



C Programming



Now Let's come back to C programming



C Programming

↙ 32 bits
`int y = -9;`

y 

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

$y = 00\ldots 001001$
 $-y = \underline{111\ldots 11} \underline{1011}$

32 bits

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



```
printf("%d", y); => -9
```

```
printf("%u", y); => huge No
```



C Programming

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

y

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

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



GO
CLASSES



printf doesn't use the information
about type of variable



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

y | 111 - - - | 1 1011

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

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

CLASSES

huge no.
====



sign
zero

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



16 bits

```
short int x = 9;
```

```
int ix = x;
```

```
short x = 9;
```

Extension

x

ix

```
short signed int x = 9
```

16 bits

`short int x = 9;`

32 bits

`int ix = x;`

Extension

x

ix

16 bits

0	-	-	0	1001
---	---	---	---	------

0	0	0	0	0	0	0	0	0	-	-	0	1001
---	---	---	---	---	---	---	---	---	---	---	---	------

16 bits

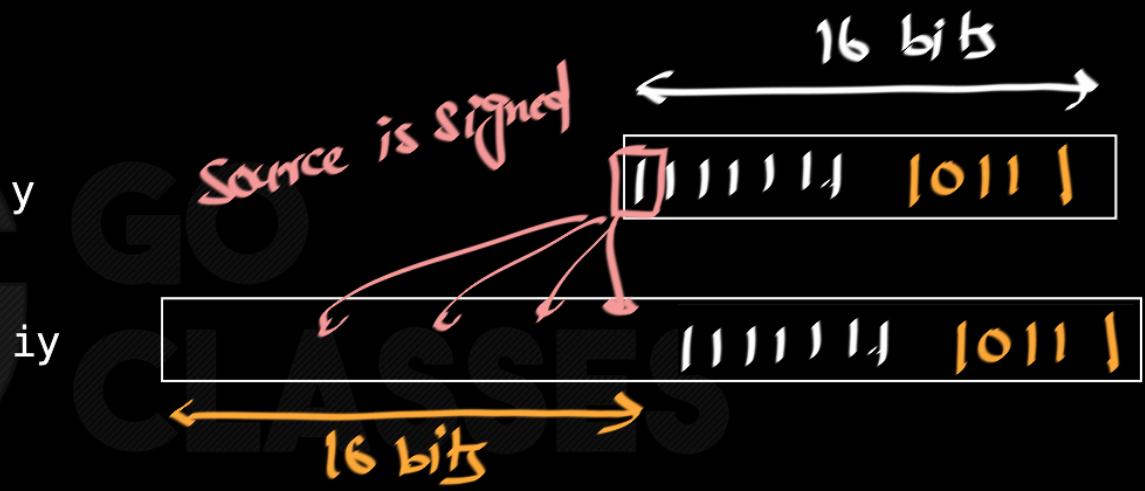




```
short int y = -9;
```

```
int iy = y;
```

Extension





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

```
int iy = y;
```

Extension

y

iy





C Programming

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

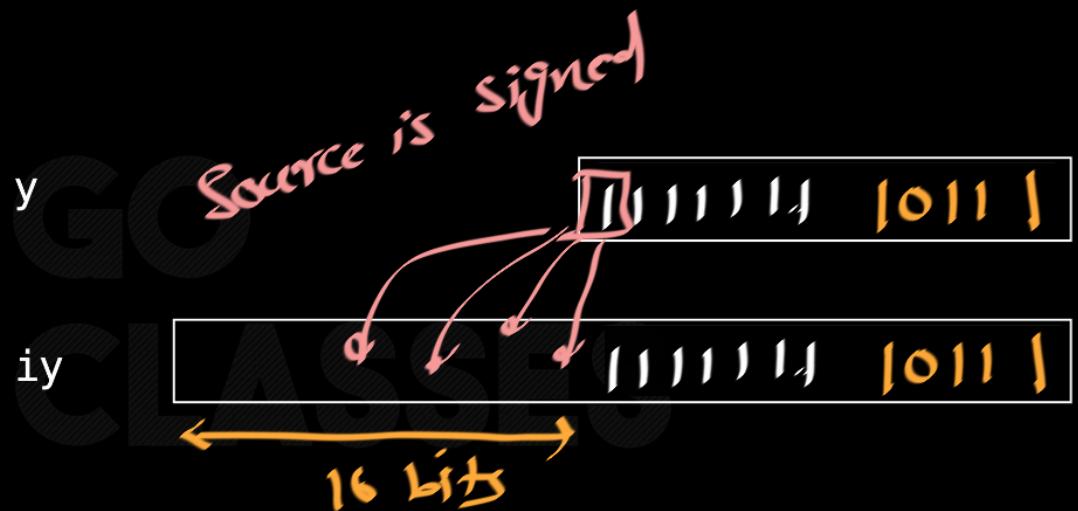
	Decimal	Binary
x	9	00000000 00001001
ix	9	00000000 00000000 00000000 00001001
y	-9	11111111 11110111
iy	-9	11111111 11111111 11111111 11110111



C Programming

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

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





C Programming

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

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



y

iy





C Programming

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

```
int iy = y;
```

y

iy

111111	10111
--------	-------

111111	10111
--------	-------





C Programming

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

y

```
int iy = y;
```

iy

Since source
is unsigned.

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

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

↔
16bits



C Programming

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

y

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

iy

Since space is
unsigned

111111	10111
--------	-------

0	00000	111111	10111
---	-------	--------	-------

← →
16 bits



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>



C Programming

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

y



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

iy



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



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



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



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





C Programming

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

y

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

```
int iy = y;
```

iy

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

16 bits

printf("%d", y); *huge*

printf("%u", y); *huge*

printf("%d", iy); *huge*

printf("%u", iy); *huge*

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



C Programming

```
int x = -1;
```

```
unsigned int u = x;
```



```
printf("%d", x);           → -1  
printf("%u", x);           → Huge  
printf("%d", u);           → -1  
printf("%u", u);           → Huge
```

GO CLASSES

Optional

2.2.6 Expanding the Bit Representation of a Number

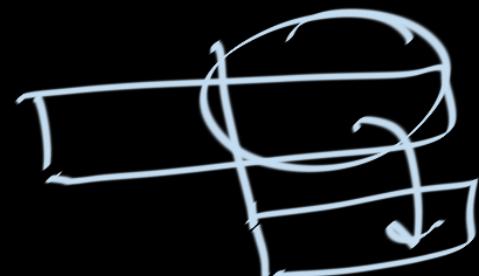
One common operation is to convert between integers having different word sizes while retaining the same numeric value. Of course, this may not be possible when the destination data type is too small to represent the desired value. Converting from a smaller to a larger data type, however, should always 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 $[x_{w-1}, x_{w-2}, \dots, x_0]$, the expanded representation is $[x_{w-1}, \dots, x_{w-1}, x_{w-1}, x_{w-2}, \dots, x_0]$. (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



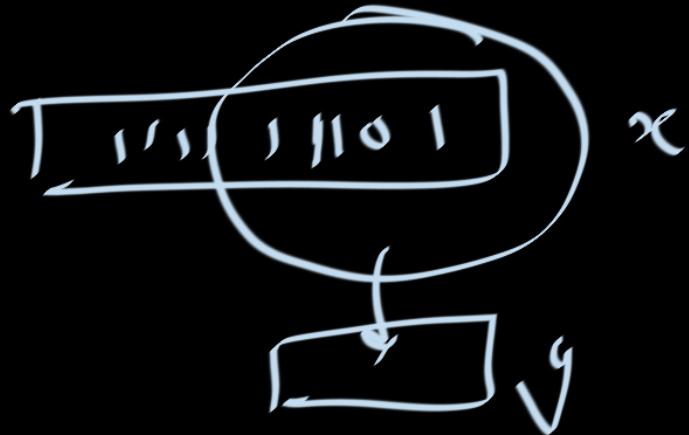
Stanford University



int x = -9

short y = x;

GO
CLASSES





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

int x = -9;



The integer promotions

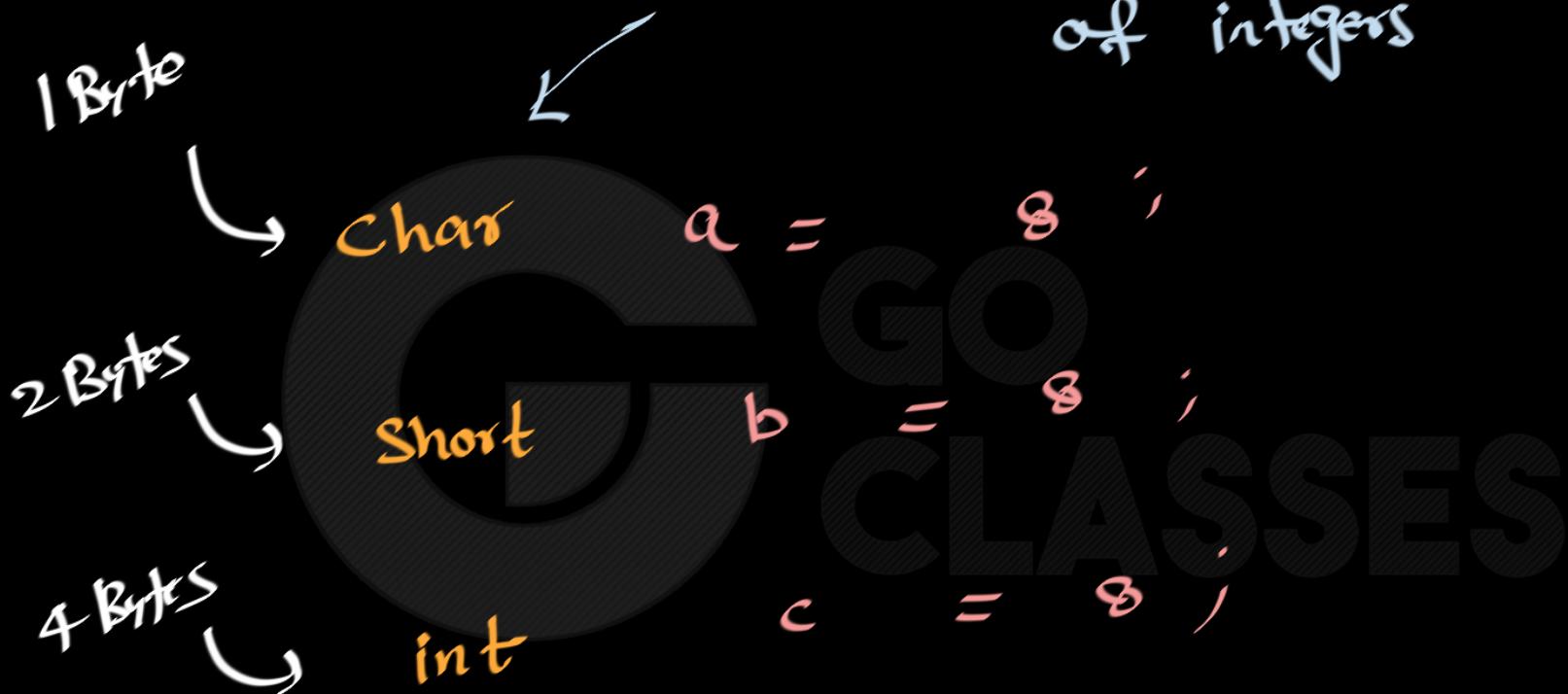
Whenever a small integer type (char or short) is used in an **expression**, it is implicitly converted to int .

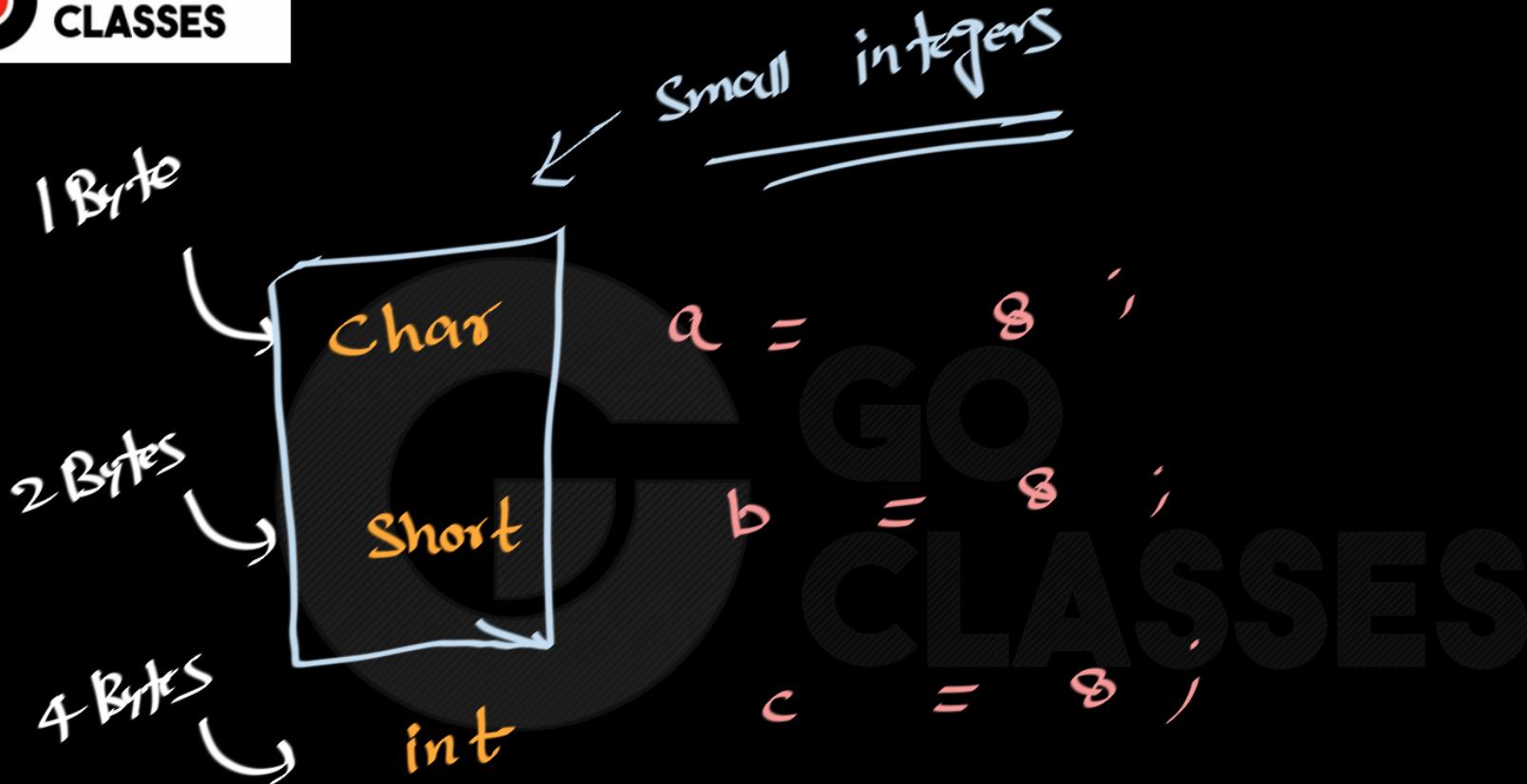
char
short

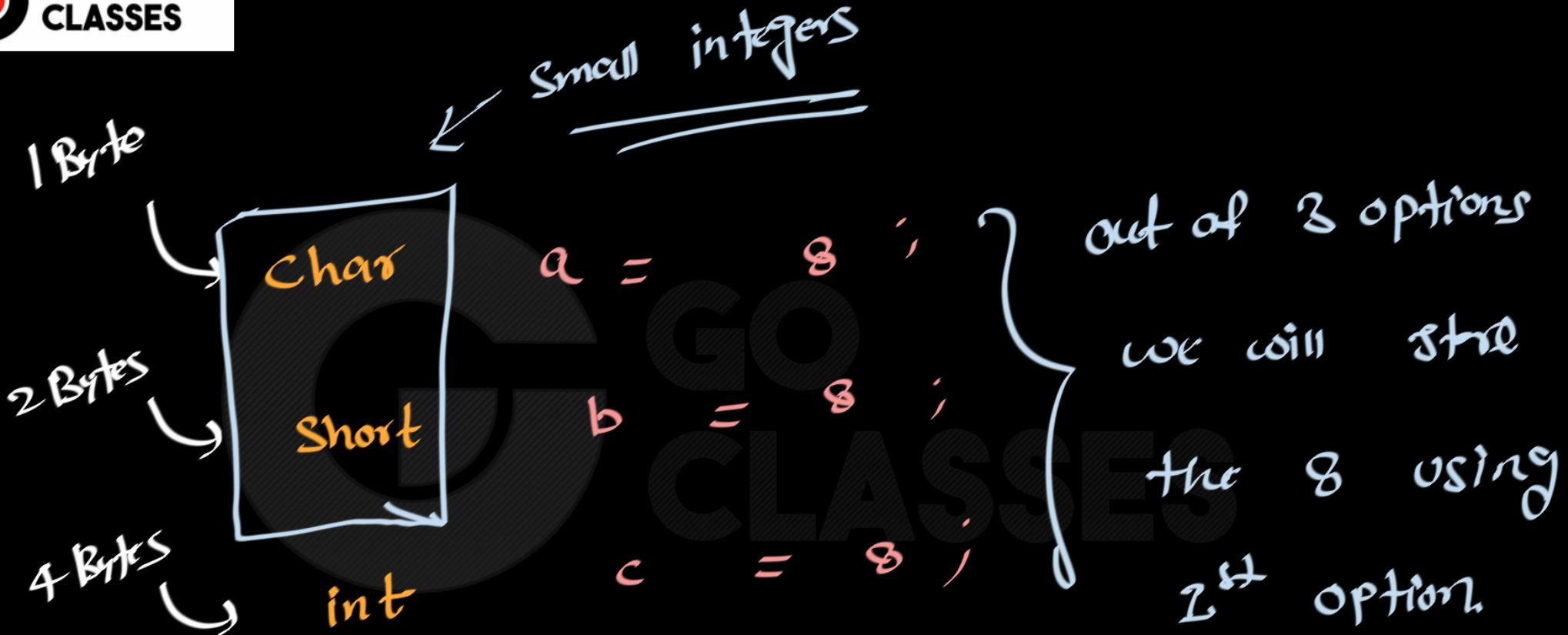
c = 90 ;

c + 0 c + 1 c - 1
 \leq

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







char c = 'a';

printf("%c", c)

output ↴ a

c memory ↴ 97

97 ↪

printf ("%d", a) ↪ output 97



97

char c = 'a';

ctrl = 97 + 98
= 97 + 98

c²-1 = 97²-1

signed char c = 90 ;

printf (" %d ", c)

01011010
c
← 24 bits →

The diagram illustrates the memory representation of the variable 'c'. A box labeled 'c' contains the binary value '01011010'. An arrow points from the 'c' in the printf statement to this box. Below the box, the text '← 24 bits →' indicates that the 8-bit value is treated as a 24-bit quantity. The background features a large, semi-transparent watermark of the 'GO CLASSES' logo.

signed char c = 90 ;

printf (" %d ", c)

01011010
c

90

signed char $x = 130;$

printf

(“%d”, x)


 x



signed char $x = 130;$

printf (" %d", x)

[1 0 0 0 0 0 1 0]
 x

-126

signed char $x = \text{130};$


 x

printf (" %u ", x)

→ a huge no.



unsigned char ux = 130 ;

1 0 0 0 0 0 1 0
x

printf (" %u ", ux)

$\xrightarrow{130}$

$\xleftarrow[24\text{ bits}]{000000000000000000000000}$



```
C a
Selection
1 #include<stdio.h>
2
3 int main(){
4
5
6     unsigned char x = 130;
7     printf("%d\n",x);
8
9     printf("%u",x);
0
1
```

Desktop — zsh — 80x24

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

`Signed char a = 258;`

`258`

`1_0_0_0_0_0_0_1_0`

`printf ("%d", a)`

`0_0_0_0_0_0_0_1_0`

`printf ("%u", a)`

`a`

`printf ("%c", a)`

$\xleftarrow[24 \text{ bits}]{00000000}$ `0_0_0_0_0_0_0_1_0`

Signed char a = 258 + 128;

258

1 1 00 0 0 0 1 0

printf ("%d", a) → 256

printf ("%u", a) → hugonot

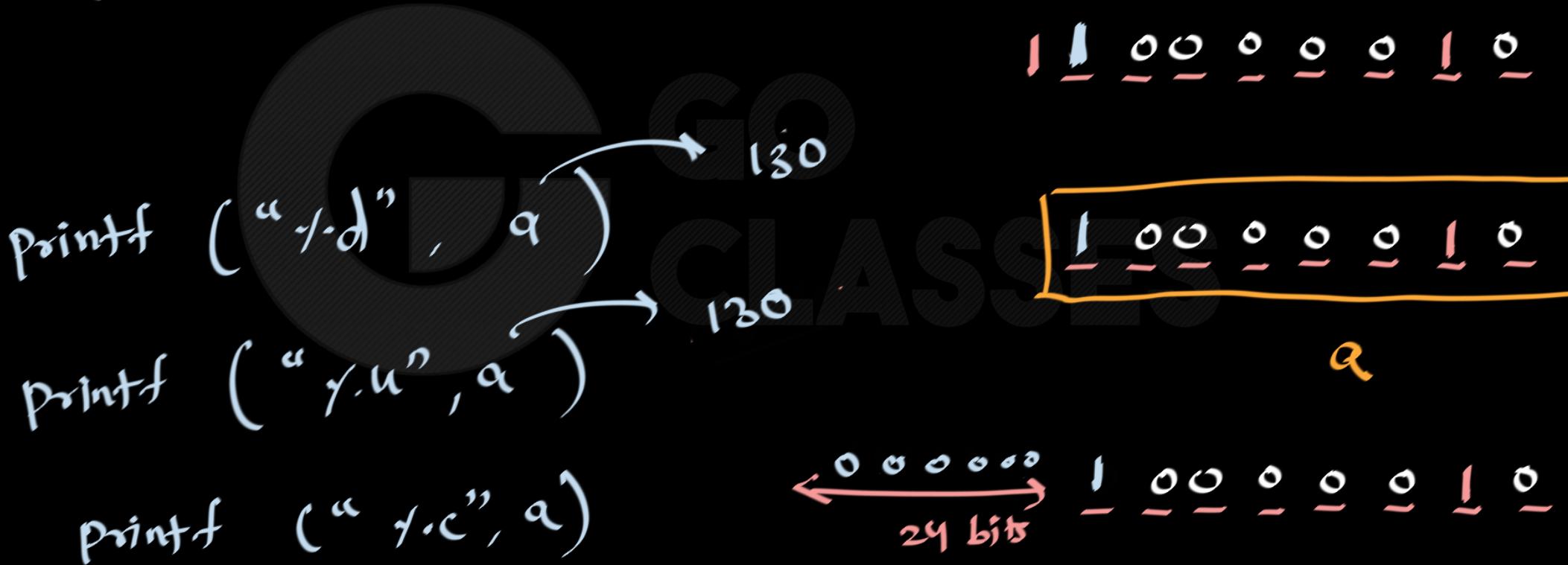
printf ("%c", a)

1 00 0 0 0 1 0

a

← 111111 → 1 00 0 0 0 1 0
24 bits

unsigned char a = 258 + 128;





C Programming

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

printf ("%d ", d); → -80

printf ("%d ", a * b); ↓ 1200

1200 in binary is
0000 0100 1011 0000

1011 0000

→ -120





```
#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);
```

A screenshot of a terminal window titled "Desktop -- zsh -- 80x24". The window shows the command line interface with three colored window control buttons (red, yellow, green) at the top. The terminal output is as follows:
[base] sachinmittal@Sachins-MacBook-Pro Desktop %
[base] sachinmittal@Sachins-MacBook-Pro Desktop %
-80
1200
[base] sachinmittal@Sachins-MacBook-Pro Desktop % |



C Programming

```
signed char f = -65;
```

```
printf("%d", f); → -65
```

```
printf("%u", f); → huge no.
```



content 10111111



The image shows a screenshot of a terminal window on a Mac OS X desktop. The terminal window has a dark grey background and a dark grey title bar. The title bar displays the path 'Desktop -- zsh -- 80x24'. The main area of the terminal shows the following command-line session:

```
(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 %
```

To the left of the terminal window, there is a code editor window titled 'a.c'. The code editor shows the following C program:

```
1 #include<stdio.h>
2
3 int main(){
4
5     signed char f = -65;
6
7     printf("%d\n",f);
8     printf("%u\n",f);
9
10}
```



C Programming

```
unsigned char f = -65;
```

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

191

191

$$\begin{array}{r} 128 \\ + 63 \\ \hline 191 \end{array}$$



← 00000000 →
24 0's



```
C a
main()
1 #include<stdio.h>
2
3 int main(){
4
5     unsigned char f = -65;
6
7     printf("%d\n",f);
8     printf("%u\n",f);
9
10
```

```
Desktop -- zsh -- 80x24
(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 %
```



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>]

Summary

- 1) Numbers $\Rightarrow -3, 3$ have fixed repr.
- 2) printf doesn't make use of the information of data type
- 3) Extension \leftarrow (truncation ✓)
- 4) integer promotion.



Optional : 2'S Complement



C Programming

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

$$\begin{array}{r} \textcolor{red}{1} \textcolor{red}{1} \\ 0101 \\ + \textcolor{red}{1011} \\ \hline 0000 \end{array}$$



C Programming

Answer

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



C Programming

$$\begin{array}{r} 0011 \\ + \textcolor{cyan}{????} \\ \hline 0000 \end{array} \quad \rightarrow \quad \begin{array}{r} 0011 \\ + \textcolor{orange}{1101} \\ \hline 0000 \end{array}$$

The diagram illustrates two binary addition problems. On the left, a binary number 0011 is added to a binary number represented by four question marks (????). The result is 0000. On the right, the same binary number 0011 is added to a binary number represented by four digits (1101). The result is 0000. The addition is performed using standard binary arithmetic rules.



C Programming

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





C Programming

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



GO
CLASSES



C Programming

Answer

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



GO
CLASSES



There Seems Like a Pattern Here...


$$\begin{array}{r} 0101 \\ +1011 \\ \hline 0000 \end{array} \quad \begin{array}{r} 0011 \\ +1101 \\ \hline 0000 \end{array} \quad \begin{array}{r} 0000 \\ +0000 \\ \hline 0000 \end{array}$$



Why adding

with

2's Complement gives zeros ?



C Programming

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 ?

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

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

The first two rows of the addition problem are circled in orange. An arrow points from the circled '0101' to the result '1111'. The result '1111' is followed by the letters 'ES'.

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



C Programming

Why adding with 2's Complement gives zeros ?

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



$$\begin{array}{r} 0101 \\ + 1010 \\ \hline 0001 \end{array} \quad \begin{array}{r} \xrightarrow{\hspace{1cm}} \\ + \end{array} \quad \begin{array}{r} 1111 \\ + 0001 \\ \hline 0000 \end{array}$$



↓ One's Complement

A binary number plus its inverse is all 1s.

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

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

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

A large, semi-transparent watermark of the letter 'G' is centered on the slide. It has a dark gray background color and a subtle radial gradient, giving it a three-dimensional appearance. The 'G' is oriented vertically and overlaps the title text.

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.

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



$$\begin{array}{r} 100 \\ \times 100 \\ \hline 000000 \end{array}$$

GO CLASSES



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.

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



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

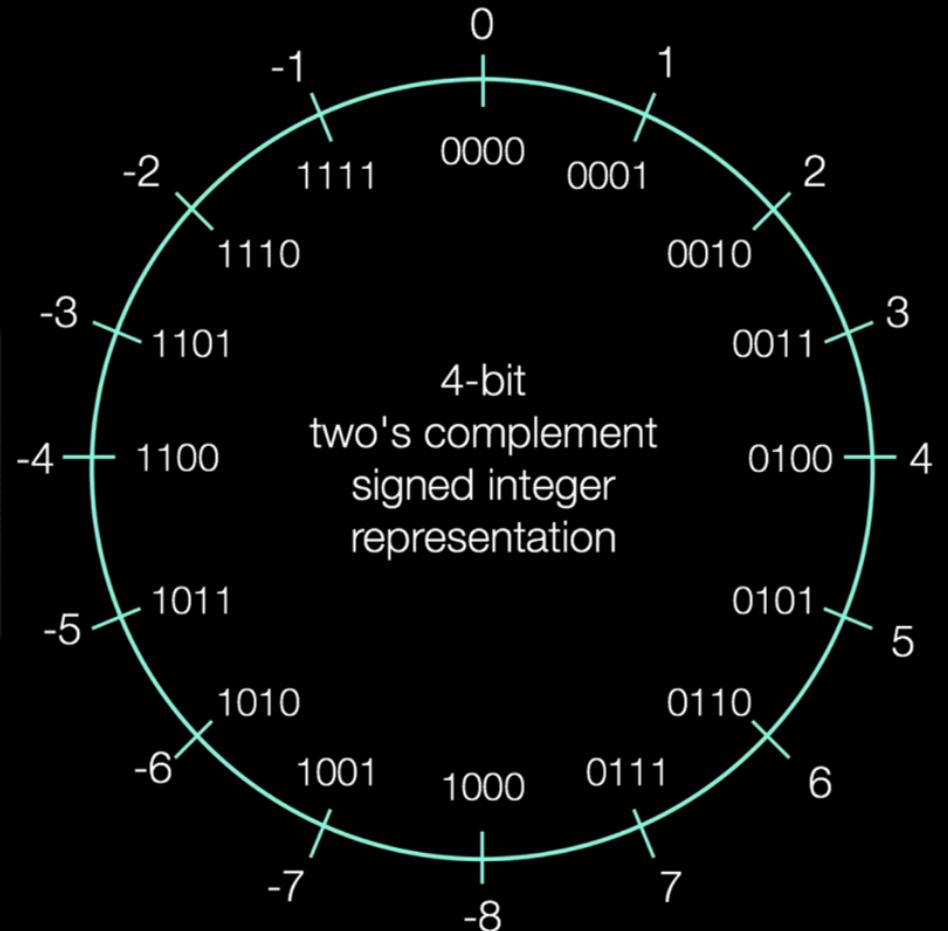
2's comp

|010100

GO
CLASSES



C Programming



S



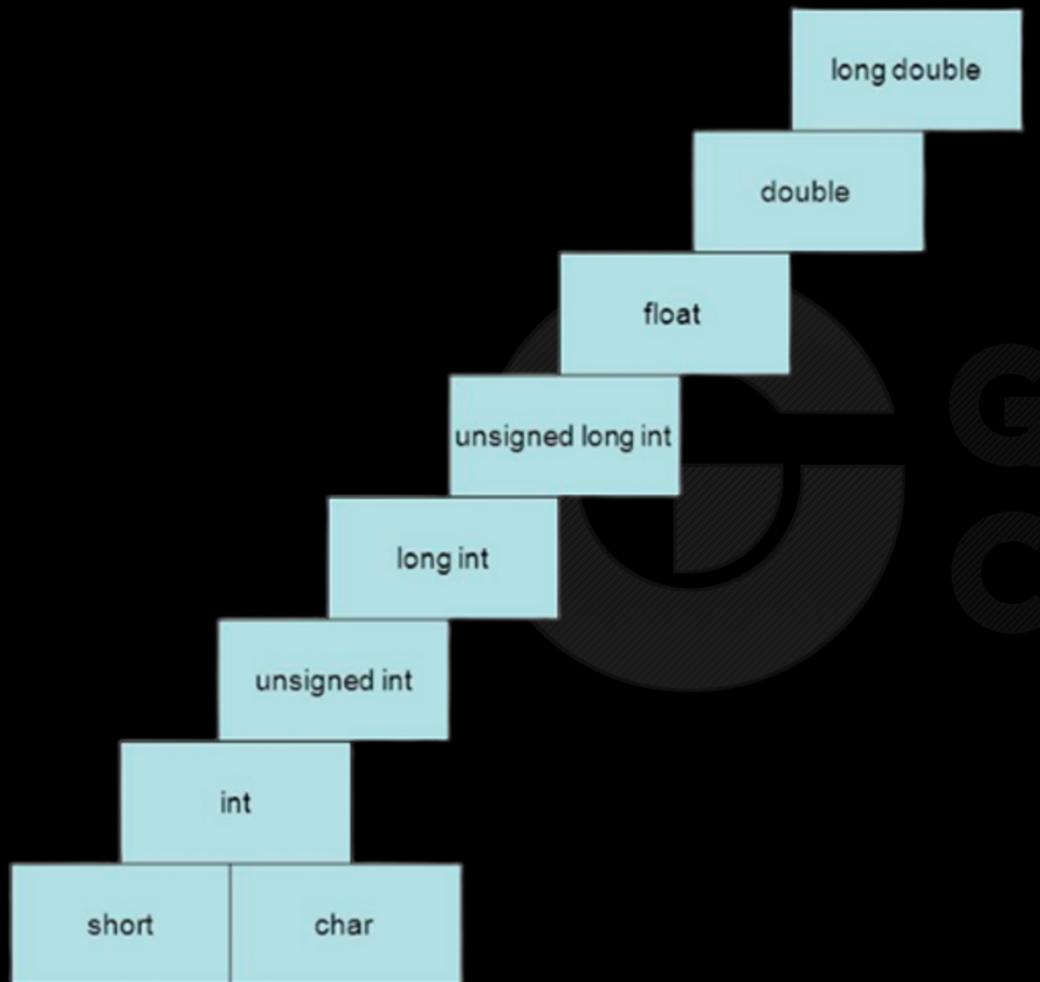
Type Conversion



Type Conversion



C Programming





Question 1

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

```
result = value1/value2;
```

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

→ 3



Question 2

```
int a = 10;  
  
float b = 3;  
  
float result;  
  
result = a/b;  
  
printf("%f", result);
```



float division

int / float
implicitly
float



Question 3

```
int x = 10;  
  
int y = 3;  
  
float result;  
  
result = ( float )x/y;  
  
printf("%f", result);
```



GO
CLASSES

(float) 10 / 3 float division

3.333



Question 4

```
int x = 10;  
int y = 3;  
float result;  
  
result = x/ (float)y;  
  
printf("%f", result);
```



implicit
↓
10 / 3 (float)
float

3.333



Question 5:

Tough *(almost same GATE PYQ)*

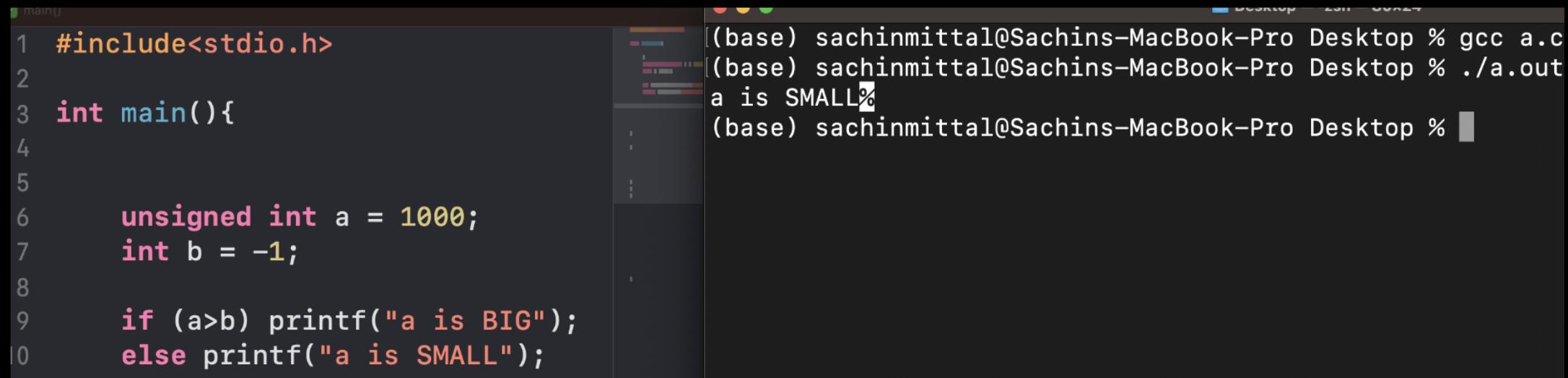
```
int main(){
    unsigned int a = 1000;
    int b = -1;

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

    return 0;
}
```

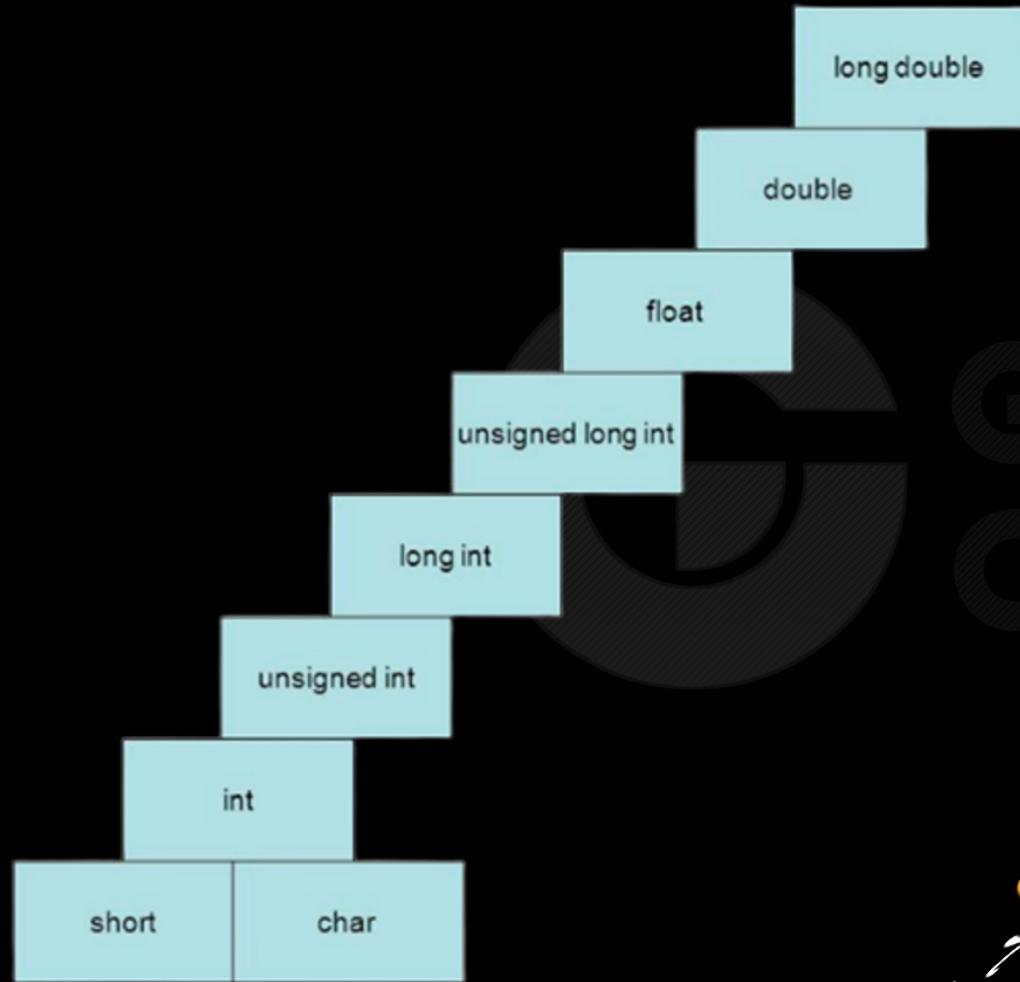
What will be the
output

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



```
main()
1 #include<stdio.h>
2
3 int main(){
4
5
6     unsigned int a = 1000;
7     int b = -1;
8
9     if (a>b) printf("a is BIG");
10    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 %



0000011111111111 ← -1 in Binary

```
int main(){  
    unsigned int a = 1000;  
    int b = -1;  
  
    if (a>b) printf("a is BIG");  
    else printf("a is SMALL");
```

a > b
unsigned → signed



Conditional Statements

- The if statement
- The switch statement