EST 102 Programming in C

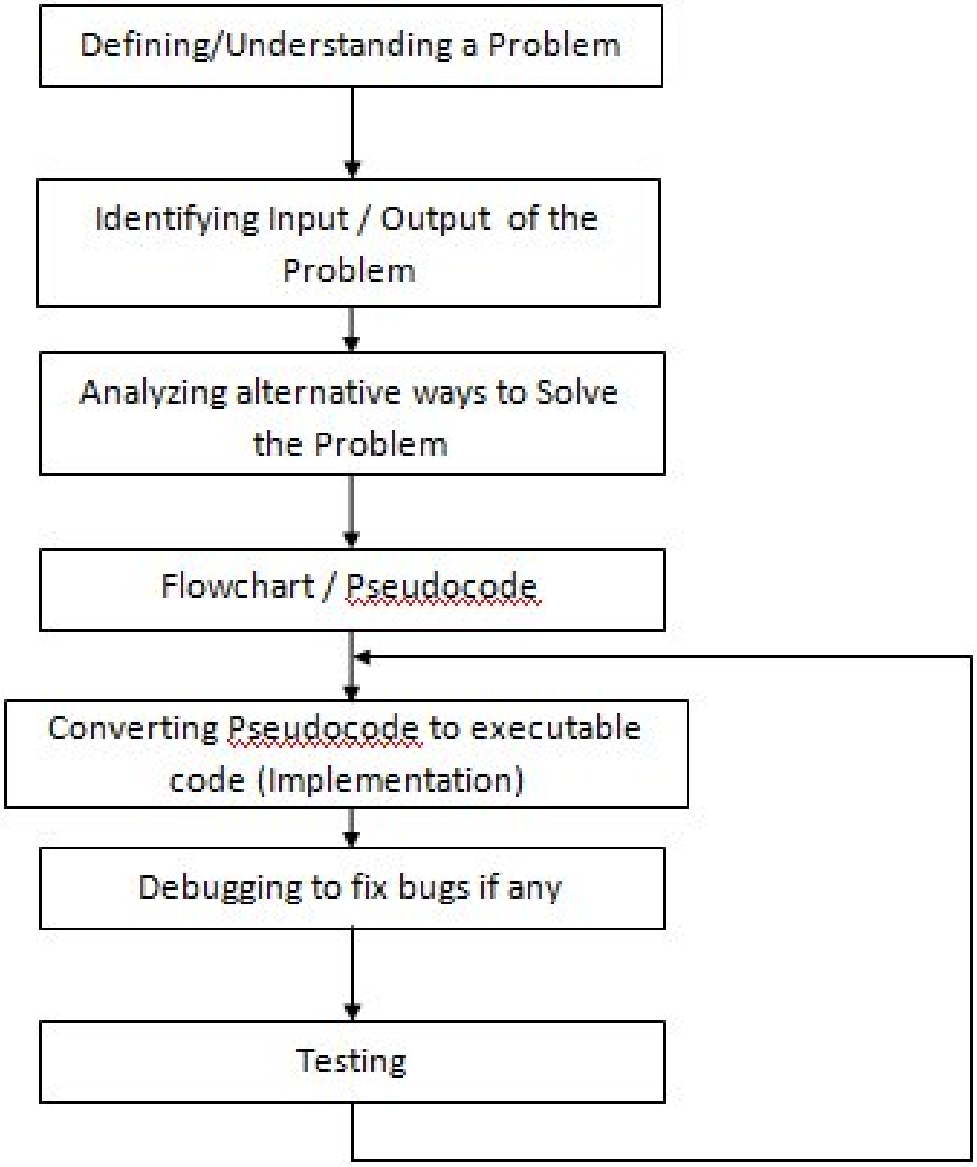
Module 1

Part – I
C Fundamentals

C Fundamentals - Syllabus

- **Character Set, Constants, Identifiers, Keywords**
- **Basic Data types**
- **Variables**
- Operators and its precedence
- Bit-wise operators
- Expressions
- **Statements - Input and Output statements**
- **Structure of a C program**
- Simple programs.

Problem Solving Steps – A Recap



History of C

- Born at AT & T Bell Laboratory of USA in 1972
- Many of C's principles and ideas were derived from the earlier language B
- Ken Thompson was the developer of B Language
- C was written by Dennis Ritchie

  C language was created for a specific purpose i.e., designing the UNIX operating system

About Dennis Ritchie

- Born September 9, 1941
- Developed C language which is widely used developing, operating systems, compiler, and embedded system development, Assemblers, Text editors
- Died on October 12, 2011

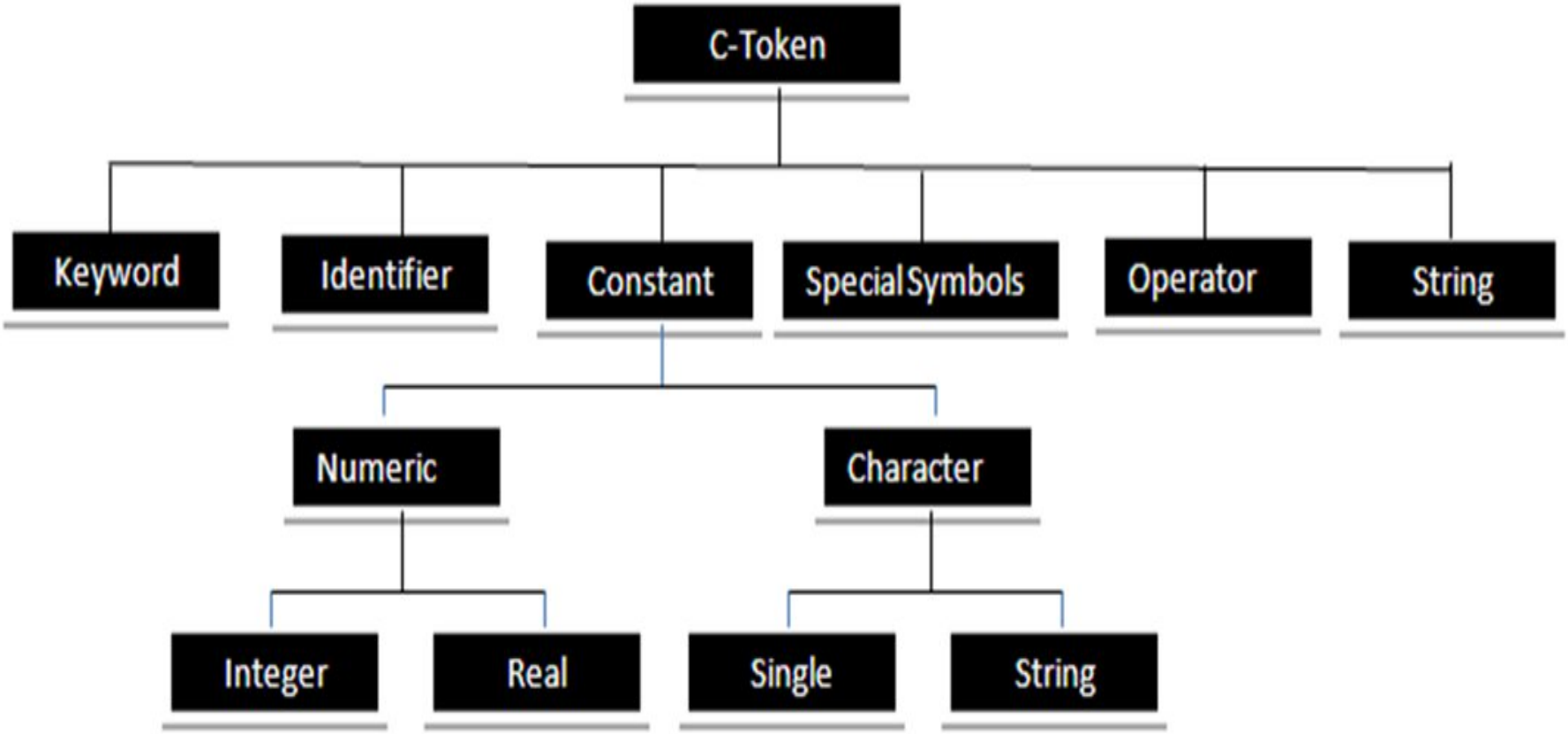
Character set of C

- The C language alphabet
  - Uppercase letters ‘A’ to ‘Z’
  - Lowercase letters ‘a’ to ‘z’
  - Digits ‘0’ to ‘9’
  - Certain special characters:

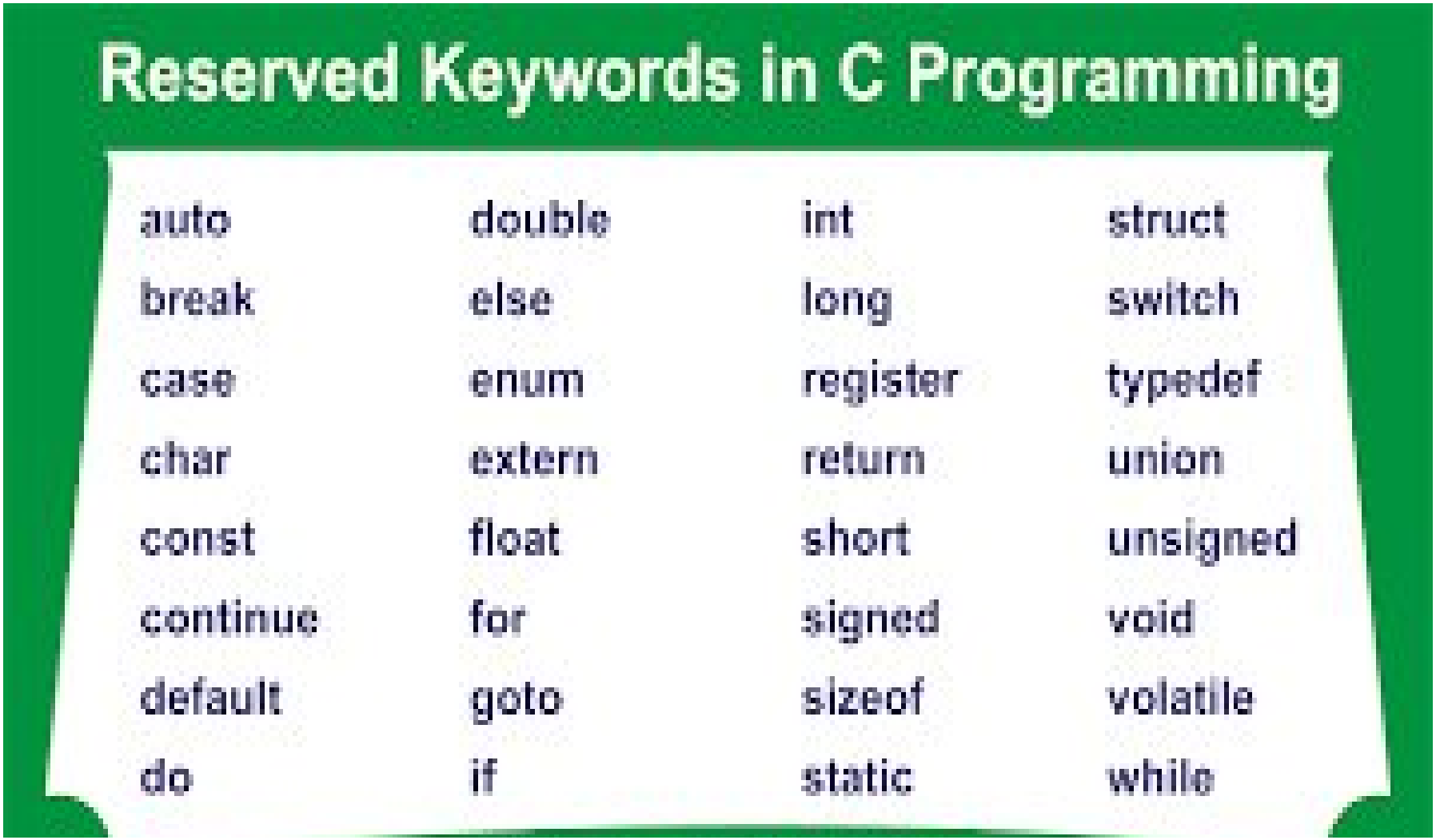
!	#	%	^	&	*	(	)
-	_	+	=	~	[	]	\
|	;	:	‘	“	{	}	,
.	<	>	/	?	blank		

Tokens

They are the smallest individual units or components of a C program.



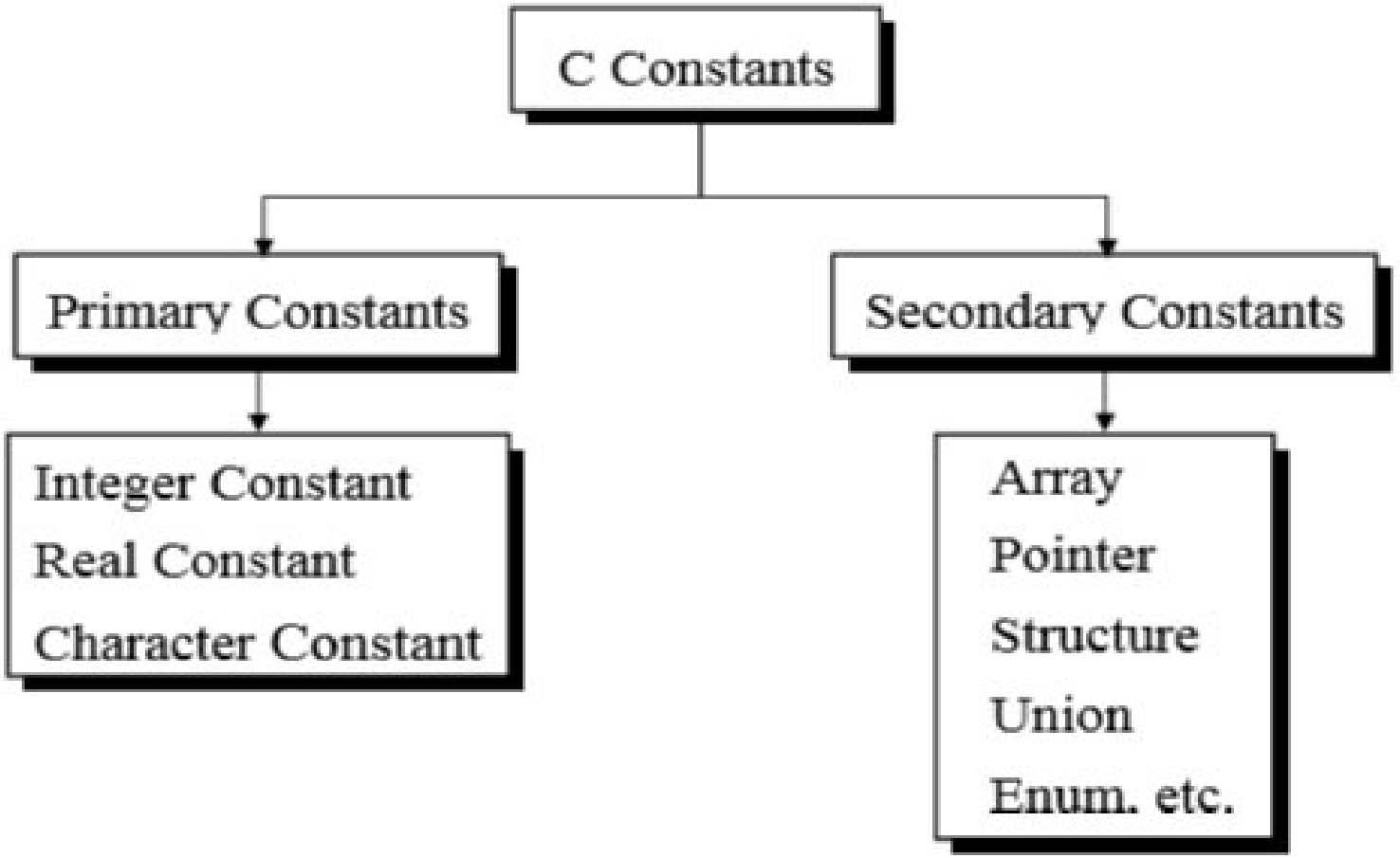
Keywords


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

Constants, Variables and Keywords

- alphabets, numbers and special symbols when properly combined - form constants, variables and keywords
- **Constant** - entity that doesn't change
- **Variable** - entity that may change
- **Keyword** – reserved words in programming language

Types of C Constants



Integer Constants

Representation	Value	Type
+123	123	int
–378	–378	int
–32271L	–32,271	long int
76542LU	76,542	unsigned long int
12789845LL	12,789,845	long long int

Rules for Exponential Form Real Constants

- Both parts should have atleast one digit
- Mantissa can be a real constant in fractional form
- Exponent can only be integers
- Both parts may have a positive or negative sign
- Default sign of both parts is positive
- Range of real constants expressed in exponential form is -3.4e38 to 3.4e38

Eg: +3.2e-5 4.1e8 -0.2e+3 -3.2e-5

Double constants

Valid double constants	Invalid double constants
3.14159	0.12345e
0.005	15e-0.3
15.0e-4	34,500.99
2.34e2	
12.0e+5	

Character Constants

- A character constant is a single alphabet, a single digit or a single special symbol enclosed within single inverted commas
- Maximum length of a character constant can be 1 character

  Eg: 'A' 'I' '5' '='

Note:
1. All escape sequences are also considered characters.
2. The character constant ‘7’ is not the same as digit 7.

Character Constants - Escape Sequence

Escape Sequence	Meaning
\b	Backspace
\n	New Line
\r	Carriage Return
\t	Horizontal Tab
\\	Backslash
\'	Single Quote
\"	Double Quote
\a	Alert sound

15

ASCII

American Code for Information Interchange:

- a standard data-encoding format for electronic communication between computers. ASCII assigns standard numeric values to letters, numerals, punctuation marks, and other characters used in computers.
- This code is basically used for identifying characters and numerals in a keyboard.

Character	Binary	Decimal	Hexadecimal
A	0100 0001	65	0x41
a	0110 0001	97	0x61

The ASCII Character Set

ASCII Value	Character	ASCII Value	Character	ASCII Value	Character	ASCII Value	Character
0	NUL	32	(blank)	64	@	96	`
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	'	71	G	103	g
8	BS	40	(	72	H	104	h
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	ETB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	\	124	|
29	GS	61	=	93	]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	_	127	DEL

Single character constant

- Character constants have integer value known as ASCII values.

Statement	Output
`printf(“a”)`	a
`printf(“%c”,‘a’);`	a
`printf(“%c”,97);`	a
`printf("97");`	97
`printf("%d", 97);`	97
`printf(“%d”,‘a’);`	97

String Constants

- A string constant is a single character or a group of characters enclosed in double quotes (“ ”).

  Eg: “hello”
  “4598”
  “hello235”
  “5+3”

Identifiers

- Identifiers are a string of characters that serves as the name of a variable, function, array, pointer, structure, etc.
- An identifier can be described as the user-specified name used to identify a specific item.

Rules for identifiers

- It cannot be a reserved keyword.
- It should not contain white space.
- It can be a combination of A-Z, a-z, 0-9, or underscore.
- It should start with an alphabet character or an underscore (_).
- It should not contain any special character other than an underscore (_).

More Valid and Invalid Identifiers

Valid identifiers	Invalid identifiers
X	10abc
abc	my-name
simple_interest	“hello”
a123	simple interest
LIST	(area)
stud_name	%rate
Empl_1	Tinu’s
avg_empl_salary	

Variables

- The memory is used to store the input data and result while executing a program
- Can have only one value assigned to it at any given time during the execution of the program
- The value of a variable can be changed during the execution of the program
- A variable is mapped to a location of the memory, called its address

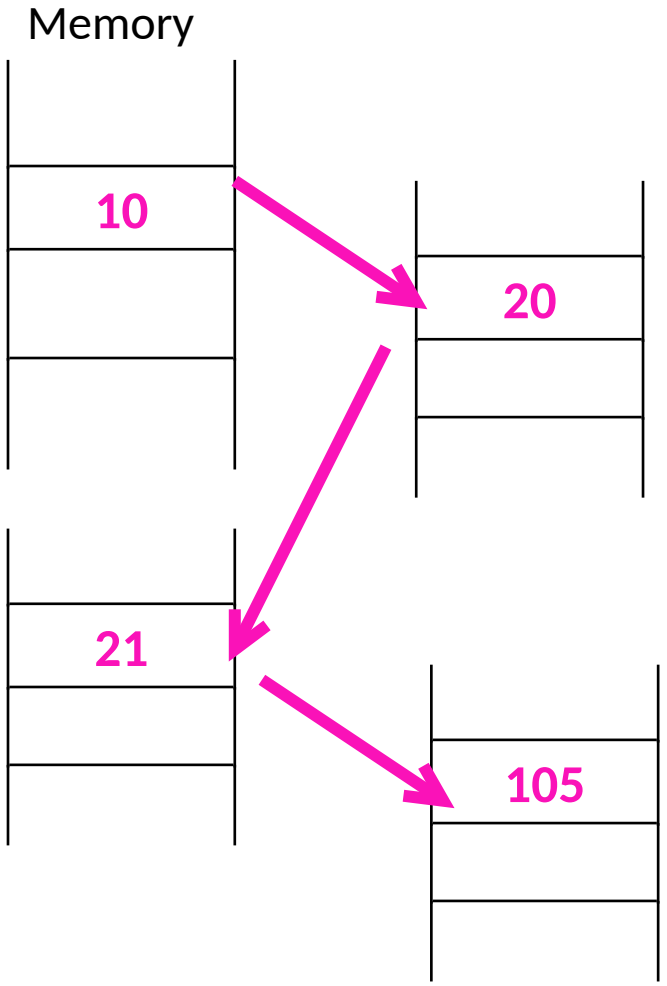
Rules for Constructing Variable Names

- Rules for constructing variable names of all types are same.
- Combination of alphabets, digits or underscores
- Some compilers allow up to 247 characters but safer to 31 characters.
- First character must be an alphabet or underscore.
- No commas or blanks are allowed.
- No special symbol other than an underscore.

Eg: num_int Salary_staff sales_Jan_24

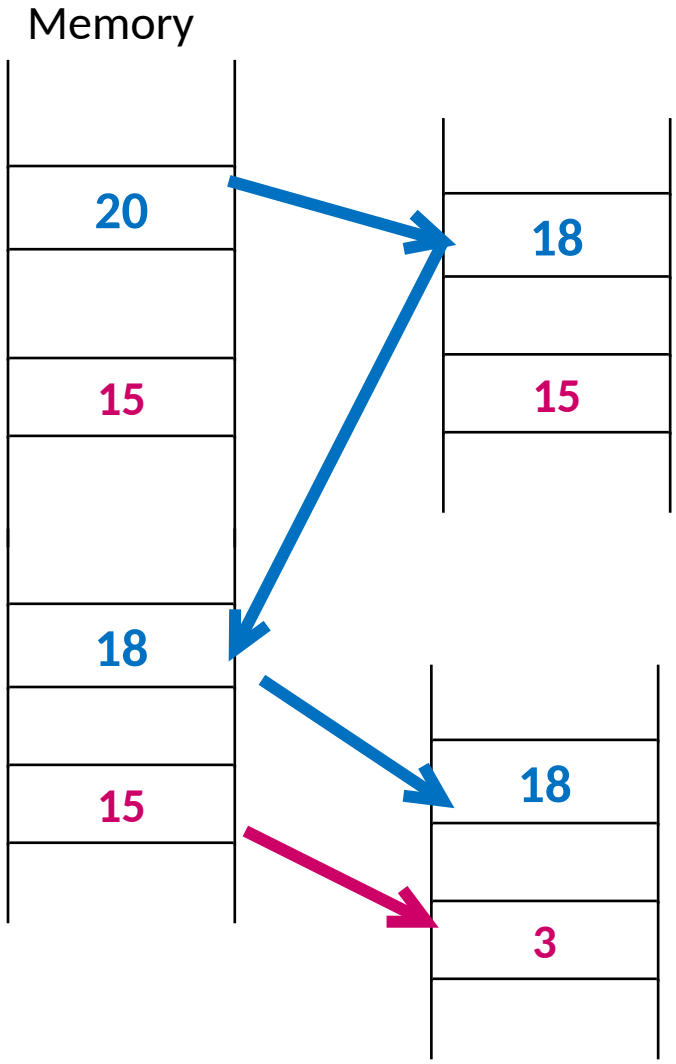
Variable changes in memory

```
1.X = 10

2.X = 20

3.X = X+1

4.X = X*5
```



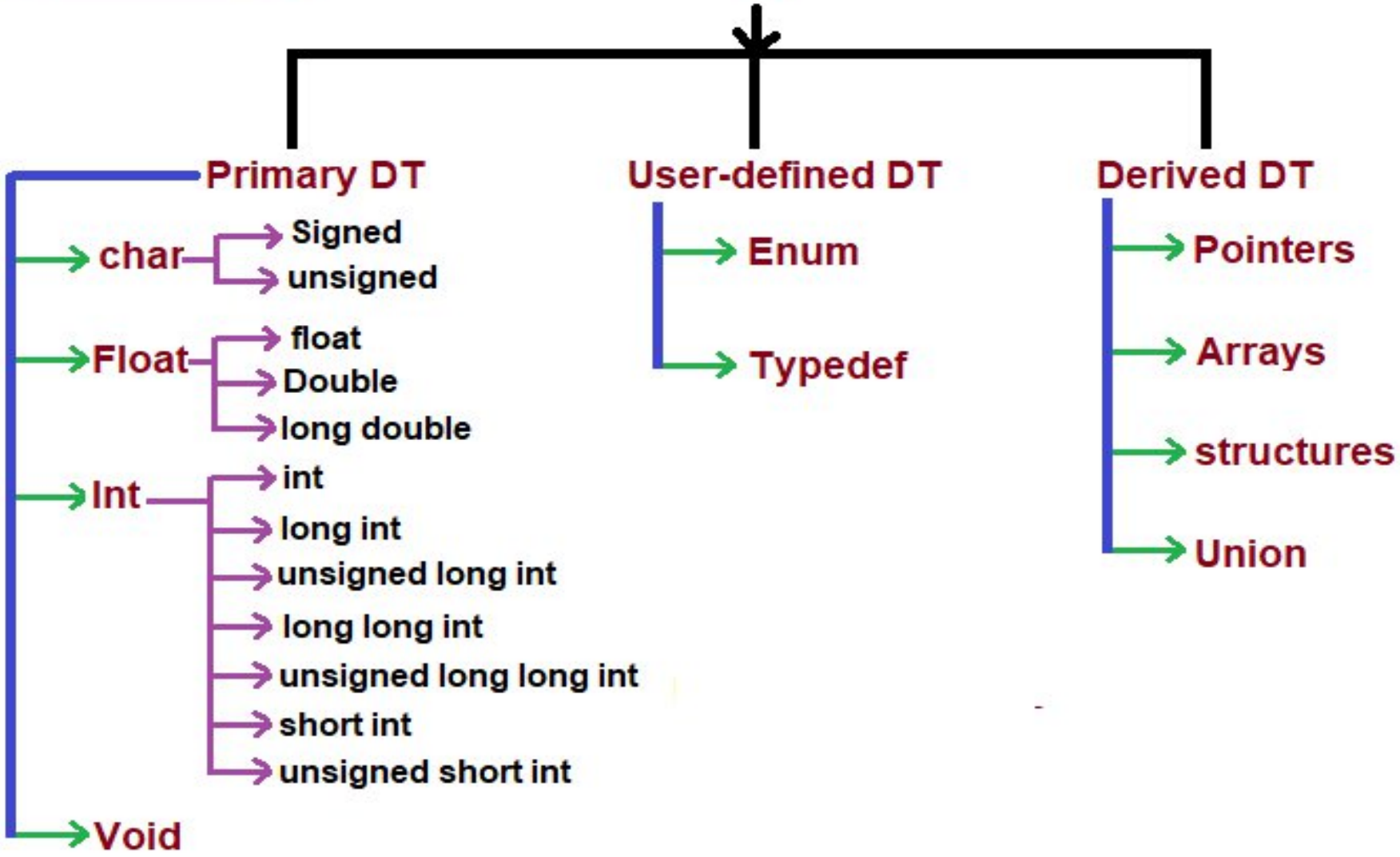
Variable changes in memory

```
1.X = 20

2.Y = 15

3.X = Y+3

4.Y = X/6
```



DT - Data type

Data Types in C

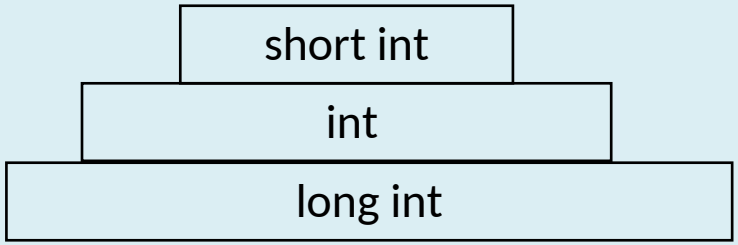


Primary data types

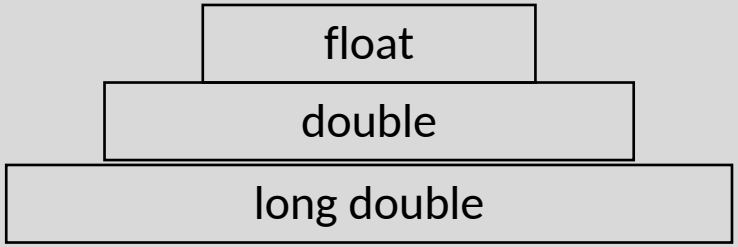
S. No	Keyword	Size in bits	Size in Bytes	Format control string
1	char	8	1	%c
2	signed char			
3	unsigned char			
4	int	16	2	%d
5	signed int			
6	short int	8	1	%h
7	signed short int			
8	long int	32	4	%l
9	signed long int			
10	unsigned int	16	2	%u
11	unsigned short int	8	1	%hu
12	unsigned long int	32	4	%lu
13	float	32	4	%f
14	double	64	8	%lf
15	long double	80	10	%Lf
16	void	0	0	

Storage of int and float

Storage of short to long int



Storage of float to long double



Binary to decimal conversion

Convert 11111111 Binary number in to Decimal Number

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
1	1	1	1	1	1	1	1

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

= 255

Binary to decimal conversion

Convert From Binary to Decimal

```
128 64 32 16  8  4  2  1 weight
 0  0  1  0  0  1  1  0
```

32 + 4 + 2 = 38 decimal

```
128 64 32 16  8  4  2  1 weight
 1  1  0  0  1  1  0  1
```

128 + 64 + 8 + 4 + 1 = 205 decimal

Powers of 2

$2^0 = 1$
$2^1 = 2$
$2^2 = 4$
$2^3 = 8$
$2^4 = 16$
$2^5 = 32$
$2^6 = 64$
$2^7 = 128$
$2^8 = 256$

Integer Data types and memory storage

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 bytes	-32,768 to 32,767
unsigned int	2 bytes	0 to 65,535
short	1 byte	-128 to 127
unsigned short	1 byte	0 to 255
long	4bytes	-2147483648 to 2147483647
unsigned long	4 bytes	0 to 4294967295

*Note : These are data storage on a 16 bit machine

Floating Point Date types and storage

Type	Storage size	Value range
float	4 byte	3.4E-38 to 3.4E+38
double	8 byte	1.7E-308 to 1.7E+308
long double	10 byte	3.4E-4932 to 1.1E+4932

*Note : These are data storage on a 16 bit machine

Write a program to accept an integer, float and character and display each

```
#include <stdio.h>
int main()
{
    int a;
    float b;
    char c;
    a = 2;
    b = 3.4;
    c = 'Y';
    printf("a= %d\t",a );
    printf("b= %f\t",b);
    printf("c= %c\t",c );
    return 0;
}
```

Output:

a = 2 b = 3.4000 c = Y

Invalid format specifier or control string

```
#include<stdio.h>
int main()
{
   printf("%d" , 9.7);
   return 0;
}
```

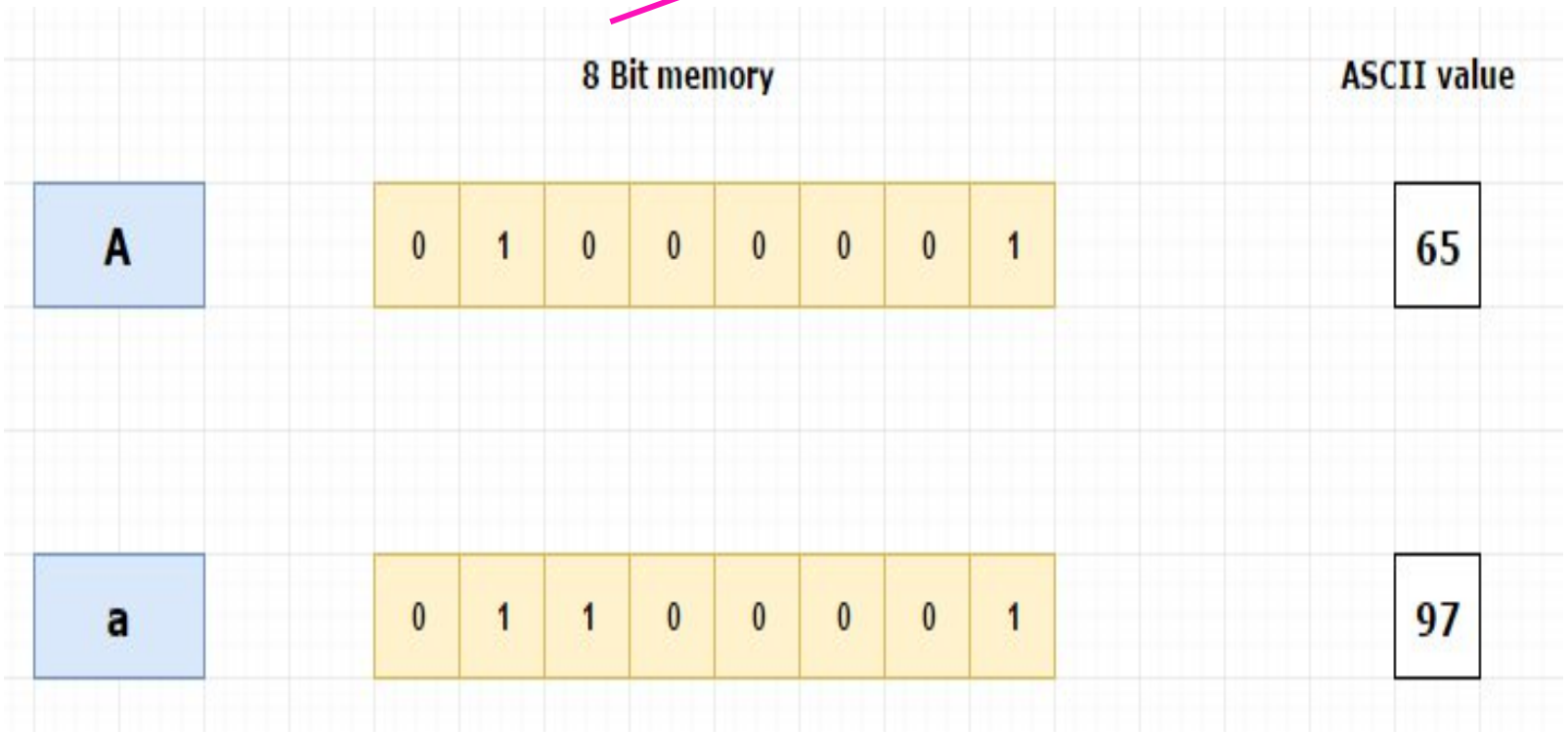
Warning: format '%d' expects argument of type 'int', but argument 2 has type 'float'

Invalid format specifier or control string

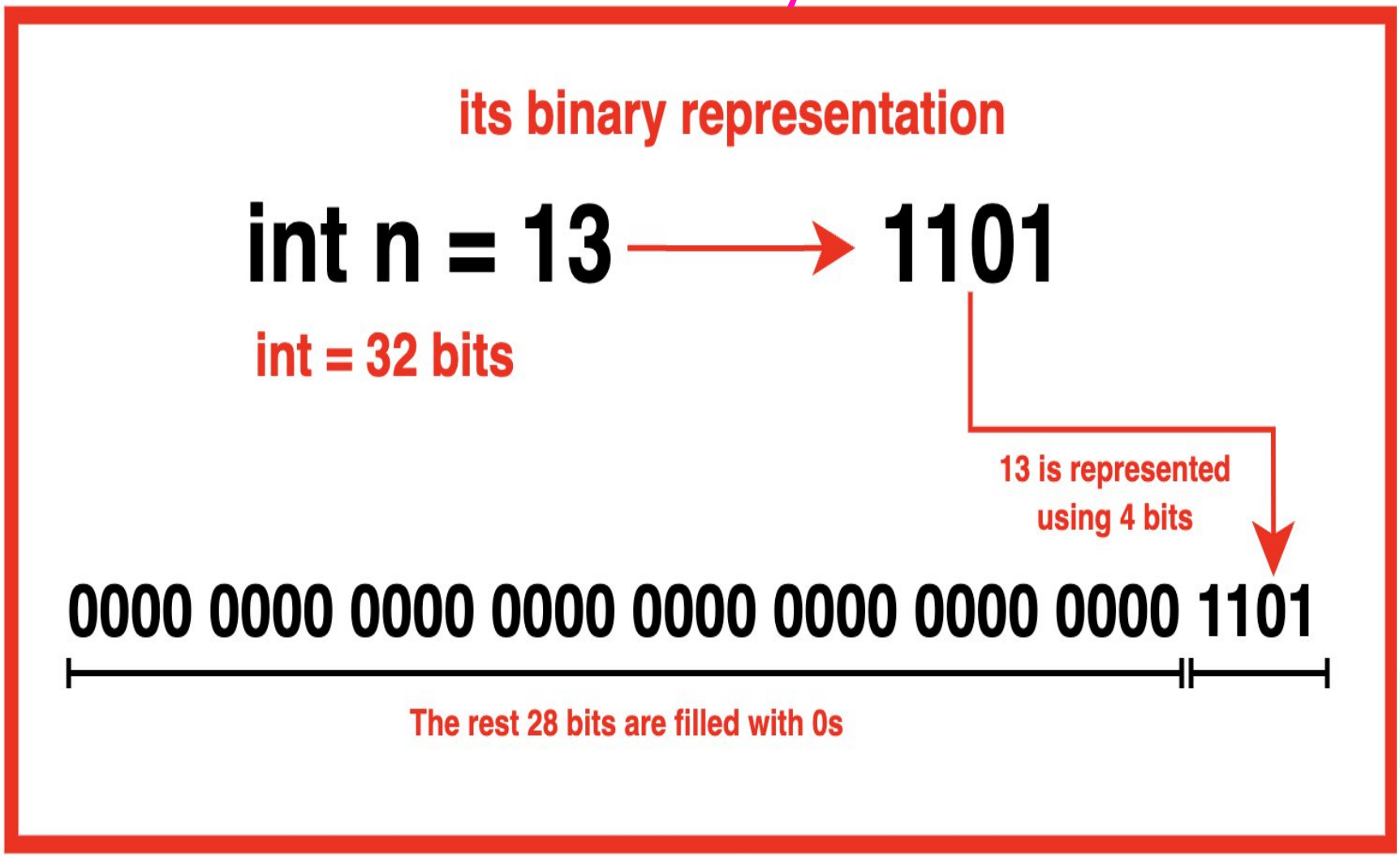
```
#include<stdio.h>
int main()
{
   printf("%f", 9);
   return 0;
}
```

Warning: format '%f' expects argument of type 'float', but argument 2 has type 'int'

How are characters stored in memory?



How are integers stored in **32 bit** memory?



How are real numbers stored in **32 bit** memory?

$$1.2345 = \underbrace{12345}_{\text{mantissa}} \times \overbrace{10^{-4}}^{\text{exponent}}$$

Sign bit
0 (+)
1 (-)

32-bit Single-Precision Floating-point Number

Derived data type

- Array
- Structure
- Union
- Pointers

Array: It contains homogeneous values

Example:

```
int marks[50];
          // contains only integer values //
char color[3];
          //contains only character values//
int matrix[3][4];
```

Derived data type contd

- Array
- Structure
- Union
- Pointers

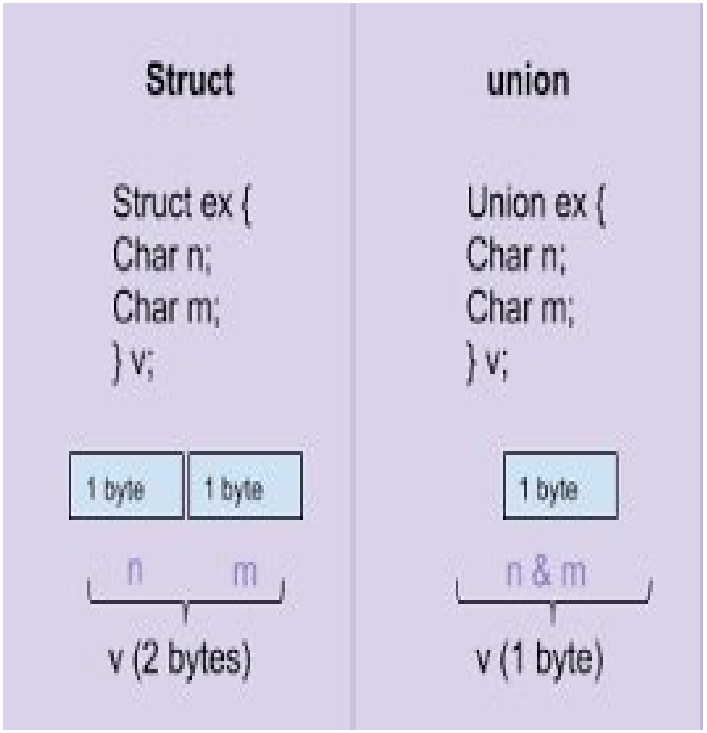
Structure: It contains collection of elements of different data types

Example:

```
struct
        {
            int account_num;
            char name[20];
            float balance;
        }old_customer, new_customer;
```

Derived data type- Union

Union: similar to structure, but share same storage area.



Structure

```
struct Employee{
char x; // size 1 byte
int y; //size 2 byte
float z; //size 4 byte
}e1; //size of e1 = 7 byte
```

size of e1= 1 + 2 + 4 = 7

Union

```
union Employee{
char x; // size 1 byte
int y; //size 2 byte
float z; //size 4 byte
}e1; //size of e1 = 4 byte
```

size of e1= 4 (maximum size of 1 element)

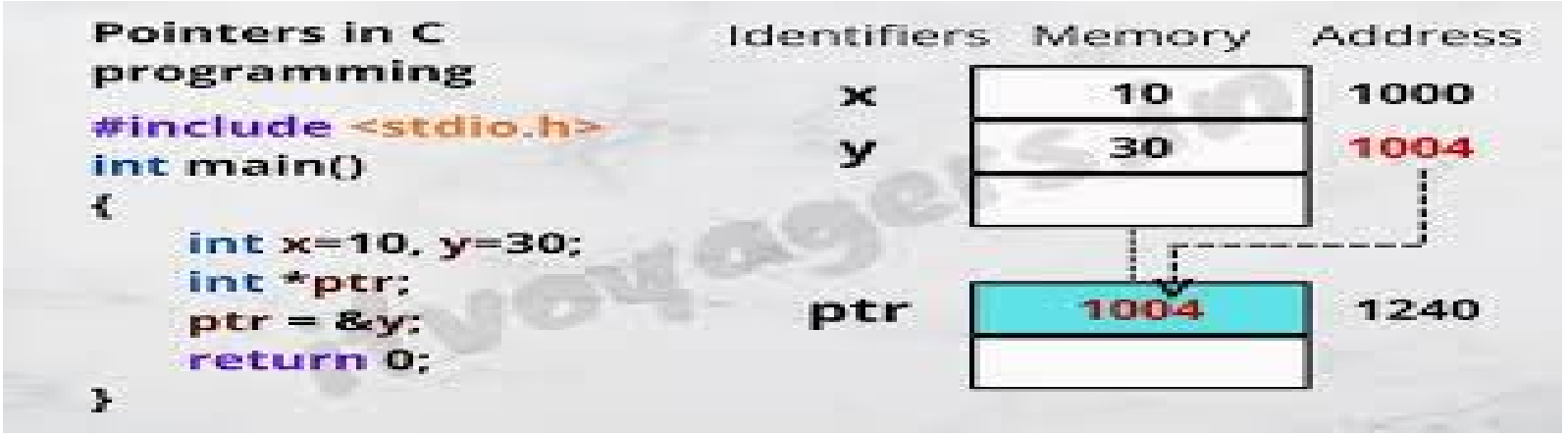
Derived data type- Pointers

- Pointers provide a means of accessing the variables stored in memory.
- A pointer variable holds the address of another variable

Declaration: `datatype *pointer_variable`

Example: `int *p;`

`p` is a pointer that points to a integer data



User Defined Data types

- enum (enumeration)
- typedef

enum

An enumeration is a datatype, similar to structure and union.

Its members are constants that are written as identifiers.

Declaration syntax

enum tag {member1, member2, member3, member4};

Example

enum colors { red, blue, green}

Enumerations are automatically assigned integer values.

red = 0 blue = 1 green = 2

User Defined Data types - enum

```
#include<stdio.h>
enum week{Mon, Tue, Wed, Thur, Fri, Sat, Sun};
int main()
{
    enum week day;
    day = Wed;
    printf("%d",day);
    return 0;
}
```

Output: 2

```
**************************************************
#include <stdio.h>
enum day {Sun = 1, Mon, Tue = 5, Wed, Thur = 10, Fri, Sat};
int main()
{
   printf("%d %d %d %d %d %d %d", Sun,Mon,Tue,Wed,Thur,Fri,Sat);
   return 0;
}
```

Output:
1 2 5 6 10 11 12

User Defined Data types - typedef

- typedef allows the user to define new data types that are equivalent to existing data type

Declaration: `typedef datatype new_type`

Example:

```
typedef int id_num;
typedef float weight;
```

Here `id_num` is new datatype name given to `int`, while `weight` is new data type given to `float`.

In the statement

```
id_num  Adam, Ajay;
weight apples, mangoes;
```

Adam and Ajay are declared as `int` data type

Console IO operations

printf(): Output can be written from computer to an output device using `printf` library function

Syntax:

```
printf("Format Specifier", var_1,
                    var_2, …., var_n);
```

Example:

```
printf("%d", a);
printf("%d %d %f", a, b, average)
```

Note : There is NO COMMA after %d or %f in scanf and printf

`printf`- syntax

- **print f**ormatted, shortened as **printf** is the main function used in printing formatted output to a terminal in C language.

Format/Conversion Specifiers

- All format specifiers are written as the percent (%) sign in front of data types).

`printf`(“ <format string>” , Argument list);

Program Expt – 1 i) Hello world

Write a C program to display Hello World.

```
#include <stdio.h>

int main()

{
    printf("Hello World \n");
    return 0;
}
```

Preprocessor directives

Function main

Output :
Hello World

Why `return 0` ?

- In general, `return` statement transfers control from the function back to the activator of the function
- The last line in **`main`** function - `return(0)` transfers control from our program to the operating system
- The value `0` in parenthesis is considered the result of the function-main's execution and it indicates that our program is executed without error!

Program – Sum of 2 numbers

Write a C program to find the sum of two given numbers

```
#include <stdio.h>
int main()
{
   int a, b, sum;
   a = 2;
   b = 7;
   sum = a + b;
   printf("The sum of %d and %d is : %d",a, b,sum);
   return 0;
}
```

Variable declaration

Variable initialization

Output

The sum of 2 and 7 is : 9

Console IO operations

scanf(): Values are assigned to variables through keyboard input using scanf library function

Format:

```
scanf("Format Specifier", &var_1,
                  &var_2, …., &var_n);
```

Example:

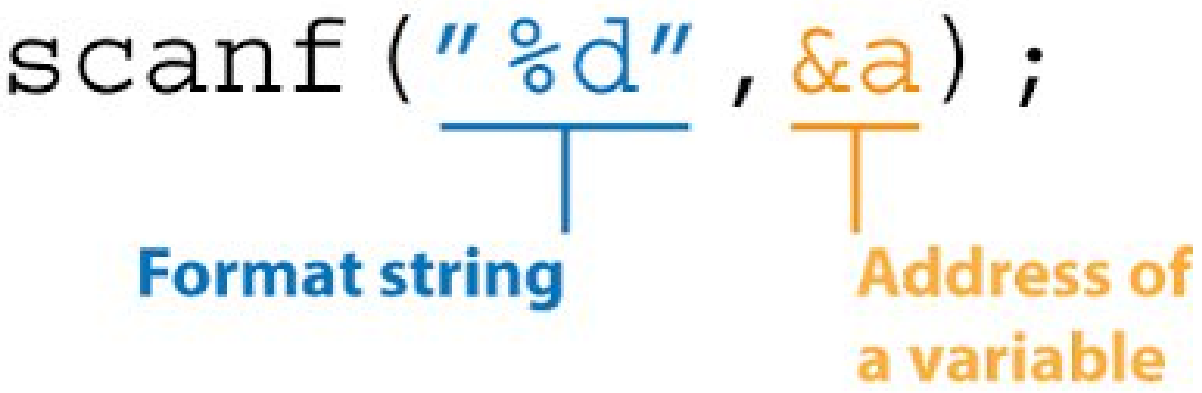
```
scanf("%d",&a);
scanf("%d %f %c",&mark,&percent,&grade)
```

Note : There is NO COMMA after %d or %f in scanf and printf

scanf - syntax

The scanf function reads data from the console, **stores** the value in the ***memory address*** of the variable provided.

`scanf("<format specifier>" , <address of variable>);`



Placeholder

- A place holder is present in format string of printf and scanf
- It begins with %
- It represents the type of value that will be entered by the user
- It marks the display position for the variable

Placeholder	Variable type
%c	char
%d	int
%f	float
%lf	double

Expt 1 ii) Sum of 2 numbers

Accept input from user, one at a time!

```
#include <stdio.h>
int main()
{
   int a, b, sum;
   printf("Enter the value of a : ");
   scanf("%d", &a);
   printf("Enter the value of b : ");
   scanf("%d", &b);
   sum = a + b;
   printf("The sum of %d and %d is : %d",a, b,sum);
   return 0;
}
```

The ampersand (**&**) is the C **Address Operator**; it returns the memory address of the given variable.

Expt 1 ii) Sum of 2 numbers contd

Accept both input from user, simultaneously

```
#include <stdio.h>
int main()
{
   int a, b, sum;
   printf("Enter any two numbers : ");
   scanf("%d %d", &a, &b);
   sum = a +  b;
   printf("The sum of %d and %d is : %d",a, b,sum);
   return 0;
}
```

Note: There is no comma in between the format string

Program – area of circle

```
#include<stdio.h>
#define PI 3.14
int main()
{
 int radius;
 float area;
 printf("Enter the radius of the circle:= ");
 scanf("%d",&radius);
 area = PI*radius*radius;
 printf("\n The area of a circle is :%f sq.cm ",area);
 return 0;
}
```

Output

Enter the radius of the circle:= 5

The area of the circle is : 78.500000 sq.cm

Printf, scanf - formats

Placeholder	Used For
%c	a single character
%s	a string
%hi	short (signed)
%hu	short (unsigned)
%Lf	long double
%d	a decimal integer
%p	an address (or pointer)
%f	a floating point number for floats
%u	int unsigned decimal

C Header files

Header Files	Description	Functions
stdio.h	Standard input/output functions	scanf(), printf(), fopen()
math.h	Maths functions	sin(), cos(), pow(), sqrt()
string.h	String functions	strcpy(), strlen(), strcat()
ctype.h	Character handling functions	isalpha(), isupper(), ispunct()
conio.h	console input-output	clrscr(), getch(), putch(), cgets(), cputs()

Some mathematical library functions

Function	Standard Header File	Purpose: Example	Argument(s)	Result
`abs(x)`	`<stdlib.h>`	Returns the absolute value of its integer argument: if `x` is `-5`, `abs(x)` is 5	`int`	`int`
`ceil(x)`	`<math.h>`	Returns the smallest integral value that is not less than `x`: if `x` is `45.23`, `ceil(x)` is `46.0`	`double`	`double`
`cos(x)`	`<math.h>`	Returns the cosine of angle `x`: if `x` is `0.0`, `cos(x)` is `1.0`	`double` (radians)	`double`
`exp(x)`	`<math.h>`	Returns e^x where e = `2.71828...`: if `x` is `1.0`, `exp(x)` is `2.71828`	`double`	`double`
`fabs(x)`	`<math.h>`	Returns the absolute value of its type `double` argument: if `x` is `-8.432`, `fabs(x)` is `8.432`	`double`	`double`
`floor(x)`	`<math.h>`	Returns the largest integral value that is not greater than `x`: if `x` is `45.23`, `floor(x)` is `45.0`	`double`	`double`
`log(x)`	`<math.h>`	Returns the natural logarithm of `x` for `x > 0.0`: if `x` is `2.71828`, `log(x)` is `1.0`	`double`	`double`

Some mathematical library functions

`log10(x)`	`<math.h>`	Returns the base-10 logarithm of `x` for `x > 0.0`; if `x` is `100.0`, `log10(x)` is `2.0`	`double`	`double`
`pow(x, y)`	`<math.h>`	Returns x^y. If `x` is negative, `y` must be integral; if `x` is `0.16` and `y` is `0.5`, `pow(x,y)` is `0.4`	`double`, `double`	`double`
`sin(x)`	`<math.h>`	Returns the sine of angle `x`; if `x` is `1.5708`, `sin(x)` is `1.0`	`double` (radians)	`double`
`sqrt(x)`	`<math.h>`	Returns the nonnegative square root of `x` ($\sqrt{x}$) for `x` $\geq$ `0.0`; if `x` is `2.25`, `sqrt(x)` is `1.5`	`double`	`double`
`tan(x)`	`<math.h>`	Returns the tangent of angle `x`; if `x` is `0.0`, `tan(x)` is `0.0`	`double` (radians)	`double`