

float result;


result = a/b;

printf("%f", result);
```

Question 3

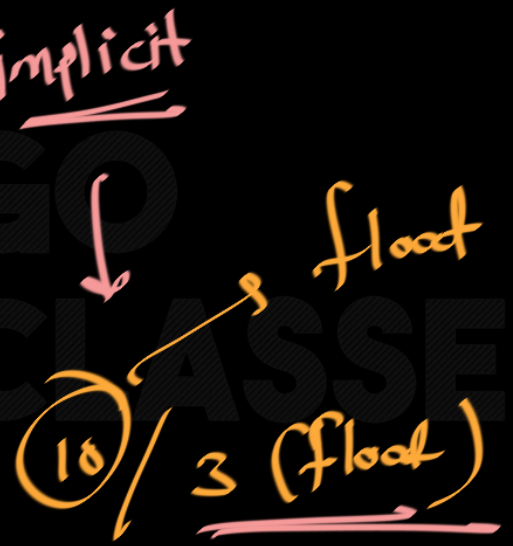
```
int x = 10;

int y = 3;

float result;


result = ( float )x/y;

printf("%f", result);
```

(float) 10 / 3 → float division

↳ 3.333

Question 4

```
int x = 10;

int y = 3;

float result;


result = x/ (float)y;

printf("%f", result);
```



3.333

Question 5: Tough (almost same GATE PYQ)

What will be the output

```
int main(){

    unsigned int a = 1000;
    int b = -1;

    if (a>b) printf("a is BIG");
    else printf("a is SMALL");

    return 0;
}
```

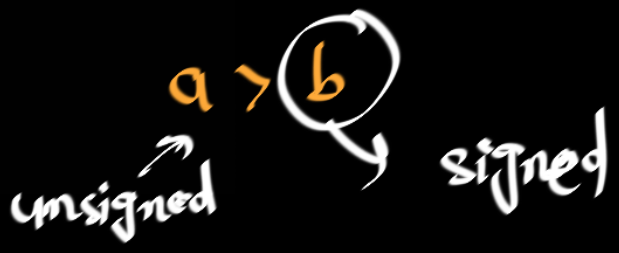
If you don't know if-else statement then don't worry about it. We will see next

GO CLASSES

```c
#include<stdio.h>

int main(){


    unsigned int a = 1000;
    int b = -1;

    if (a>b) printf("a is BIG");
    else printf("a is SMALL");
```

```
(base) sachinmittal@Sachins-MacBook-Pro Desktop % gcc a.c
(base) sachinmittal@Sachins-MacBook-Pro Desktop % ./a.out
a is SMALL%
(base) sachinmittal@Sachins-MacBook-Pro Desktop %
```


```c
int main(){


    unsigned int a = 1000;
    int b = -1;

    if (a>b) printf("a is BIG");
    else printf("a is SMALL");
```



Conditional Statements

- The if statement
- The switch statement