“Advanced C Programming”

Day 1

1. Introduction to Linux, vi editor and C
2. **Data Types in C & Operators in C**

2. (a) Data types

C Language Character Set

- C is a language, hence it requires characters to build its building blocks.
- Every character has its own ASCII Value
- C character Set contains:
  - The uppercase letters A to Z
  - The lowercase letters a to z
  - The digits 0 to 9
  - Certain special characters

+	-	*	/	=	%	&	#
!	?	^	"	'	~	\	|
<	>	(	)	[	]	{	}
:	;	.	,	_		*(blank space)*	

Character Set – ASCII Values

ASCII - American Standard Code for Information Interchange

Total – 128 Characters
Printable Characters – 95
Non-printable Control characters - 33

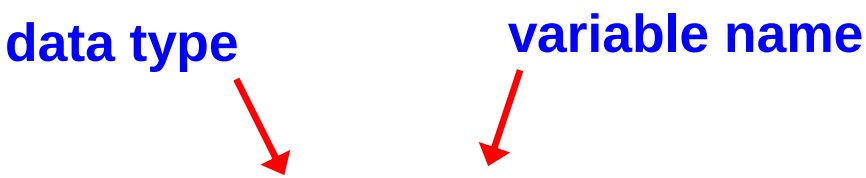
Characters	ASCII Values
A – Z	65 – 90
a – z	97 – 122
0 – 9	48 – 57
special symbols	0 - 47, 58 - 64, 91 - 96, 123 - 127

Basics

- Tokens: Individual unit of C program is called as Tokens
- Variable declarations: int i; float f;
- Initialization: char c='A'; int x=y=10;
- Operators: +,−,∗,/,% and many more….
- Expressions: int x,y,z; x=y∗2+z∗3;
- Functions: int factorial (int n); /∗function takes int , returns int ∗/

Variables

- A variable is a name for a location in memory
- A variable must be declared by specifying the variable's name and the type of information that it will hold
- Variable should be declared before we use it




```
int total;

int count, temp, result;
```

Multiple variables can be created in one declaration

Variable Initialization

- A variable can be given an initial value in the declaration

```
int sum = 0;
int base = 32, max = 149;
```

- When a variable is referenced in a program, its current value is used

Variable Assignment

- An *assignment statement* changes the value of a variable
- The assignment operator is the = sign

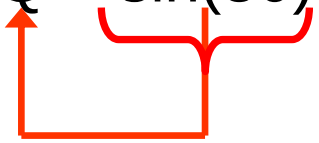
```
total = 55;
```

- The expression on the right is evaluated and the result is stored in the variable on the left
  - The value that was in total is overwritten
  - You can only assign a value to a variable that is consistent with the variable's declared type

Assignment Through a Function

- y = f(x);

Q = sin(30);



- The assignment operator is still the = sign

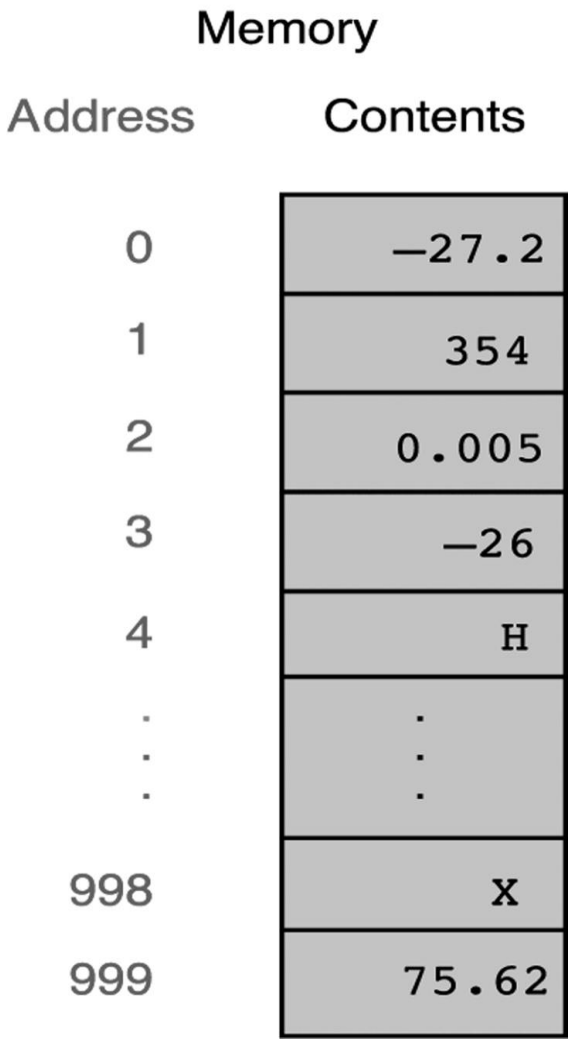
Assignment Through scanf()

```
int variable;

scanf("%d", &variable);
```

- <keyboardinput> 30
- There is not assignment operator in this case

Variables in Memory



Binary Numbers

- Once information is digitized, it is represented and stored in memory using the *binary number system*
- A single binary digit (0 or 1) is called a *bit*
- Devices that store and move information are cheaper and more reliable if they have to represent only two states
- A single bit can represent two possible states, like a light bulb that is either on (1) or off (0)
- Permutations of bits are used to store values

Bit Permutations

1 bit	2 bits	3 bits	4 bits	
0	00	000	0000	1000
1	01	001	0001	1001
	10	010	0010	1010
	11	011	0011	1011
		100	0100	1100
		101	0101	1101
		110	0110	1110
		111	0111	1111

Each additional bit doubles the number of possible permutations

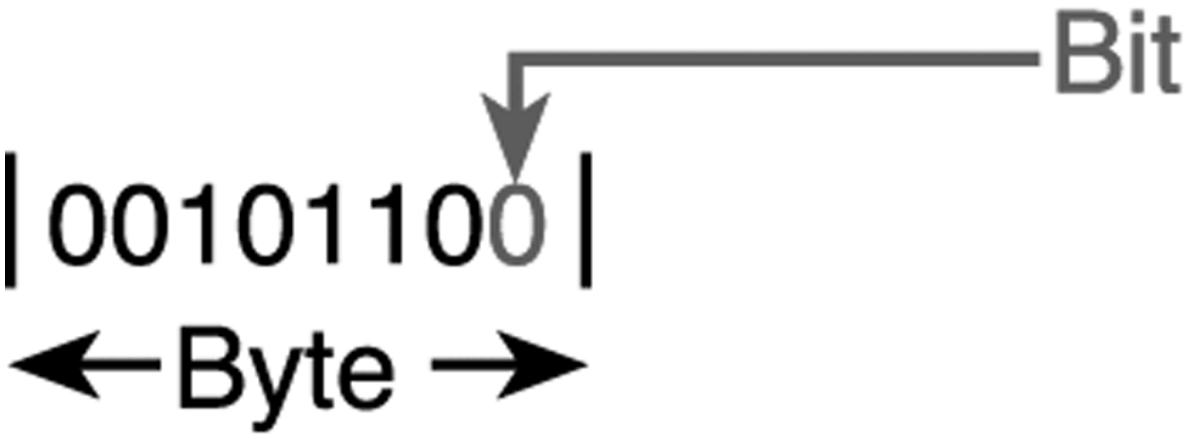
Bit Permutations

- Each permutation can represent a particular item
- There are 2^N permutations of N bits
- Therefore, N bits are needed to represent 2^N unique items

How many items can be represented by

1 bit ?	$2^1 = 2$ **items**
2 bits ?	$2^2 = 4$ **items**
3 bits ?	$2^3 = 8$ **items**
4 bits ?	$2^4 = 16$ **items**
5 bits ?	$2^5 = 32$ **items**

Relationship Between a Byte and a Bit



What is the value of this binary

0 0 1 0 1 1 0 0

0 0 1 0 1 1 0 0

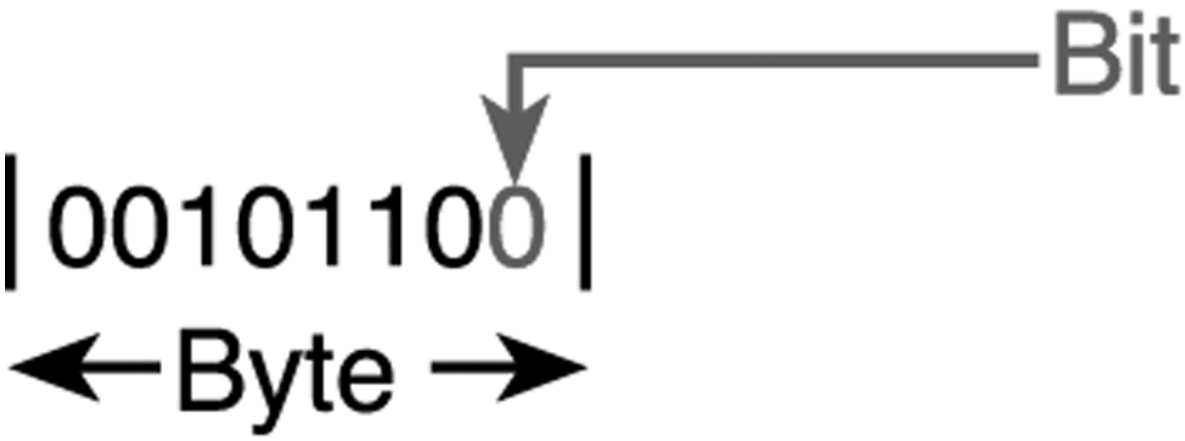
$0*2^7 + 0*2^6 + 1*2^5 + 0*2^4 + 1*2^3 + 1*2^2 + 0*2^1 + 0*2^0$

0*128 + 0*64 + 1*32 + 0*16 + 1*8 + 1*4 + 0*2 + 0*1

0*128 + 0*64 + 1*32 + 0*16 + 1*8 + 1*4 + 0*2 + 0*1

32+ 8 + 4 = 44 (in decimal)

What is the maximum number that can be stored in one byte (8 bits)?



What is the max.num. that can be stored in one byte (8 bits)?

1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1

$1*2^7 + 1*2^6 + 1*2^5 + 1*2^4 + 1*2^3 + 1*2^2 + 1*2^1 + 1*2^0$

$1*128 + 1*64 + 1*32 + 1*16 + 1*8 + 1*4 + 1*2 + 1*1$

$128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255$ (in decimal)

Another way is: $1*2^8 – 1 = 256 – 1 = 255$

What would happen if we try to add 1 to the largest number that can be stored in one byte (8 bits)?

```
    1  1  1  1  1  1  1  1
+
                        1
    ---------------------------

 1  0  0  0  0  0  0  0  0

    0  0  0  0  0  0  0  0
```

Variables…

Variable Naming Rules:

- Variable names can contain letters, digits and _;
- First character must be a letter
- Variable names should start with letters.
- Keywords (e.g., for, while, do, if, switch etc.) cannot be used as variable names and are reserved
- Variable names are case sensitive.

Example: int x, X; → x and X are two different variables

Variable Definition & Declaration

Variable Definition: A variable definition means to tell the compiler where and how much to **create the storage** for the variable.

Examples:

```
int i, j, k;
char ch;
```

Variable Declaration: A variable declaration provides assurance to the compiler that **there is one variable existing with the given type and name** so that compiler proceed for further compilation without needing complete detail about the variable.

Example:

```
extern int dssd; ← Declaration
int main()
{
        int dssd; ← Definition

}
```

Keywords in C

List of C keyword			
auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

- **Total keywords - 32**
- **Additional C99 key words (Total -5):**
  - **inline –** For writing inline functions (Discuss in functions)
  - **_imaginary** – To declare imaginary values (iy)
  - **_complex** – To declare complex variables (a+ib)
  - **_Bool** – To declare a Boolean type variable (stdbool.h)
  - **restrict –** To declare restricted pointers

Quiz

int money$owed;	←	Incorrect
int total_count;	←	Correct
int score2 ;	←	Correct
int 2ndscore;	←	Incorrect
int long;	←	Incorrect
_Bool x;	←	Correct (B must be capital letter or “bool”)

Data Types

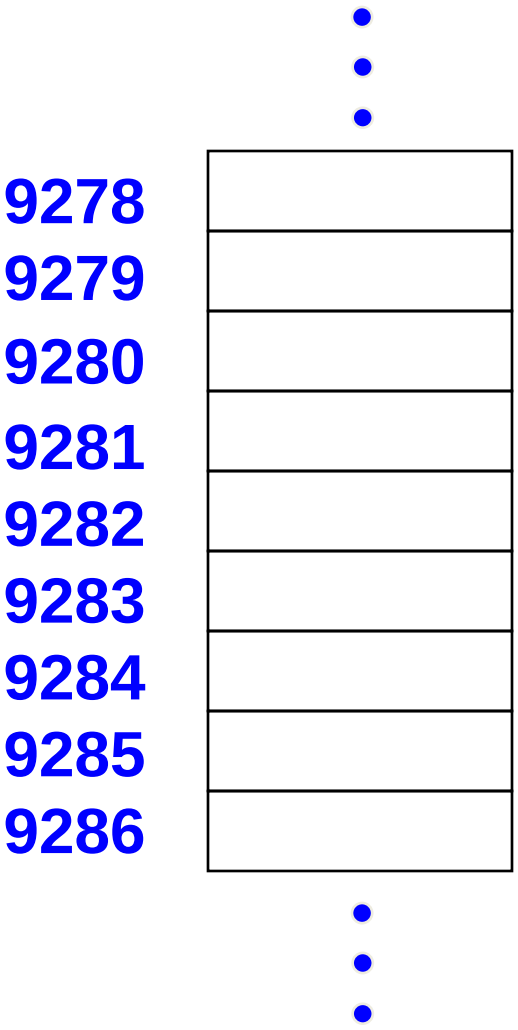
- Data type of an object determines:
  - Type of values it can have
  - Type of operations that can be performed on it

- Data Types:
  - Basic data types → int, char, float, double
  - Enumerated data types → enum
  - Void data type → void
  - Derived data types → Arrays, Pointers, Structures and Unions

Sizes of Basic data types

Type	Size	Value Range
char	1	-128 to 127
unsigned char	1	0 to 255
signed char	1	-128 to 127
int	2 or 4	-32,768 to 32,767 (or) -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4	0 to 65,535 (or) 0 to 4,294,967,295
Short	2	-32,768 to 32,767
unsigned short	2	0 to 65,535
long	4	-2,147,483,648 to 2,147,483,647
unsigned long	4	0 to 4,294,967,295

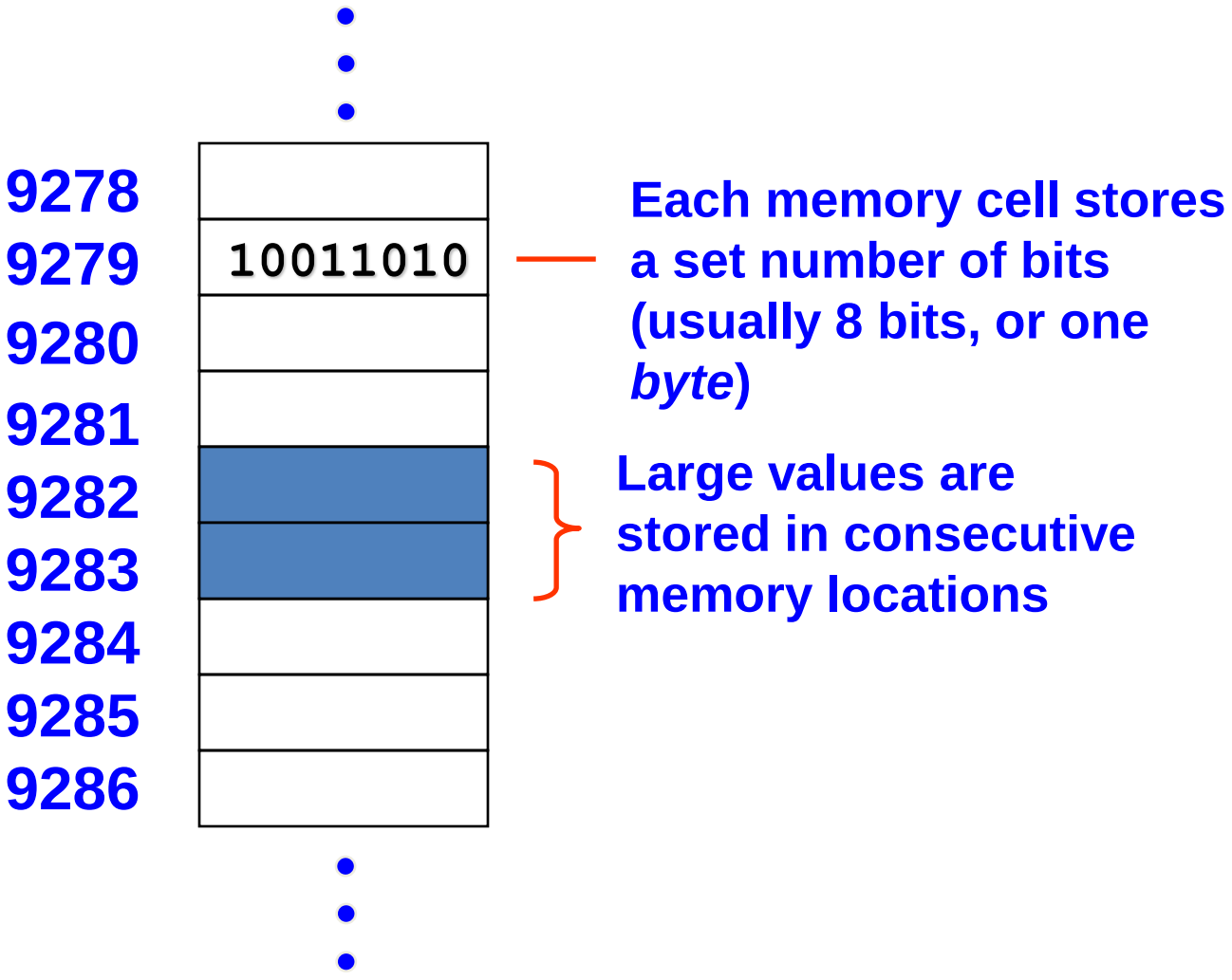
Computer Memory



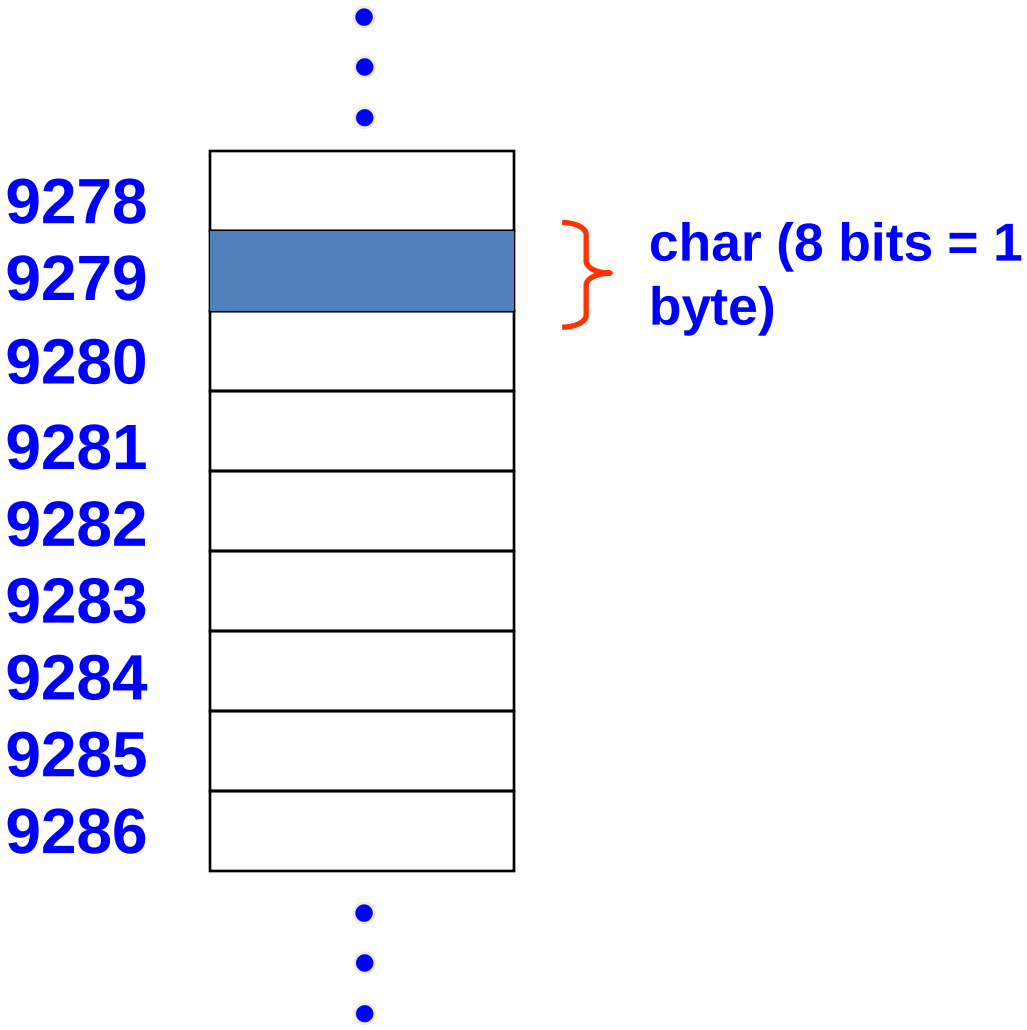
Main memory is divided into many memory locations (or *cells*)

Each memory cell has a numeric *address*, which uniquely identifies it

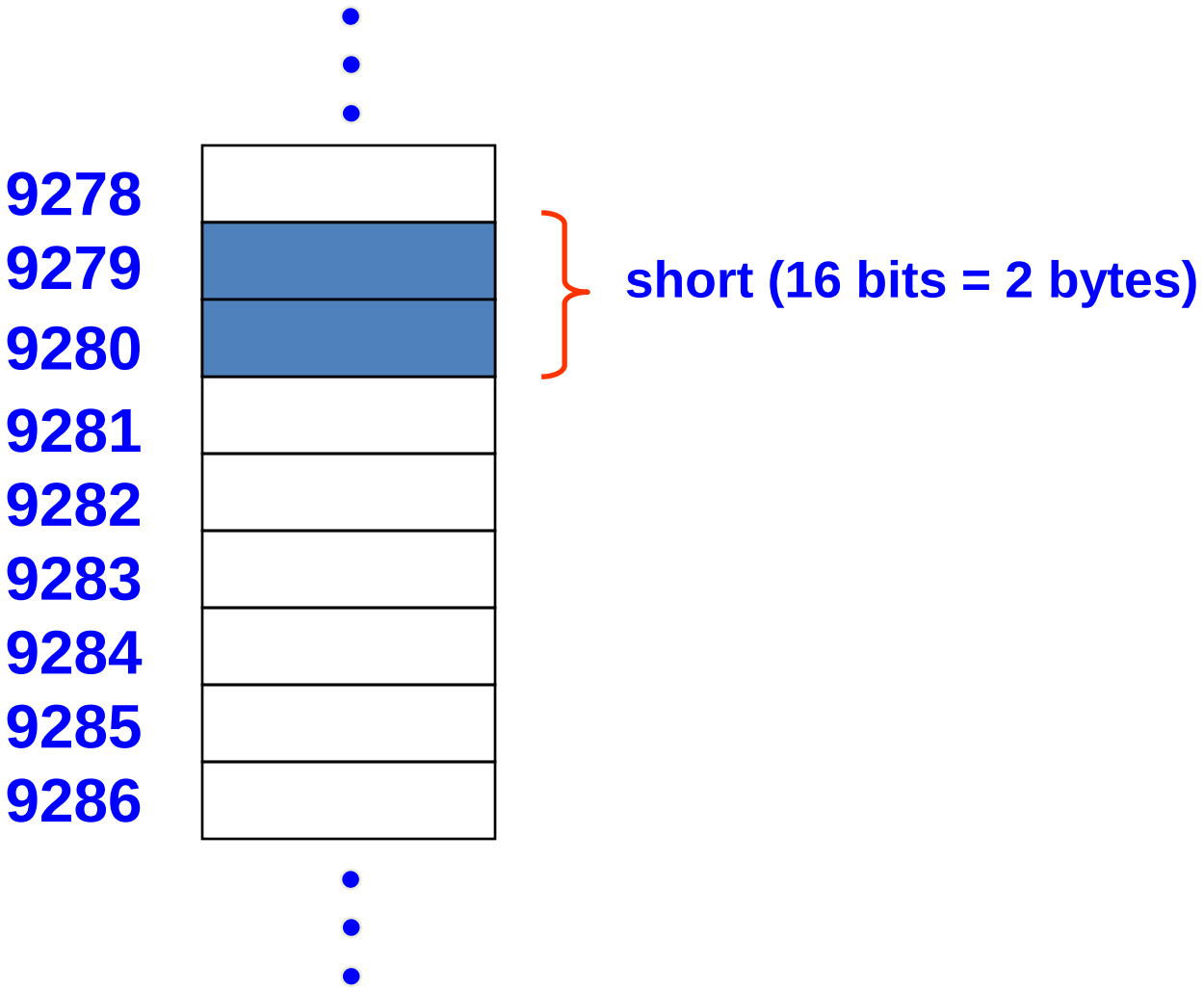
Storing Information



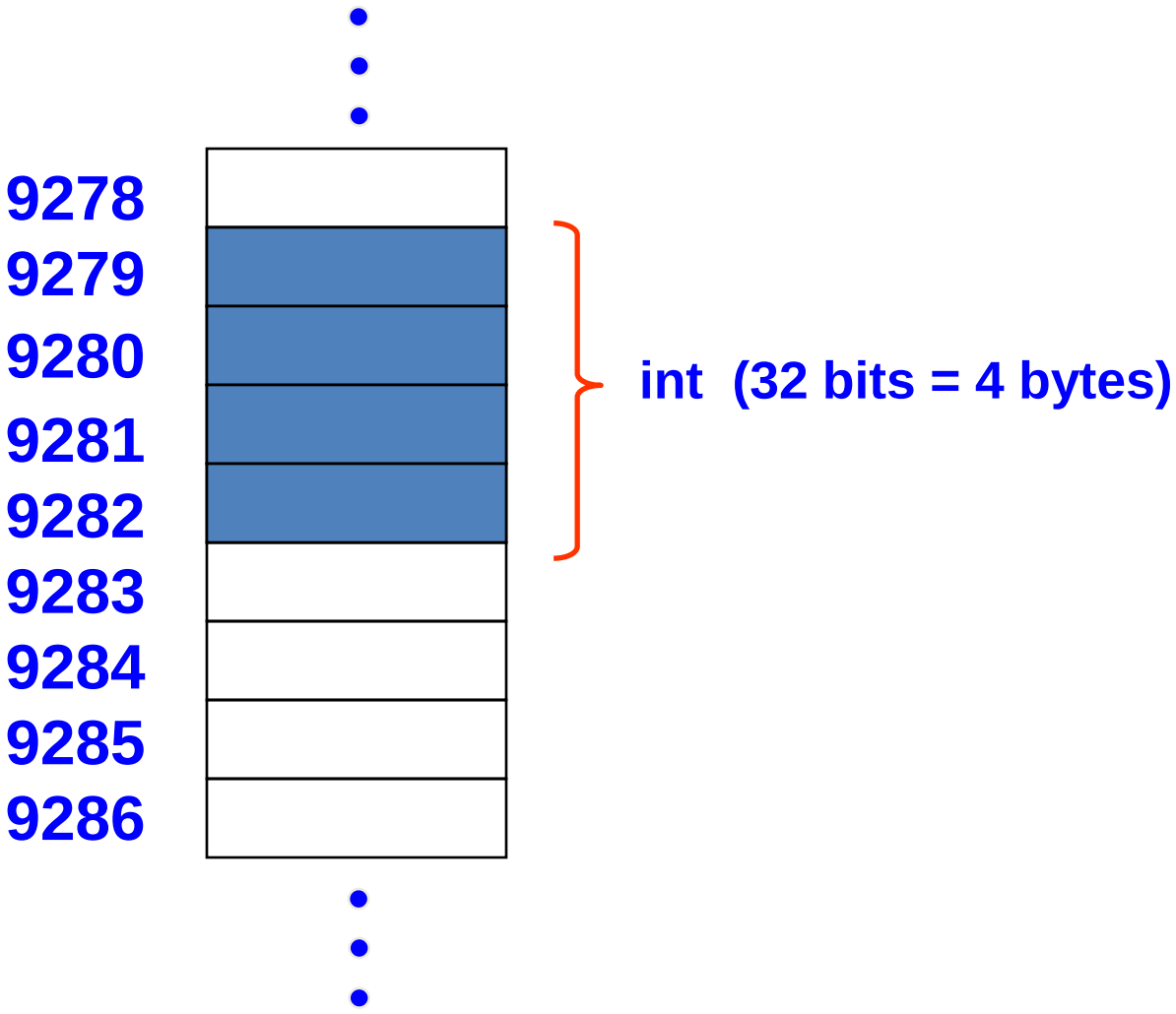
Storing a char



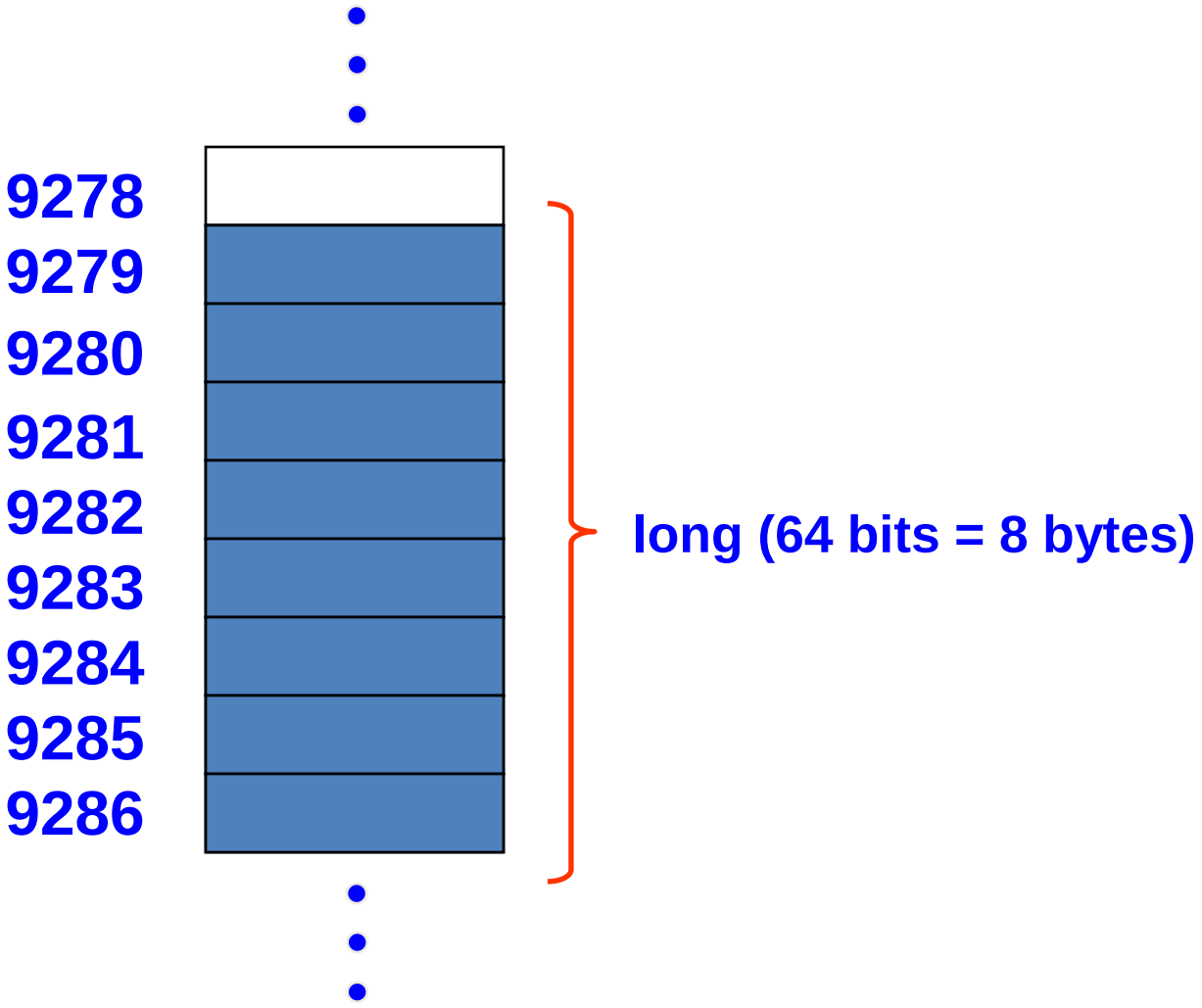
Storing a short



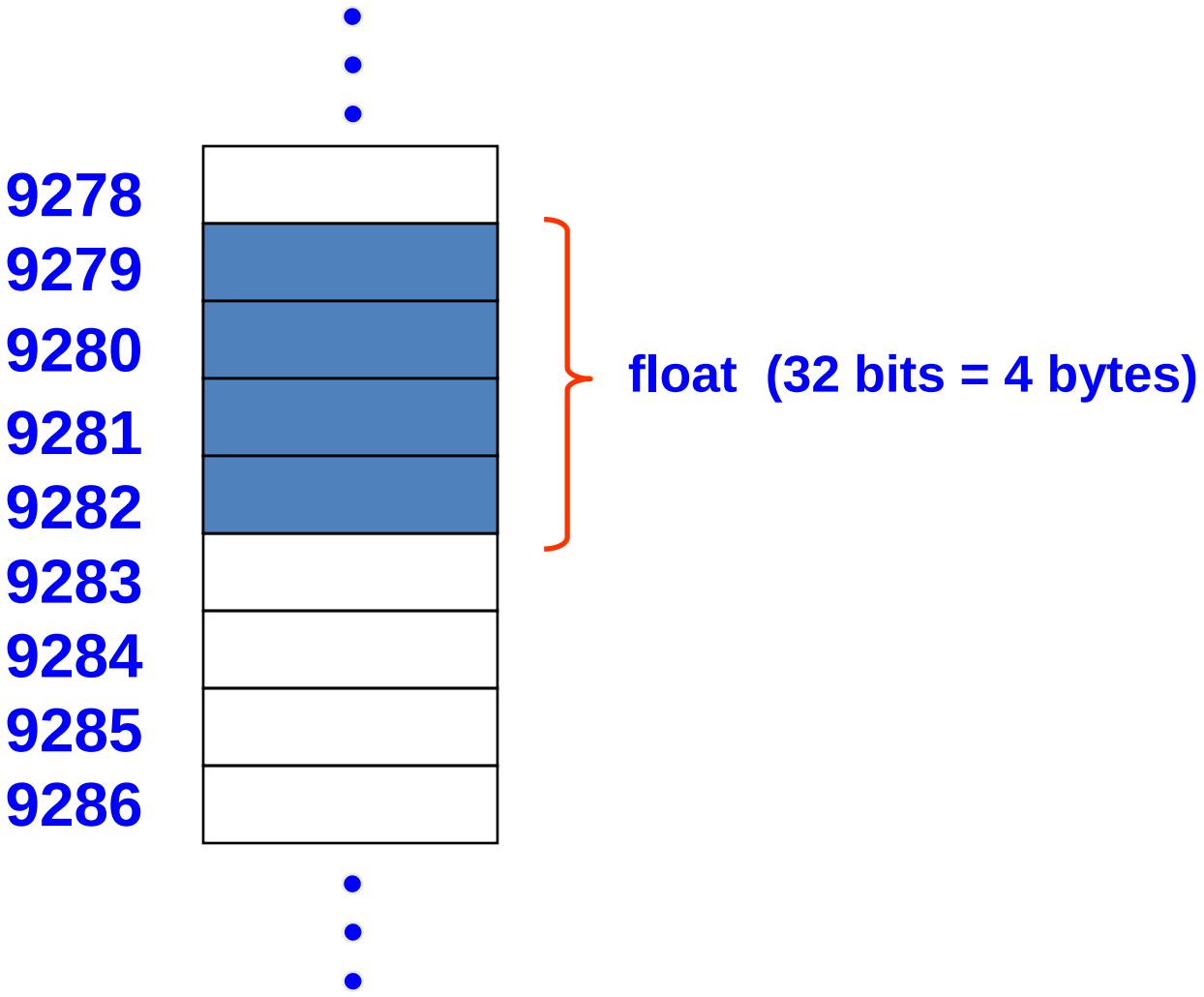
Storing an int



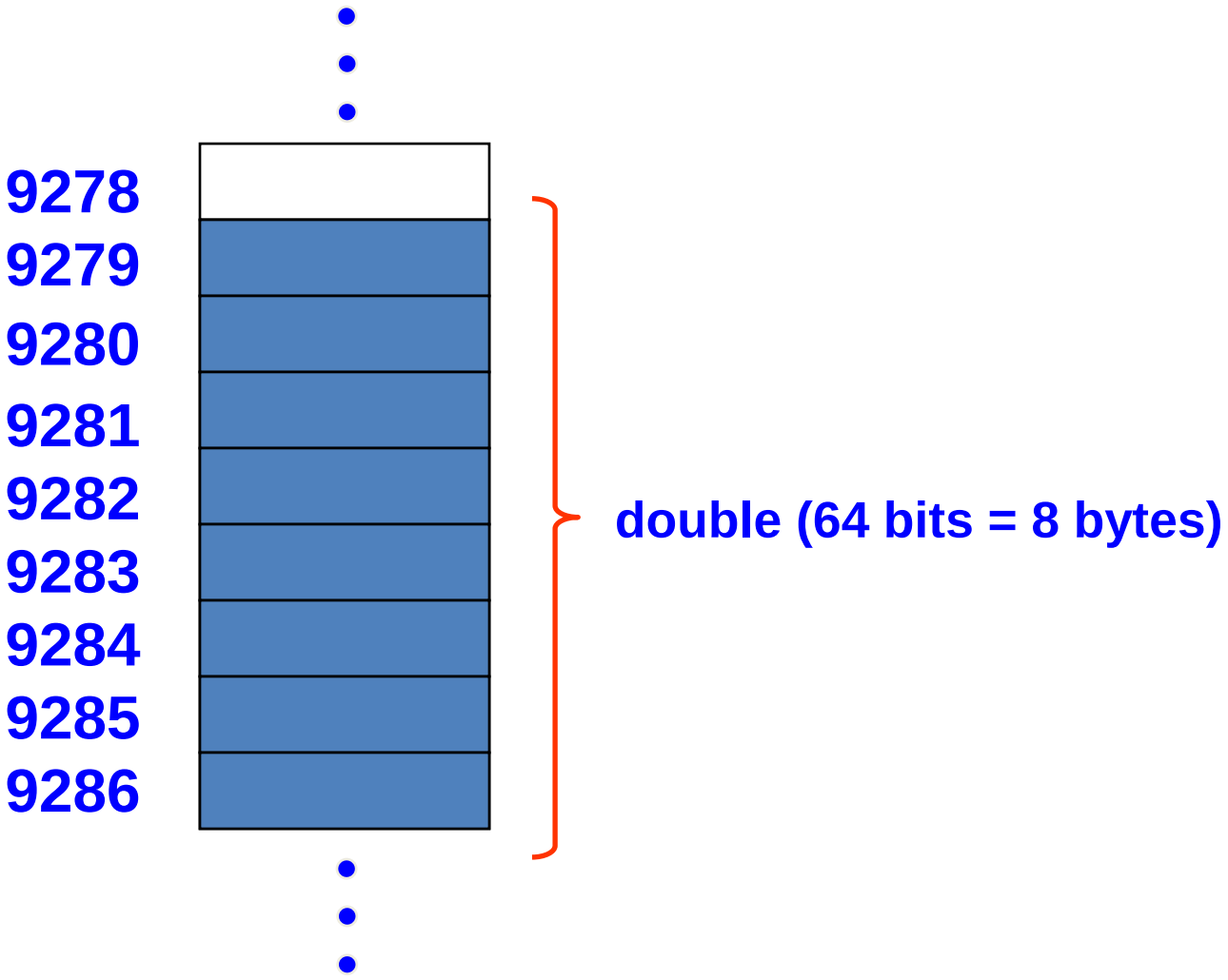
Storing a long



Storing a float



Storing a double



The sizeof() operator

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
  printf("Size of char        = %d \n", sizeof(char));
  printf("Size of short       = %d \n", sizeof(short));
  printf("Size of int         = %d \n", sizeof(int));
  printf("Size of long        = %d \n", sizeof(long));
  printf("Size of long long   = %d \n", sizeof(long long));

  printf("Size of float       = %d \n", sizeof(float));
  printf("Size of double      = %d \n", sizeof(double));
  printf("Size of long double = %d \n", sizeof(long double));


  return 0;
}
```

Void Data Type

- The type specifier void indicates that **no value** is available.
- It is an **empty data type**
- It can be used in the following situations
  - **Function returns as void**

    A function with no return value has the return type as void.

    Example: void exit (int status);
  - **Function arguments as void**

    A function with no parameter can accept as a void.

    Example: int rand(void);
  - **Pointers to void**

    A pointer of type “void *” represents the address of an object, but not its type. For example a memory allocation function

    Example: void *malloc(size_t size);

    It returns a pointer to void **which can be casted to any data type.**

Type Conversions

- Type conversion: It is a way to convert a variable/constant from one data type to another data type. There are two types:
  a) Implicit type conversion
  b) Explicit Type conversion
- a) Implicit type conversion: Implicit type conversion, also known as **coercion**, is an automatic type conversion by the compiler.

```
#include <stdio.h>
main()
{
   int  i = 10;
   char c = 'a'; /* ascii value is 97 */
   float sum;

   sum = i + c;

 printf("Value of sum : %f\n", sum );
}
```

Output: **107.0**

Type Conversions

- b) Explicit type conversion: Converting the data type of a variable/operand/expression from one data type to another data type explicitly by the programmer using type cast operator.
- Syntax:

  (type_name) expression

```
#include <stdio.h>

main()
{
   int sum = 14, count = 4;
   double mean;

   mean = (double) sum / count;

   printf("Value of mean : %f\n", mean );
}
```

Output: 3.5

<ctype.h>

`isalnum(c)`	`isalpha(c)` or `isdigit(c)` is true
`isalpha(c)`	`isupper(c)` or `islower(c)` is true
`iscntrl(c)`	control character
`isdigit(c)`	decimal digit
`isgraph(c)`	printing character except space
`islower(c)`	lower-case letter
`isprint(c)`	printing character including space
`ispunct(c)`	printing character except space or letter or digit
`isspace(c)`	space, formfeed, newline, carriage return, tab, vertical tab
`isupper(c)`	upper-case letter
`isxdigit(c)`	hexadecimal digit

Type Modifiers

- The modifiers define the amount of storage allocated to the variable.
  - short
  - long
  - signed
  - unsigned
- **Rule**:
  - short int **<=** int **<=** long int float **<=** double **<=** long double
- **Syntax**:
  - **short int x;**
  - **long int x;**
  - **unsigned int x;**
  - **unsigned long int x**

Type Modifiers…

```
int main()
{
  printf("sizeof(char) == %d\n", sizeof(char));
  printf("sizeof(short) == %d\n", sizeof(short));
  printf("sizeof(int) == %d\n", sizeof(int));
  printf("sizeof(long) == %d\n", sizeof(long));
  printf("sizeof(float) == %d\n", sizeof(float));
  printf("sizeof(double) == %d\n", sizeof(double));
  printf("sizeof(long double) == %d\n", sizeof(long double));
  printf("sizeof(long long) == %d\n", sizeof(long long));
  return 0;
}
```

Output: Machine dependent (1, 2, 4, 8, 4, 8, 16, 8)

Escape Sequences

- Combination of characters comprising of backslash followed by a character
- backslash causes an "escape" from the normal way characters are interpreted by the compiler

Escape sequence	Value in hex	Connotation
\a	07	Alarm (Beep, Bell)
\b	08	Backspace
\f	0C	Formfeed
\n	0A	Newline (Line Feed)
\r	0D	Carriage Return
\t	09	Horizontal Tab
\v	0B	Vertical Tab
\\	5C	Backslash
\'	27	Single quotation mark
\"	22	Double quotation mark
\?	3F	Question mark
\0	00	Null (string terminator)
nnn	nnn	Octal representation
\x*hh*	hh	Hexadecimal representation
\u*hhhh*	hhhh	Unicode character

2. (b) Operators

Operators, operands and expressions

- Operators are symbols which take one or more operands or expressions and perform arithmetic or logical computations.

  Examples: +, -, *, /, =, ==

- An "operand" is an entity on which an operator acts.

  Example: a+b-5 → a, b and 5 are operands

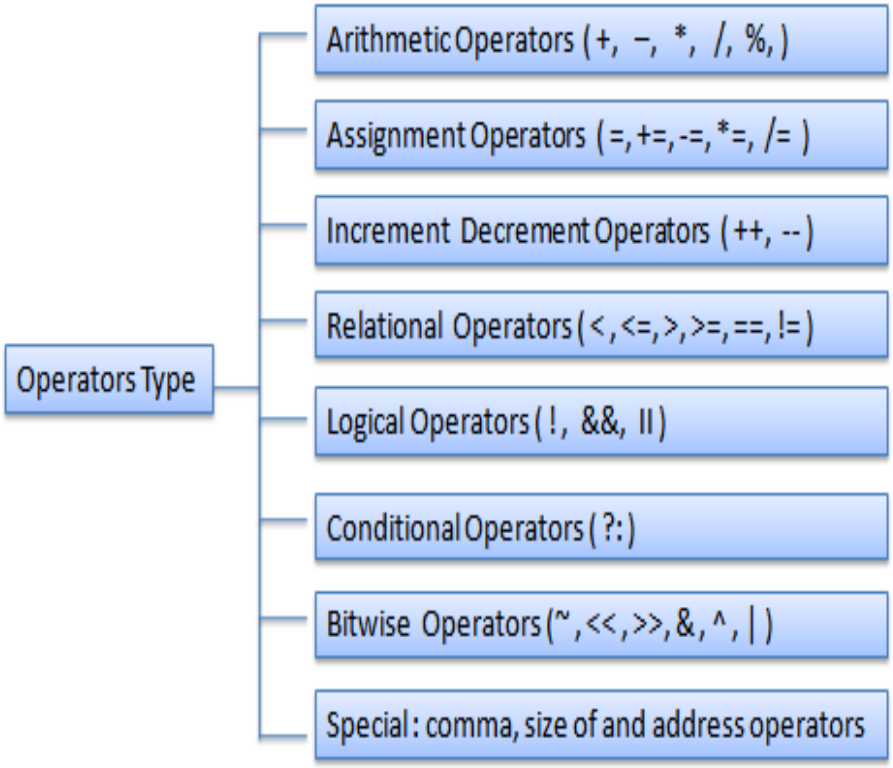
- An "expression" is a sequence of operators and operands that performs any combination of these actions:
  - Computes a value
  - Designates an object or function
  - Generates side effects

  Example: a + (b / 6.0)

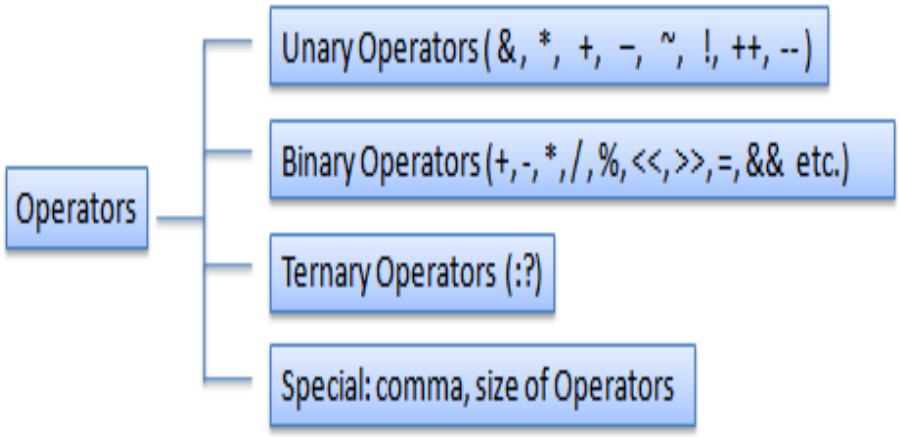
Types of Operators

- Operators are classified in two ways

1. Based on Type of Operation



2. Based on number of operands



Precedence and Order of Evaluation

	Operator	Associativity	Precedence
( )	Function call	Left-to-Right	Highest 14
[ ]	Array subscript		
.	Dot (Member of structure)		
– >	Arrow (Member of structure)		
!	Logical NOT	Right-to-Left	13
~	One's-complement		
–	Unary minus (Negation)		
++	Increment		
– –	Decrement		
&	Address-of		
*	Indirection		
(type)	Cast		
sizeof	Sizeof		
*	Multiplication	Left-to-Right	12
/	Division		
%	Modulus (Remainder)		
+	Addition	Left-to-Right	11
–	Subtraction		
<<	Left-shift	Left-to-Right	10
>>	Right-shift		
<	Less than	Left-to-Right	8
<=	Less than or equal to		
>	Greater than		
>=	Greater than or equal to		
==	Equal to	Left-to-Right	8
! =	Not equal to		
&	Bitwise AND	Left-to-Right	7
^	Bitwise XOR	Left-to-Right	6
|	Bitwise OR	Left-to-Right	5
&&	Logical AND	Left-to-Right	4
||	Logical OR	Left-to-Right	3
? :	Conditional	Right-to-Left	2
=, + = * =, etc.	Assignment operators	Right-to-Left	1
,	Comma	Left-to-Right	Lowest 0

Bitwise Operations

- What is Memory?
  - Collection of Bits
- In real life applications, some times it is necessary to deal with memory bit by bit
- For example,
  - Gaming and Puzzles (Ex: Sudoku)
  - Controlling attached devices (Ex: Printers)
  - Obtaining status
  - Checking buffer overflows…
- Note: The combination of bit level operators and the pointers can replace the assembly code. For example, only 10% of UNIX is written using assembly code and the rest is in C.

Bitwise Operations in Integers

There are six operators

- & – AND
  - Result is **1** if both operand bits are **1**
- | – OR
  - Result is **1** if either operand bit is **1**
- ^ – Exclusive OR
  - Result is **1** if operand bits are different
- ~ – Complement
  - Each bit is reversed
- << – Shift left
  - Multiply by 2
- >> – Shift right
  - Divide by 2

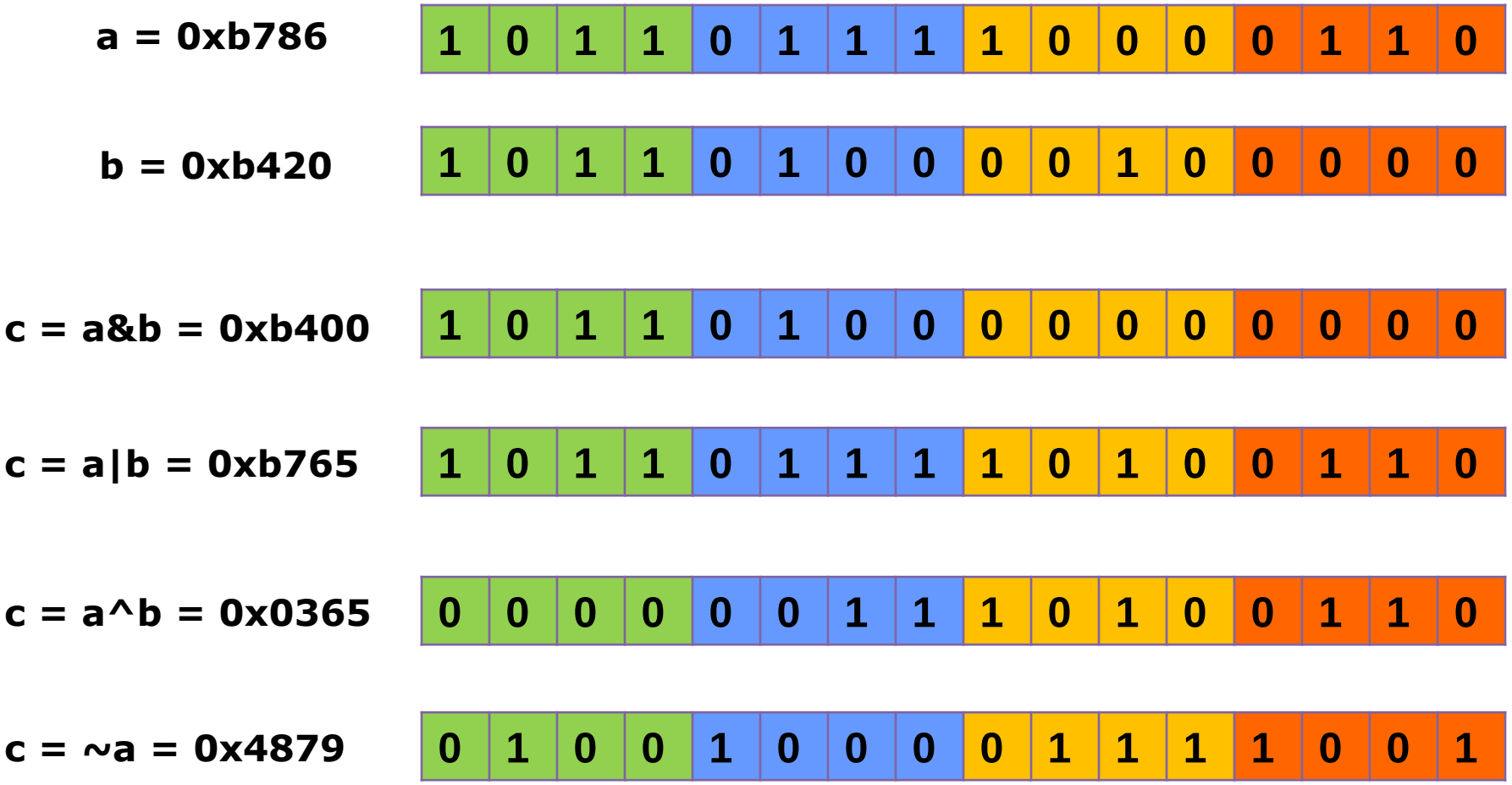
Restrictions: We can use these operators only on int and char data typed variables - Signed and unsigned char, short, int, long, long long

Bitwise Operations in Integers…

	a	0	0	1	1
	b	0	1	0	1
and	a & b	0	0	0	1
or	a | b	0	1	1	1
exclusive or	a ^ b	0	1	1	0
one's complement	~a	1	1	0	0

Examples - &, |, ^ and ~

```
unsigned short int a,b;
unsigned short int c;
```



Example - Left Shift (<<)

```
unsigned short int a = 0xb786;
A << = 3;
```

Before shift

a= 0xb786

After shift

a= 0x9a30

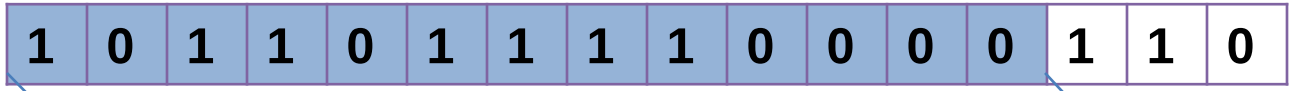
Last three bits are filled with zeroes

Example - Right Shift (<<)

```
unsigned short int a = 0xb786;
A << = 3;
```

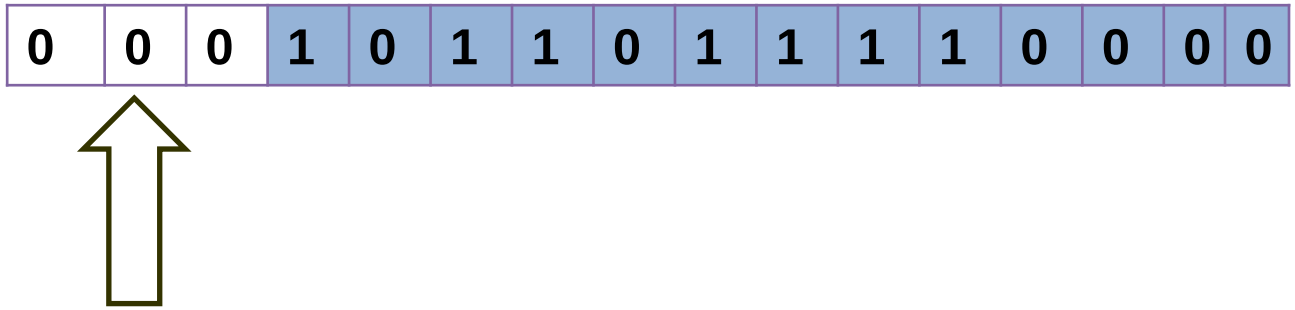
Before shift

a= 0xb786



After shift

a= 0x16f0



First three bits are filled with zeroes

Quiz

1. What do you mean by Binary Operators?
2. What is an unary operator?
3. If x=10 and y=2, what is the value of -x * y++ ?

Quiz 1

1. **What is the output of this code?**

```
#include <stdio.h>
int main()
{
        int x = 1, y = 0, z = 5;
        int a = x && y || z++;
        printf("%d \n", a); return 0;
}
```

(a) Syntax Error
(b) 6
(c) 1
(d) 5

Answers: C (z++=5, 0 || 5 = 1)

Quiz …

2. What is the output of this code?

```
#include <stdio.h>
void main()
{
        int x = 0, y = 2, z = 3;
        int a = x & y | z;
        printf("%d \n", a);
}
```

(a) Syntax Error
(b) 6
(c) 3
(d) 0

Answers: c (0 & 2 | 3 = 00 & 10 | 11 = 10 | 11 = 11 = 3 (L→R)

Quiz 3

3a and 3b. What is the output of this code?:wq

```
#include <stdio.h>
    void main()
    {
        int x = 5 * 9 / 3 + 9;
            printf(“%d \n”x);
    }
```

```
#include <stdio.h>
int main()
{
        int x = 10, y = 20, z = 5;
        printf("%d\n", x+y*z);
        return 0;
```

(a) 3
(b) 24
(c) 60
(d) None

Answers: b

Quiz 4…

6. What is the output of this code?

```
#include<stdio.h>
int main()
{
   int n=6;
(n%2)?printf("Odd\n"):printf("Even\n");
return 0;
}
```

(a) Syntax Error
(b) Odd
(c) Even
(d) 0

Quiz 5…

6. **What is the output of this code?**

```
#include<stdio.h>
int main()
{
int i;
int x=10, y=2;

i=-x*y++;
printf("i..%d\n", i);
printf("x..%d\n", x);
printf("y..%d\n", y);

i=-x*++y;
printf("i..%d\n", i);
printf("x..%d\n", x);
printf("y..%d\n", y);

return 0;
}
```

Class Room Work

1. Write a program that asks the user to enter two numbers, obtain the two numbers from the user and prints the sum, product, difference, quotient and remainder of the two numbers.
2. Write a program to convert a positive integer (2 digits) to binary.

THANK YOU