# **Basic Refreshers**

# Number System

## **Number Systems**

- A number is generally represented as
  - Decimal
  - Octal
  - Hexadecimal
  - Binary

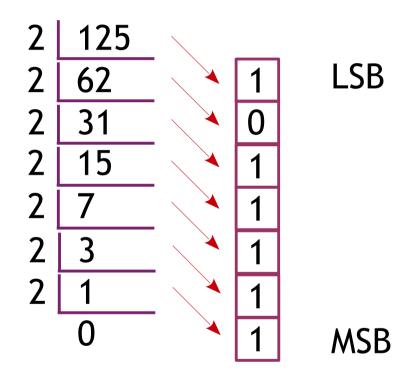
Туре	Range (8 Bits)
Decimal	0 - 255
Octal	<mark>0</mark> 00 - <mark>0</mark> 377
Hexadecimal	0x00 - 0xFF
Binary	<mark>0b</mark> 00000000 - <mark>0b</mark> 11111111

Туре	Dec	Oct	Hex		B	in	
Base	10	8	16		2	2	
	0	0	0	0	0	0	0
	1	1	1	0	0	0	1
	2	2	2	0	0	1	0
	3	3	3	0	0	1	1
	4	4	4	0	1	0	0
	5	5	5	0	1	0	1
	6	6	6	0	1	1	0
	7	7	7	0	1	1	1
	8	10	8	1	0	0	0
	9	11	9	1	0	0	1
	10	12	A	1	0	1	0
	11	13	В	1	0	1	1
	12	14	С	1	1	0	0
	13	15	D	1	1	0	1
	14	16	E	1	1	1	0
	15	17	F	1	1	1	1



Number Systems - Decimal to Binary

• 125<sub>10</sub> to Binary



• So 125<sub>10</sub> is 1111101<sub>2</sub>



Number Systems - Decimal to Octal

• 212<sub>10</sub> to Octal

• So 212<sub>10</sub> is 324<sub>8</sub>



## Number Systems - Decimal to Hexadecimal

• 472<sub>10</sub> to Hexadecimal

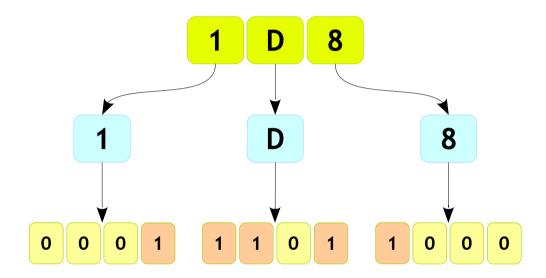
Representation	Su	bsti	tute	S												
Dec	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hex	0	1	2	3	4	5	6	7	8	9	A	В	C	D	E	F

So 472<sub>10</sub> is 1D8<sub>16</sub>



Number Systems - Hexadecimal to Binary

• 1D8<sub>16</sub> to Binary

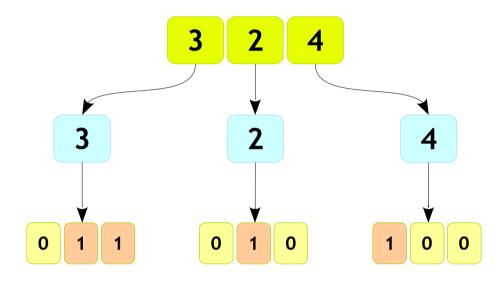


So 1D8<sub>16</sub> is 000111011000<sub>2</sub> which is nothing but 111011000<sub>2</sub>



Number Systems - Octal to Binary

• 324<sub>8</sub> to Binary



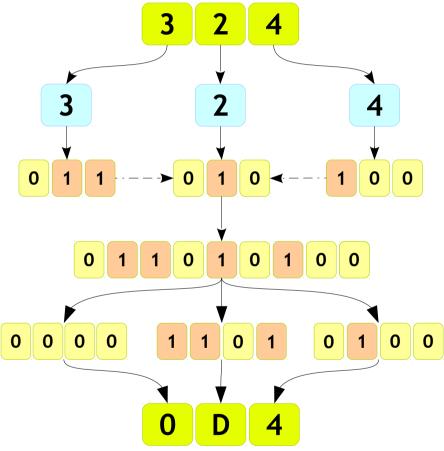
• So 324<sub>8</sub> is 011010100<sub>2</sub> which is nothing but 11010100<sub>2</sub>





Number Systems - Octal to Hexadecimal

• 324<sub>8</sub> to Hexadecimal

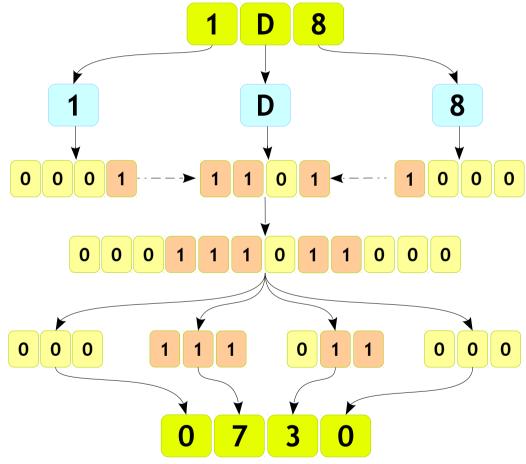


So 324<sub>8</sub> is 0D4<sub>16</sub> which is nothing but D4<sub>16</sub>



## Number Systems - Hexadecimal to Octal

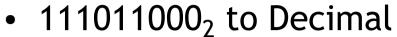
• 1D8<sub>16</sub> to Octal



So 1D8<sub>16</sub> is 0730<sub>8</sub> which is nothing but 730<sub>8</sub>



## Number Systems - Binary to Decimal



```
1 1 1 0 1 1 0 0 0 0 Bit Position 8 7 6 5 4 3 2 1 0 28 27 26 25 24 23 22 21 20 256 + 128 + 64 + 0 + 16 + 8 + 0 + 0 + 0 =
```

472

• So 111011000<sub>2</sub> is 472<sub>10</sub>



# Data Representations

Data Representation - Bit



- Literally computer understand only two states HIGH and LOW making it a binary system
- These states are coded as 1 or 0 called binary digits
- "Binary Digit" gave birth to the word "Bit"
- Bit is known a basic unit of information in computer and digital communication

Value	No of Bits
0	0
1	1



Data Representation - Byte

- A unit of digital information
- Commonly consist of 8 bits
- Considered smallest addressable unit of memory in computer

Value	No of Bits	
0	00000	000
1	00000	0 0 1



## Data Representation - Character



- One byte represents one unique character like 'A', 'b', '1', '\$' ...
- Its possible to have 256 different combinations of 0s and 1s to form a individual character
- There are different types of character code representation like
  - ASCII → American Standard Code for Information Interchange - 7 Bits (Extended - 8 Bits)
  - EBCDIC → Extended BCD Interchange Code 8 Bits
  - Unicode → Universal Code 16 Bits and more



Data Representation - Character



- ASCII is the oldest representation
- Please try the following on command prompt to know the available codes
  - \$ man ascii
- Can be represented by char datatype

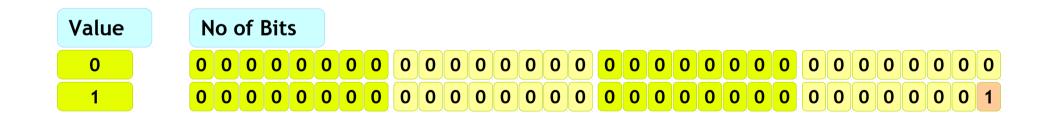
Value	N	0	of I	Bit	S			
0	0	0	1	1	0	0	0	0
A	0	1	0	0	0	0	0	1



Data Representation - word



- Amount of data that a machine can fetch and process at one time
- An integer number of bytes, for example, one, two, four, or eight
- General discussion on the bitness of the system is references to the word size of a system, i.e., a 32 bit chip has a 32 bit (4 Bytes) word size





## Integer Number - Positive



- Integers are like whole numbers, but allow negative numbers and no fraction
- An example of 13<sub>10</sub> in 32 bit system would be

Bit	No of Bits	
Position	31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1	0
Value	000000000000000000000000000000000000000	1



## Integer Number - Negative



- Negative Integers represented with the 2's complement of the positive number
- An example of -13<sub>10</sub> in 32 bit system would be

Bit	N	lo (	of	Bit	S																											
Position	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1
1's Compli	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	0
Add 1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
																<u> </u>																

• Mathematically:  $-k \equiv 2^n - k$ 



#### Float Point Number



- Computers are integer machines and are capable of representing real numbers only by using complex codes
- The most popular code for representing real numbers is called the IEEE Floating-Point Standard

	Sign	Exponent	Mantissa
Float (32 bits) Single Precision	1 bit	8 bits	23 bits
Double (64 bits) Double Precision	1 bit	11 bits	52 bits



#### Float Point Number - Conversion Procedure



- STEP 1: Convert the absolute value of the number to binary, perhaps with a fractional part after the binary point. This can be done by -
  - Converting the integral part into binary format.
  - Converting the fractional part into binary format.

The integral part is converted with the techniques examined previously.

The fractional part can be converted by multiplying it with 2.

• STEP 2: Normalize the number. Move the binary point so that it is one bit from the left. Adjust the exponent of two so that the value does not change.

Float : 
$$V = (-1)^s * 2^{(E-127)} * 1.F$$

Double : 
$$V = (-1)^s * 2^{(E-1023)} * 1.F$$



## Float Point Number - Conversion - Example 1



Convert 2.5 to IEEE 32-bit floating point format

Step 1:

Step 2:

× 2 1.0

1.0

Normalize:  $10.1_2 = 1.01_2 \times 2^1$ 

 $2.5_{10} = 10.1_{2}$ 

Sign bit is 0

Bit	S			Ex	крс	ne	ent	•												Ma	nti	SSã	à									
Position	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



## Float Point Number - Conversion - Example 2



#### Convert 0.625 to IEEE 32-bit floating point format

#### Step 1:

0.625	× 2	1.25	1
0.25	× 2	0.5	0
0.5	× 2	1.0	1

$$0.625_{10} = 0.101_2$$

#### Step 2:

Normalize:  $0.101_2 = 1.01_2 \times 2^{-1}$ 

Bit	S			Ex	крс	nε	ent	1												Ma	nti	SSā	ì									
Position	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value	0	0	1	1	1	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



## Float Point Number - Conversion - Example 3



#### Convert 39887.5625 to IEEE 32-bit floating point format

#### Step 1:

0.5625	× 2 =	<b>1</b> .125	1
0.125	× 2 =	0.25	0
0.25	× 2 =	0.5	0
0.5	× 2 =	1.0	1

39887.5625<sub>10</sub> = 10011011111001<sub>2</sub>

#### Step 2:

Normalize: 10011011111001111.1001<sub>2</sub> = 1.00110111110011111001<sub>2</sub> × 2<sup>15</sup>

Mantissa is 001101111100111110010000 Exponent is 15 + 127 = 142 = 10001110<sub>2</sub> Sign bit is 0

Bit		S		Exponent																I	Ma	nti	SSā	ì									
Posi	tion	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Valu	e	0	1	0	0	0	1	1	1	0	0	0	1	1	0	1	1	1	1	0	0	1	1	1	1	1	0	0	1	0	0	0	0



### Float Point Number - Conversion - Example 5



#### Convert -13.3125 to IEEE 32-bit floating point format

#### Step 1:

0.3125	× 2 =	0.625	0
0.625	× 2 =	1.25	1
0.25	× 2 =	0.5	0
0.5	× 2 =	1.0	1

$$13.3125_{10} = 1101.0101_{2}$$

#### Step 2:

Normalize:

$$1101.0101_2 = 1.1010101_2 \times 2^3$$

Bit	S			E	крс	one	ent	1												Ma	nti	SSã	ì									
Position	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value	1	1	0	0	0	0	0	1	0	1	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0



## Float Point Number - Conversion - Example 6



#### Step 1:

0.7	× 2 =	1.4	1
0.4	× 2 =	8.0	0
0.8	× 2 =	1.6	1
0.6	× 2 =	1.2	1
0.2	x 2 =	0.4	0
0.4	× 2 =	0.8	0
0.8	× 2 =	1.6	1
0.6	× 2 =	<b>1.2</b>	1

#### Step 2:

#### Normalize:

1.1011001100110011001<sub>2</sub> =

 $1.1011001100110011001_2 \times 2^0$ 

Mantissa is 1011001100110011001 Exponent is 0 + 127 = 127 = 011111111<sub>2</sub> Sign bit is 1

 $1.7_{10} = 1.1011001100110011001_2$ 

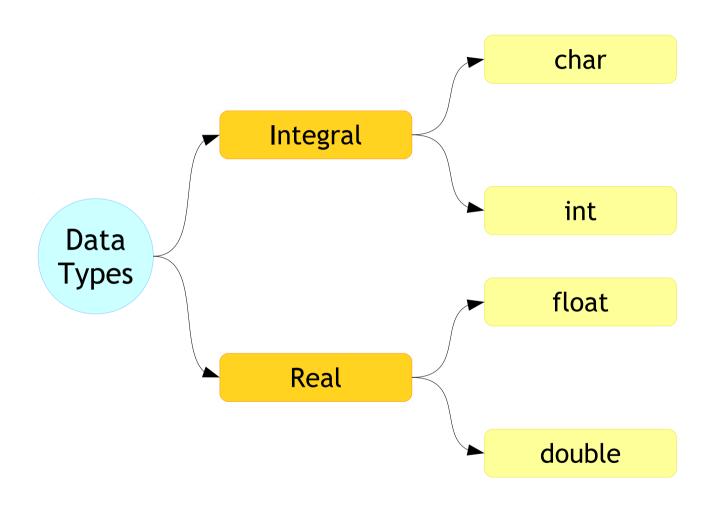
Bit	S			Ex	кро	on€	ent													Ma	nti	SS	à									
Position	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value	0	0	1	1	1	1	1	1	1	1	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1



# Data Types

# Advanced C Data Types - Categories







# Advanced C Data Types - Usage

#### **Syntax**

```
data_type name_of_the_variable;
```

#### Example

```
char option;
int age;
float height;
```

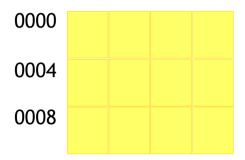


## Data Types - Storage

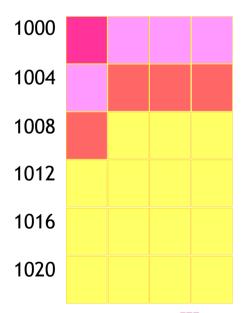
#### Example

```
char option;
int age;
float height;
```











## Data Types - Printing

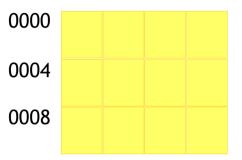
#### 001\_example.c

```
#include <stdio.h>
int main()
{
    char option;
    int age;
    float height;

    printf("The character is %c\n", option);
    printf("The integer is %d\n", age);
    printf("The float is %f\n", height);

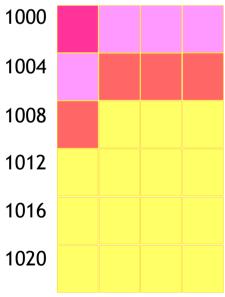
    return 0;
}
```





•

•





## Data Types - Scaning

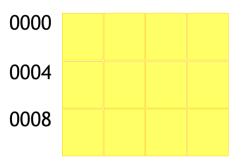
#### 002\_example.c

```
#include <stdio.h>
int main()
{
    char option;
    int age;
    float height;

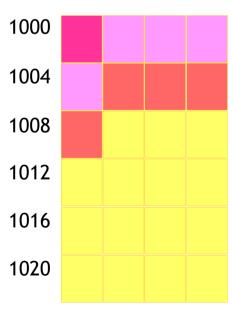
    scanf("%c", &option);
    printf("The character is %c\n", option);
    scanf("%d", &age);
    printf("The integer is %d\n", age);
    scanf("%f", &height);
    printf("The float is %f\n", height);

    return 0;
}
```





•





## Data Types - Size

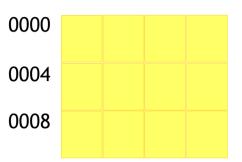
#### 003\_example.c

```
#include <stdio.h>
int main()
{
    char option;
    int age;
    float height;

    printf("The size of char is %u\n", sizeof(char));
    printf("The size of int is %u\n", sizeof(int));
    printf("The float is %u\n", sizeof(float));

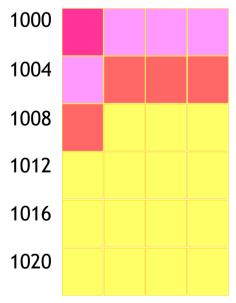
    return 0;
}
```





•

•





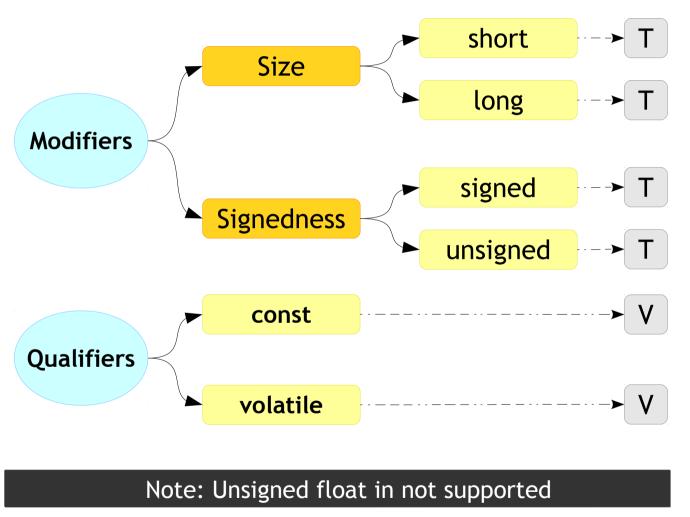
## Data Types - Modifiers and Qualifiers



- These are some keywords which is used to tune the property of the data type like
  - Its width
  - Its sign
  - Its storage location in memory
  - Its access property
- The K & R C book mentions only two types of qualifiers (refer the next slide). The rest are sometimes interchangably called as specifers and modifiers and some people even call all as qualifiers!



## Data Types - Modifiers and Qualifiers





Variables

T Data Types

**F** Functions







#### **Syntax**

```
<modifier / qualifier> <data_type> name_of_the_variable;
```

#### Example

```
short int count1;
long int count2;
const int flag;
```



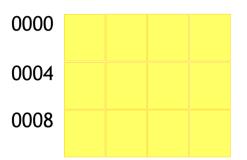
#### Data Types - Modifiers - Size

```
#include <stdio.h>
int main()
{
    short int count1;
    int long count2;
    short count3;

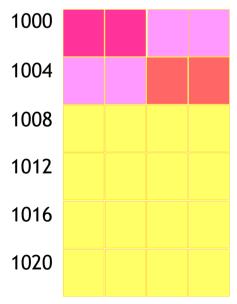
    printf("short is %u bytes\n", sizeof(short int));
    printf("long int is %u bytes\n", sizeof(int long));
    printf("short is %u bytes\n", sizeof(short));

    return 0;
}
```









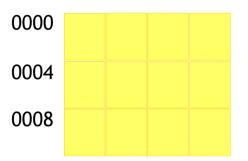


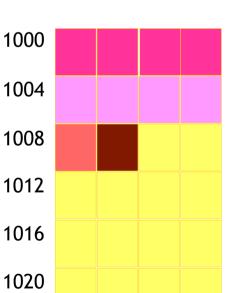
#### Data Types - Modifiers - Sign

```
#include <stdio.h>
int main()
{
    unsigned int count1;
    signed int count2;
    unsigned char count3;
    signed char count4;

    printf("count1 is %u bytes\n", sizeof(unsigned int));
    printf("count2 is %u bytes\n", sizeof(signed int));
    printf("count3 is %u bytes\n", sizeof(unsigned char));
    printf("count3 is %u bytes\n", sizeof(signed char));
    return 0;
}
```









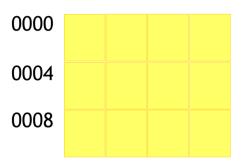
#### Data Types - Modifiers - Sign and Size

```
#include <stdio.h>
int main()
{
    unsigned short count1;
    signed long count2;
    short signed count3;

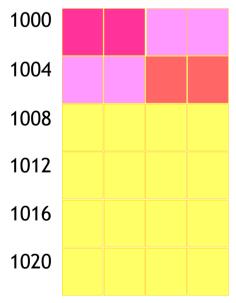
    printf("count1 is %u bytes\n", sizeof(count1));
    printf("count2 is %u bytes\n", sizeof(count2));
    printf("count3 is %u bytes\n", sizeof(count3));

    return 0;
}
```











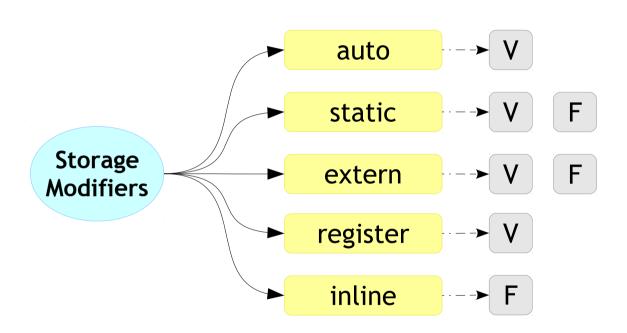
#### Data Types - Modifiers - Sign and Size

```
#include <stdio.h>
int main()
    unsigned int count1 = 10;
    signed int count2 = -1;
    if (count1 > count2)
        printf("Yes\n");
    else
        printf("No\n");
    return 0;
```



### Data Types and Function - Storage Modifiers





Variables

T Data Types

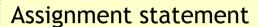
**F** Functions



# Statements

#### Code Statements - Simple

```
int main()
   number = 5;
   3; +5;
   sum = number
   4 + 5;
```



Valid statement, But smart compilers might remove it

Assignment statement. Result of the number + 5 will be assigned to sum

Valid statement, But smart compilers might remove it

This valid too!!



#### Code Statements - Compound

```
int main()
   if (num1 > num2)
      if (num1 > num3)
         printf("Hello");
      else
         printf("World");
```



If conditional statement

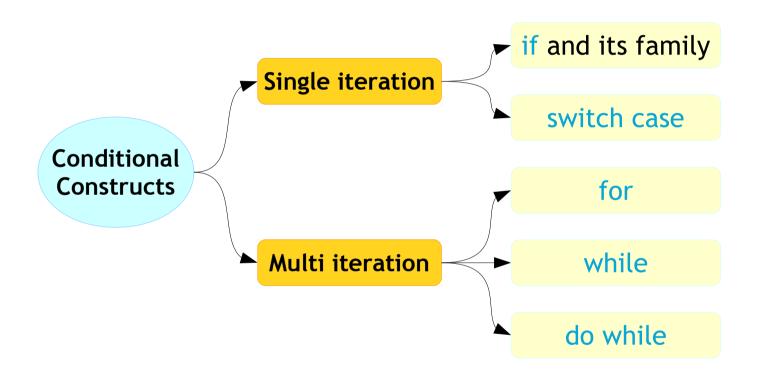
Nested if statement



## **Conditional Constructs**

## Advanced C Conditional Constructs





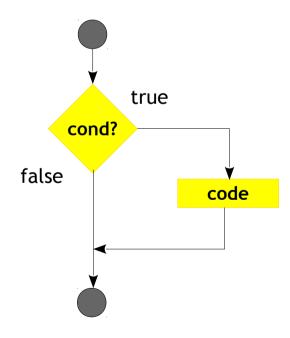


#### Conditional Constructs - if

#### **Syntax**

```
if (condition)
{
    statement(s);
}
```

#### Flow



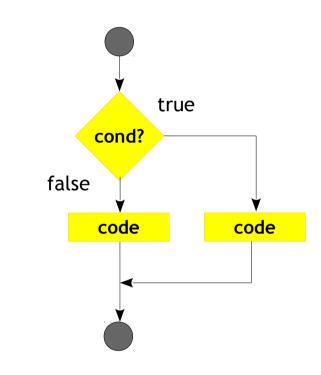
```
#include <stdio.h>
int main()
{
    int num = 2;
    if (num < 5)
    {
        printf("num < 5\n");
    }
    printf("num is %d\n", num);
    return 0;
}</pre>
```



#### Conditional Constructs - if else

#### **Syntax**

```
if (condition)
{
    statement(s);
}
else
{
    statement(s);
}
```





#### Conditional Constructs - if else

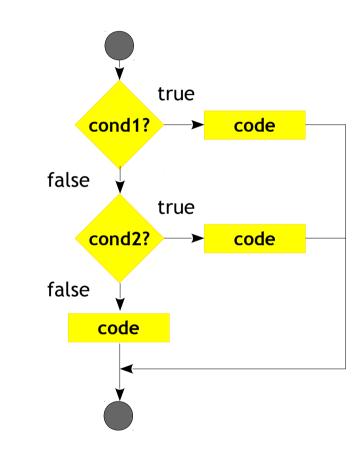
```
#include <stdio.h>
int main()
   int num = 10;
   if (num < 5)
       printf("num is smaller than 5\n'');
   else
       printf("num is greater than 5\n");
   return 0;
```



#### Conditional Constructs - if else if

#### **Syntax**

```
if (condition1)
{
    statement(s);
}
else if (condition2)
{
    statement(s);
}
else
{
    statement(s);
}
```





#### Conditional Constructs - if else if

```
#include <stdio.h>
int main()
   int num = 10;
   if (num < 5)
       printf("num is smaller than 5\n'');
   else if (num > 5)
       printf("num is greater than 5\n'');
   else
       printf("num is equal to 5\n'');
   return 0;
```



#### Conditional Constructs - Exercise



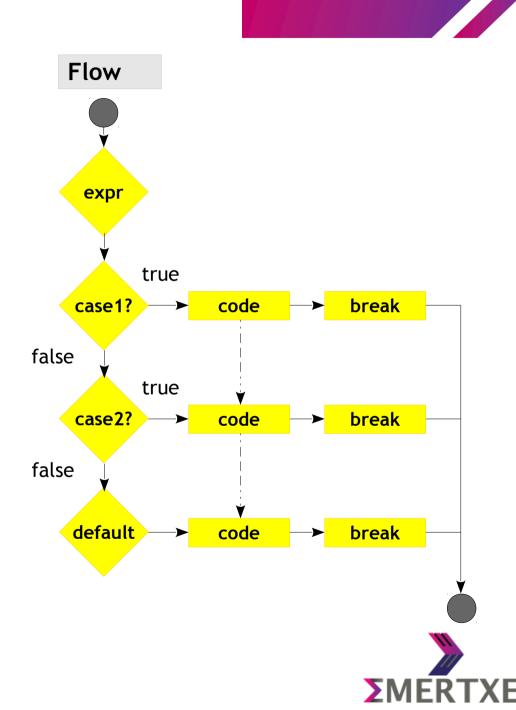
- WAP to find the max of two numbers
- WAP to print the grade for a given percentage
- WAP to find the greatest of given 3 numbers
- WAP to check whether character is
  - Upper case
  - Lower case
  - Digit
  - No of the above
- WAP to find the middle number (by value) of given 3 numbers



#### Conditional Constructs - switch

#### **Syntax**

```
switch (expression)
   case constant:
       statement(s);
      break;
   case constant:
       statement(s);
      break;
   case constant:
       statement(s);
      break;
   default:
       statement(s);
```



#### Conditional Constructs - switch

```
#include <stdio.h>
int main()
   int option;
   printf("Enter the value\n");
   scanf("%d", &option);
   switch (option)
    {
       case 10:
           printf("You entered 10\n");
           break;
       case 20:
           printf("You entered 20\n");
           break:
       default:
           printf("Try again\n");
   return 0;
```



#### Conditional Constructs - switch - DIY

- W.A.P to check whether character is
  - Upper case
  - Lower case
  - Digit
  - None of the above
- W.A.P for simple calculator



#### Conditional Constructs - while

#### **Syntax**

```
while (condition)
{
    statement(s);
}
```

- Controls the loop.
- Evaluated before each execution of loop body

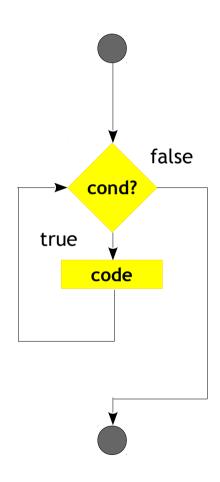
#### 011\_example.c

```
#include <stdio.h>
int main()
{
   int iter;

   iter = 0;
   while (iter < 5)
   {
      printf("Looped %d times\n", iter);
      iter++;
   }

   return 0;
}</pre>
```







#### Conditional Constructs - do while

#### **Syntax**

```
do
{
    statement(s);
} while (condition);
```

- Controls the loop.
- Evaluated after each execution of loop body

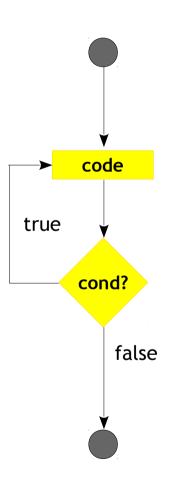
#### 012\_example.c

```
#include <stdio.h>
int main()
{
   int iter;

   iter = 0;
   do
   {
      printf("Looped %d times\n", iter);
      iter++;
   } while (iter < 10);

   return 0;
}</pre>
```







#### Conditional Constructs - for

#### **Syntax**

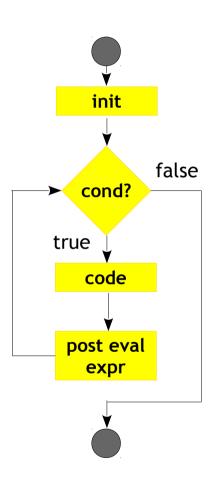
```
for (init; condition; post evaluation expr)
   statement(s);
```

- Controls the loop.
- Evaluated **before** each execution of loop body

#### 013\_example.c

```
#include <stdio.h>
int main()
    int iter;
    for (iter = 0; iter < 10; iter++)</pre>
       printf("Looped %d times\n", iter);
    return 0;
```







#### Conditional Constructs - Classwork



- W.A.P to print the power of two series using for loop
  - $-2^{1}, 2^{2}, 2^{3}, 2^{4}, 2^{5} \dots$
- W.A.P to print the power of N series using Loops
  - N<sup>1</sup>, N<sup>2</sup>, N<sup>3</sup>, N<sup>4</sup>, N<sup>5</sup> ...
- W.A.P to multiply 2 nos without multiplication operator
- W.A.P to check whether a number is palindrome or not



#### Conditional Constructs - for - DIY



- WAP to print line pattern
  - Read total (n) number of pattern chars in a line (number should be "odd")
  - Read number (m) of pattern char to be printed in the middle of line ("odd" number)
  - Print the line with two different pattern chars
  - Example Let's say two types of pattern chars '\$' and '\*' to be printed in a line. Total number of chars to be printed in a line are 9. Three '\*' to be printed in middle of line.
    - Output ==> \$\$\$\* \* \*\$\$\$



#### Conditional Constructs - for - DIY



Based on previous example print following pyramid



## Advanced C Conditional Constructs - for - DIY

Print rhombus using for loops



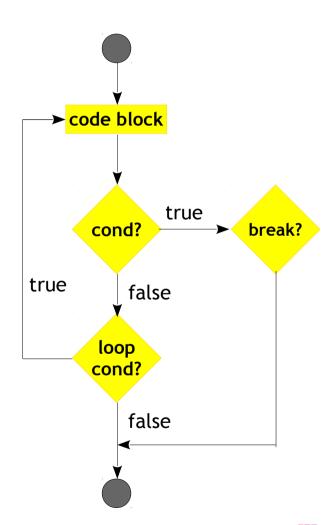
#### Conditional Constructs - break

- A break statement shall appear only in "switch body" or "loop body"
- "break" is used to exit the loop, the statements appearing after break in the loop will be skipped

#### **Syntax**

```
do
{
    conditional statement
       break;
} while (condition);
```







#### Conditional Constructs - break

```
#include <stdio.h>
int main()
   int iter;
   for (iter = 0; iter < 10; iter++)</pre>
       if (iter == 5)
           break;
       printf("%d\n", iter);
   return 0;
```

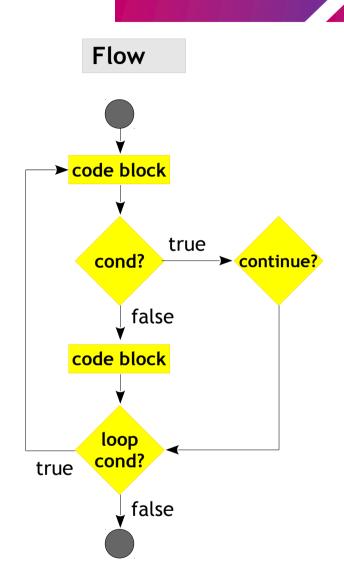


#### Conditional Constructs - continue

- A continue statement causes a jump to the loop-continuation portion, that is, to the end of the loop body
- The execution of code appearing after the continue will be skipped
- Can be used in any type of multi iteration loop

#### **Syntax**

```
do
{
    conditional statement
        continue;
} while (condition);
```





#### Conditional Constructs - continue

```
#include <stdio.h>
int main()
   int iter;
   for (iter = 0; iter < 10; iter++)</pre>
       if (iter == 5)
           continue;
       printf("%d\n", iter);
   return 0;
```



#### Conditional Constructs - goto

- A goto statement causes a unconditional jump to a labeled statement
- Generally avoided in general programming, since it sometimes becomes tough to trace the flow of the code

#### **Syntax**

```
goto label;
...
label:
...
```



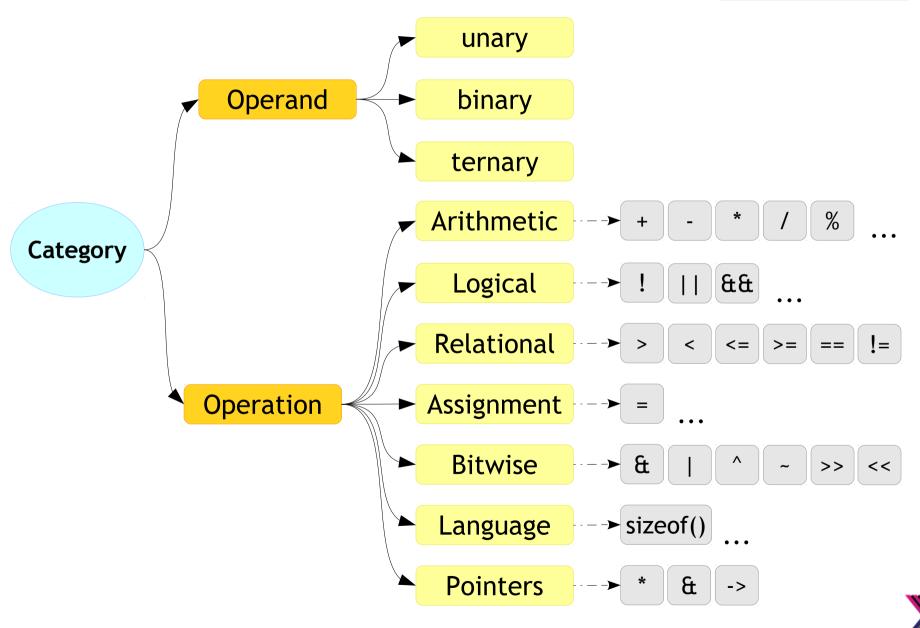
# Operators

# Advanced C Operators

- Symbols that instructs the compiler to perform specific arithmetic or logical operation on operands
- All C operators do 2 things
  - Operates on its operands
  - Returns a value



**Operators** 



#### Operators - Precedence and Associativity

Operators	Associativity	Precedence
() [] -> .	L - R	HIGH
! ~ ++ + * & (type) sizeof	R - L	
/ % *	L - R	
+ -	L - R	
<< >>	L - R	
< <= > >=	L - R	
== !=	L - R	
&	L - R	
^	L - R	
	L - R	
££	L - R	
11	L - R	
?:	R - L	
= += -= *= /= %= &t= ^=  = <<= >>=	R - L	
,	L - R	LOW

#### Note:

post ++ and -operators have
higher precedence
than pre ++ and -operators

(Rel-99 spec)



#### Operators - Arithmetic



Operator	Description	Associativity
/	Division	
*	Multiplication	
%	Modulo	L to R
+	Addition	
_	Subtraction	

#### 016\_example.c

```
#include <stdio.h>
int main()
{
   int num;
   num = 7 - 4 * 3 / 2 + 5;
   printf("Result is %d\n", num);
   return 0;
}
```

What will be the output?



Operators - Language - sizeof()

#### 017\_example.c

```
#include <stdio.h>
int main()
{
   int num = 5;
   printf("%u:%u:%u\n", sizeof(int), sizeof num, sizeof 5);
   return 0;
}
```

```
#include <stdio.h>
int main()
{
   int num1 = 5;
   int num2 = sizeof(++num1);

   printf("num1 is %d and num2 is %d\n", num1, num2);

   return 0;
}
```



Operators - Language - sizeof()



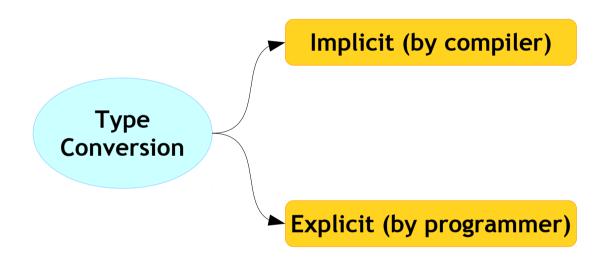
- Any type of operands
- Type as an operand
- No brackets needed across operands





# Advanced C Type Conversion







# Advanced C Type Conversion Hierarchy



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



## Type Conversion - Implicit



- Automatic Unary conversions
  - The result of + and are promoted to int if operands are char and short
  - The result of ~ and ! is integer
- Automatic Binary conversions
  - If one operand is of LOWER RANK (LR) data type & other is of HIGHER RANK (HR) data type then LOWER RANK will be converted to HIGHER RANK while evaluating the expression.
  - Example: LR + HR → LR converted to HR



#### Type Conversion - Implicit



- Type promotion
  - LHS type is HR and RHS type is LR → int = char → LR is promoted to HR while assigning
- Type demotion
  - LHS is LR and RHS is HR → int = float → HR rank will be demoted to LR. Truncated



Type Conversion - Explicit (Type Casting)

#### **Syntax**

```
(data_type) expression
```

```
#include <stdio.h>
int main()
{
   int num1 = 5, num2 = 3;
   float num3 = (float) num1 / num2;
   printf("nun3 is %f\n", num3);
   return 0;
}
```



#### Operators - Logical



Associativity

```
Operator
                                          Description
020_example.c
                                          Logical NOT
                                          Logical AND
                               88
#include <stdio.h>
                                          Logical OR
int main()
    int num1 = 1, num2 = 0;
    if (++num1 || num2++)
    {
       printf("num1 is %d num2 is %d\n", num1, num2);
    num1 = 1, num2 = 0;
    if (num1++ && ++num2)
       printf("num1 is %d num2 is %d\n", num1, num2);
    }
    else
       printf("num1 is %d num2 is %d\n", num1, num2);
   return 0;
```

What will be the output?

R to L

L to R

L to R



#### Operators - Circuit Logical



- Have the ability to "short circuit" a calculation if the result is definitely known, this can improve efficiency
  - Logical AND operator ( && )
    - If one operand is false, the result is false.
  - Logical OR operator ( | | )
    - If one operand is true, the result is true.



#### Operators - Relational



Operator	Description	Associativity
>	Greater than	
<	Lesser than	
>=	Greater than or equal	L to R
<=	Lesser than or equal	LUK
==	Equal to	
!=	Not Equal to	

#### 021\_example.c

```
#include <stdio.h>
int main()
   float num1 = 0.7;
   if (num1 == 0.7)
       printf("Yes, it is equal\n");
   else
       printf("No, it is not equal\n");
   return 0;
```

What will be the output?



#### Operators - Assignment

#### 022\_example.c

```
#include <stdio.h>
int main()
{
    int num1 = 1, num2 = 1;
    float num3 = 1.7, num4 = 1.5;

    num1 += num2 += num3 += num4;

    printf("num1 is %d\n", num1);

    return 0;
}
```

```
#include <stdio.h>
int main()
{
    float num1 = 1;
    if (num1 = 1)
    {
        printf("Yes, it is equal!!\n");
    }
    else
    {
        printf("No, it is not equal\n");
    }
    return 0;
}
```



# Advanced C Operators - Bitwise

- Bitwise operators perform operations on bits
- The operand type shall be integral
- Return type is integral value



# Advanced C Operators - Bitwise



		Dituria a ANDina a of	Operand	Value								
G.	& Bitwise AND	Bitwise ANDing of all the bits in two operands	A	0x61	C	1	1	0	0	0	0	1
α			В	0x13	C	0	0	1	0	0	1	1
		operands	A & B	0x01	C	0	0	0	0	0	0	1

Bitwise OR Bitwise ORing of all the bits in two operands

Operand

Operand

Operand

Ox61

Ox61

Ox13

Ox13

Ox13

Ox73

Ox11

Ox73

Ox11

Ox73

Ox73

Ox11

Ox73

Ox11

Ox73



# Advanced C Operators - Bitwise



^ Bitwise XOR

Bitwise XORing of all the bits in two operands

Operand	Value	
A	0x61	01100001
В	0x13	00010011
A ^ B	0x72	0 1 1 1 0 0 1 0

Compliment

Complimenting all the bits of the operand

 Operand
 Value

 A
 0x61

 0
 1

 0
 1

 0
 1

 1
 0

 1
 0

 1
 0

 1
 0

 0
 1

 1
 0

 0
 1

 0
 1

 0
 1

 0
 1

 0
 0

 1
 0

 0
 0

 0
 0

 0
 0

 0
 0

 0
 0

 0
 0

 0
 0

 0
 0

 0
 0

 0
 0

 0
 0

 0
 0

 0
 0

 0
 0

 0
 0

 0
 0

 0
 0

 0
 0

 0



Operators - Bitwise - Shift



#### **Syntax**

```
Left Shift:

shift-expression << additive-expression
(left operand) (right operand)

Right Shift:

shift-expression >> additive-expression
(left operand) (right operand)
```

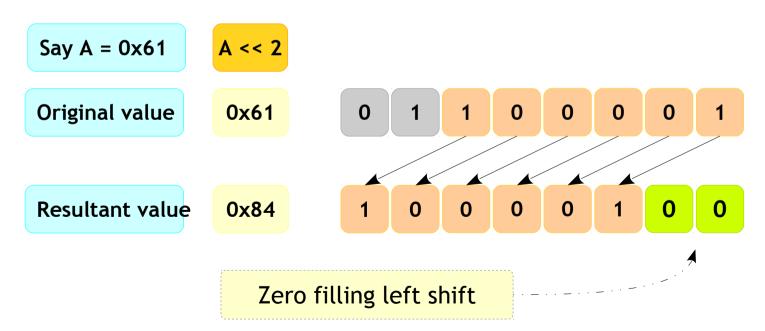


Operators - Bitwise - Left Shift



#### 'Value' << 'Bits Count'

- Value: Is shift operand on which bit shifting effect to be applied
- Bits count: By how many bit(s) the given "Value" to be shifted



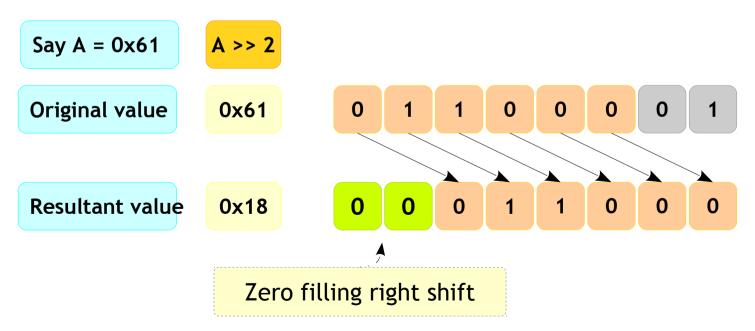


Operators - Bitwise - Right Shift



#### 'Value' >> 'Bits Count'

- Value: Is shift operand on which bit shifting effect to be applied
- Bits count: By how many bit(s) the given "Value" to be shifted



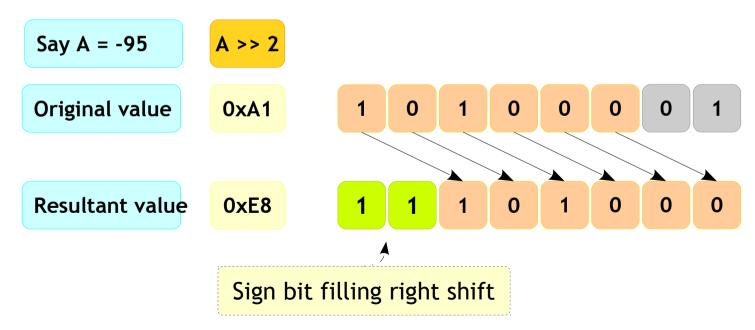






## "Signed Value' >> 'Bits Count'

- Same operation as mentioned in previous slide.
- But the sign bits gets propagated.





#### Operators - Bitwise

```
#include <stdio.h>
int main()
    int count;
    unsigned char iter = 0xFF;
    for (count = 0; iter != 0; iter >>= 1)
        if (iter & 01)
             count++;
    printf("count is %d\n", count);
    return 0;
```



Operators - Bitwise - Shift



- Each of the operands shall have integer type
- The integer promotions are performed on each of the operands
- If the value of the right operand is negative or is greater than or equal to the width of the promoted left operand, the behavior is undefined
- Left shift (<<) operator: If left operand has a signed type and nonnegative value, and (left\_operand \* (2^n)) is representable in the result type, then that is the resulting value; otherwise, the behavior is undefined



#### Operators - Bitwise - Shift



```
#include <stdio.h>
int main()
{
   int x = 7, y = 7;

   x = 7 << 32;
   printf("x is %x\n", x);

   x = y << 32;
   printf("x is %x\n", x);

   return 0;
}</pre>
```



Operators - Bitwise - Mask

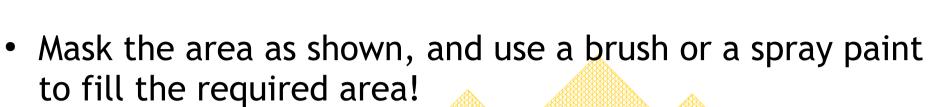


• If you want to create the below art assuming your are not a good painter, What would you do?





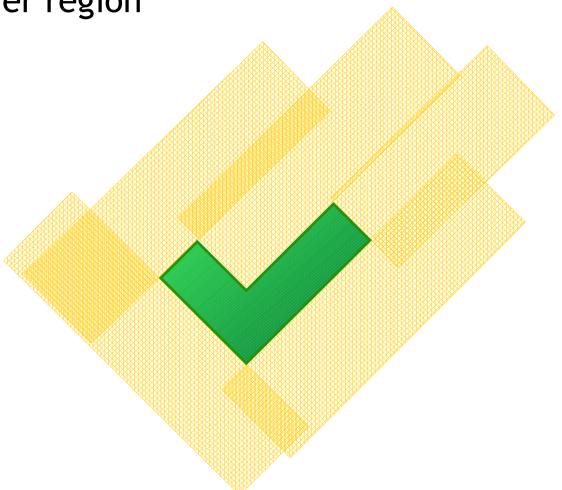
Operators - Bitwise - Mask





Operators - Bitwise - Mask







Operators - Bitwise - Mask

Remove the mask tape





Operators - Bitwise - Mask



- So masking, technically means unprotecting the required bits of register and perform the actions like
  - Set Bit
  - Clear Bit
  - Get Bitetc,..



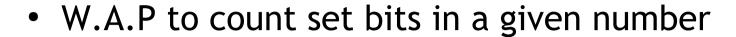
Operators - Bitwise - Mask



	7	5	5	4	3	2	1	0	Bit Position
	0 1	1	1	0	0	0	0	1	The register to be modified
& Operator	1 1	1	0	1	1	1	1	1	Mask to <b>CLEAR 5</b> <sup>th</sup> <b>bit</b> position
	0 1	1	0	0	0	0	0	1	The result
l O	0 1	1	0	0	0	0	0	1	The register to be modified
Operator	0	ו	1	0	0	0	0	0	Mask to <b>SET 5</b> <sup>th</sup> <b>bit</b> position
	0 1	1	1	0	0	0	0	1	The result



Operators - Bitwise - Shift



- W.A.P to print bits of given number
- W.A.P to swap nibbles of given number



#### Operators - Ternary

#### **Syntax**

```
Condition ? Expression 1 : Expression 2;
```

```
#include <stdio.h>
int main()
   int num1 = 10;
   int num2 = 20;
   int num3;
   if (num1 > num2)
       num3 = num1;
   else
       num3 = num2;
   printf("%d\n", num3);
   return 0;
```

```
#include <stdio.h>
int main()
{
   int num1 = 10;
   int num2 = 20;
   int num3;

   num3 = num1 > num2 ? num1 : num2;
   printf("Greater num is %d\n", num3);

   return 0;
}
```



#### Operators - Comma



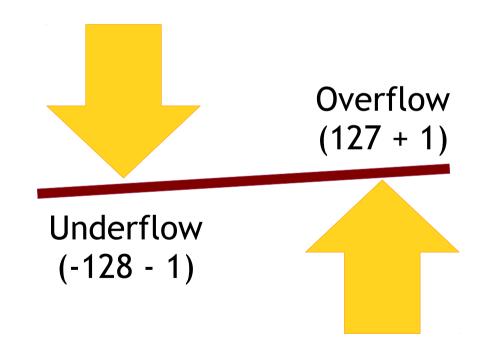
- The left operand of a comma operator is evaluated as a void expression (result discarded)
- Then the right operand is evaluated; the result has its type and value
- Comma acts as separator (not an operator) in following cases -
  - Arguments to function
  - Lists of initializers (variable declarations)
- But, can be used with parentheses as function arguments such as -
  - foo ((x = 2, x + 3)); // final value of argument is 5



#### Over and Underflow



- 8-bit Integral types can hold certain ranges of values
- So what happens when we try to traverse this boundary?





## Overflow - Signed Numbers



Say 
$$A = +127$$

Original value

0x7F

1

 0
 1
 1
 1
 1
 1
 1
 1

Add

0 0

0 0

0 1

Resultant value 0

0x80

1 0 0 0 0 0 0 0

Sign bit



## **Underflow - Signed Numbers**



Say 
$$A = -128$$

Original value

0x80

Add

-1

Resultant value

0x7F

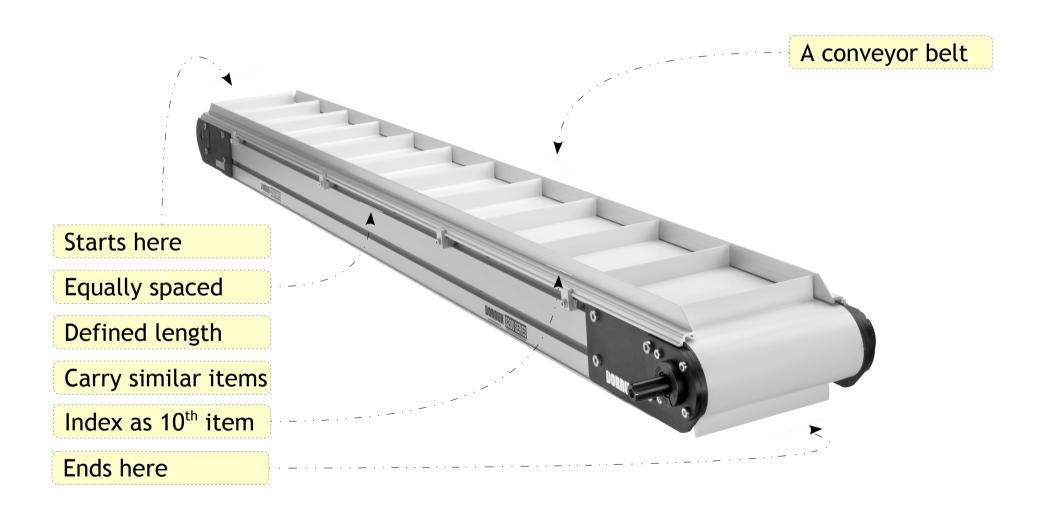
Sign bit Spill over bit is discarded



# Arrays

Arrays - Know the Concept

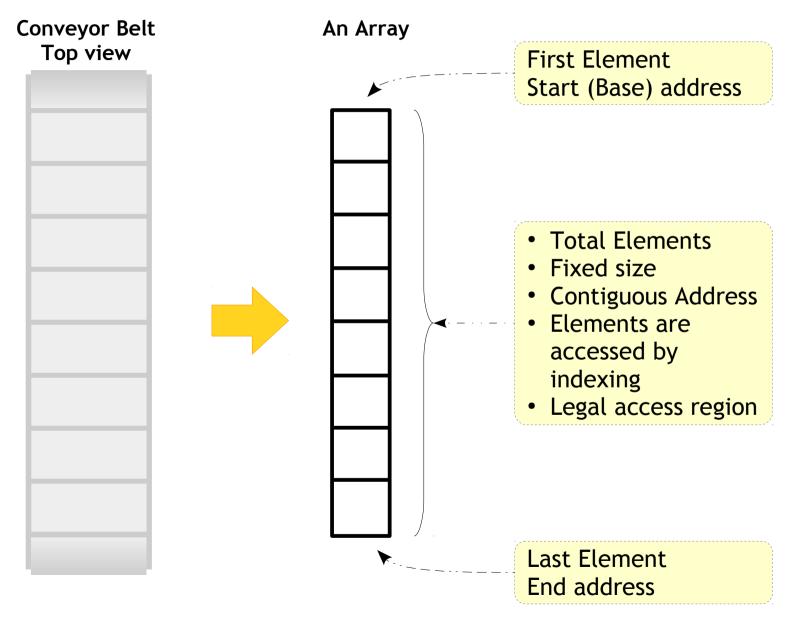






#### Arrays - Know the Concept







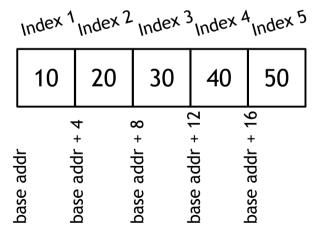
# Advanced C Arrays



#### **Syntax**

#### Example

```
int age[5] = \{10, 20, 30, 40, 50\};
```





#### Arrays - Points to be noted



- An array is a collection of similar data type
- Elements occupy consecutive memory locations (addresses)
- First element with lowest address and the last element with highest address
- Elements are indexed from 0 to SIZE 1. Example: 5 elements array (say array[5]) will be indexed from 0 to 4
- Accessing out of range array elements would be "illegal access"
  - Example: Do not access elements array[-1] and array[SIZE]
- Array size can't be altered at run time



#### Arrays - Why?

```
#include <stdio.h>
int main()
    int num1 = 10;
    int num2 = 20;
   int num3 = 30;
    int num4 = 40;
    int num5 = 50;
   printf("%d\n", num1);
   printf("%d\n", num2);
   printf("%d\n", num3);
   printf("%d\n", num4);
   printf("%d\n", num5);
   return 0;
```

```
#include <stdio.h>
int main()
{
    int num_array[5] = {10, 20, 30, 40, 50};
    int index;

    for (index = 0; index < 5; index++)
        {
            printf("%d\n", num_array[index]);
        }

    return 0;
}</pre>
```



## Arrays - Reading

```
#include <stdio.h>
int main()
   int num_array[5] = {1, 2, 3, 4, 5};
   int index;
   index = 0;
   do
       printf("Index %d has Element %d\n", index, num array[index]);
       index++;
   \} while (index < 5);
   return 0;
```



#### Arrays - Storing

```
#include <stdio.h>
int main()
{
    int num_array[5];
    int index;

    for (index = 0; index < 5; index++)
    {
        scanf("%d", &num_array[index]);
    }

    return 0;
}</pre>
```



#### Arrays - Initializing

```
#include <stdio.h>
int main()
   int array1[5] = \{1, 2, 3, 4, 5\};
   int array2[5] = \{1, 2\};
   int array3[] = {1, 2};
   int array4[]; /* Invalid */
   printf("%u\n", sizeof(array1));
   printf("%u\n", sizeof(array2));
   printf("%u\n", sizeof(array3));
   return 0;
```



Arrays - Copying

Can we copy 2 arrays? If yes how?

```
#include <stdio.h>
int main()
   int array_org[5] = {1, 2, 3, 4, 5};
   int array bak[5];
   int index;
   array bak = array org;
   if (array bak == array org)
       printf("Copied\n");
   return 0;
```





# Advanced C Arrays - Copying



- No!! its not so simple to copy two arrays as put in the previous slide. C doesn't support it!
- Then how to copy an array?
- It has to be copied element by element





- Arrays DIY
- W.A.P to find the average of elements stored in a array.
  - Read value of elements from user
  - For given set of values: { 13, 5, -1, 8, 17 }
  - Average Result = 8.4
- W.A.P to find the largest array element
  - Example 100 is the largest in {5, **100**, -2, 75, 42}



#### Arrays - DIY



- W.A.P to compare two arrays (element by element).
  - Take equal size arrays
  - Arrays shall have unique values stored in random order
  - Array elements shall be entered by user
  - Arrays are compared "EQUAL" if there is one to one mapping of array elements value
  - Print final result "EQUAL" or "NOT EQUAL"

#### Example of Equal Arrays:

$$- A[3] = \{2, -50, 17\}$$

$$-B[3] = \{17, 2, -50\}$$



Arrays - Oops!! what is this now?





