


# Basic Program Structure and Program Development

**Lecture 2 – ICT1132**

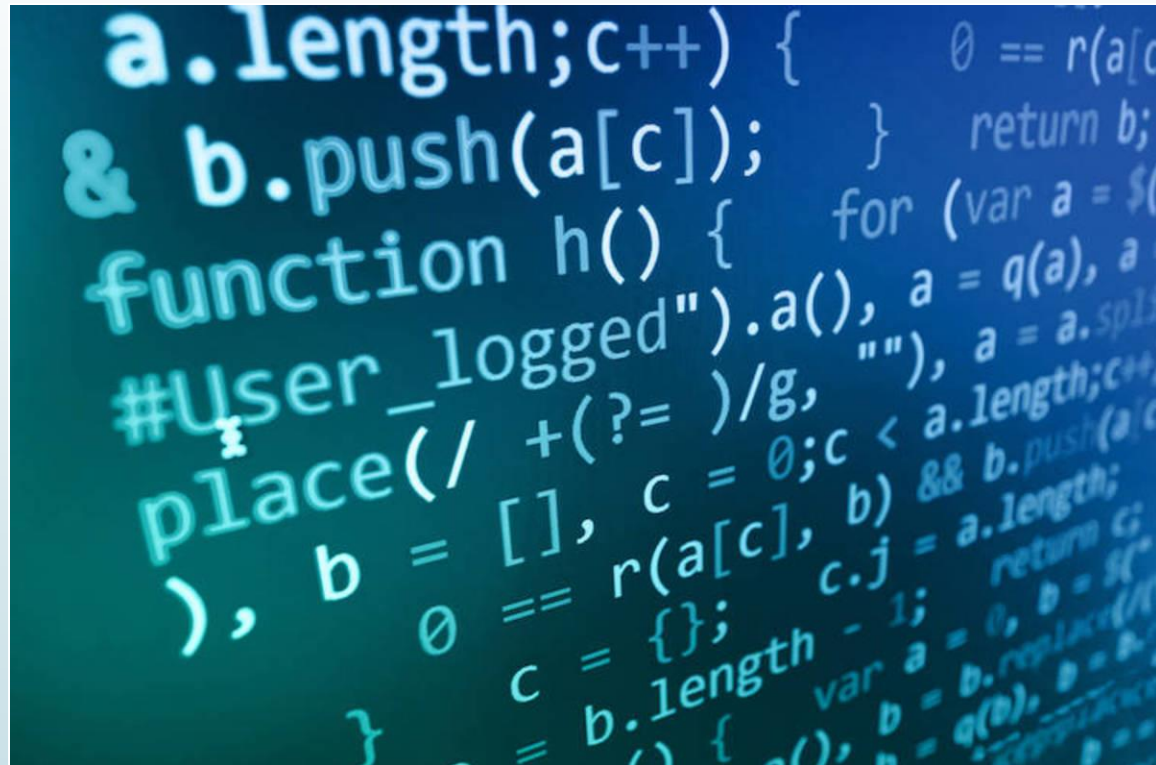
**Piyumi Wijerathna**  
Department of ICT  
Faculty of Technology



# Overview

- What is Programming?
  - What is IDE?
  - Variables
  - Data types
  - Formatted Console Input and Output
  - Type Conversion
  - Errors
- 

# Computer Programming



```
a.length; c++) {  
& b.push(a[c]);  
function h() {  
#User_logged".a(), a = q(a), a  
place(/ +(?= )/g, ""), a = a.split  
, b = [], c = 0; c < a.length; c++)  
    0 == r(a[c], b) && b.push(a[c])  
    c = {}; c.j = a.length;  
    return c;  
    var a = 0, b = {}  
    b = b.replace(/  
    b = q(b), b = q(b)  
    }
```



## **What is a Computer Program?**

- A set of instructions which are written in a computer understandable format.

## **What is a Computer Programming Language?**

- An artificial language which can be used to control the behaviors of computer systems.

# What is Programming?

- **Creating a set of instructions for completing some specific tasks.**
- In the context of computing,
  - Creating a set of instructions not for a person but for a computer, in order to accomplish a specific task.
  - The process of designing, writing, testing and maintaining a source code of a computer program.

A computer program will provide,

- Control
- Repetition
- Reusability
- High degree of accuracy
- High speed

A dark grey arrow points to the right at the top left. Below it, several thin, curved lines in shades of blue and grey sweep across the left side of the slide.

# Common Programming Structures

- **Sequence of Commands**

The right commands in the right order.

- **Conditional Structures**

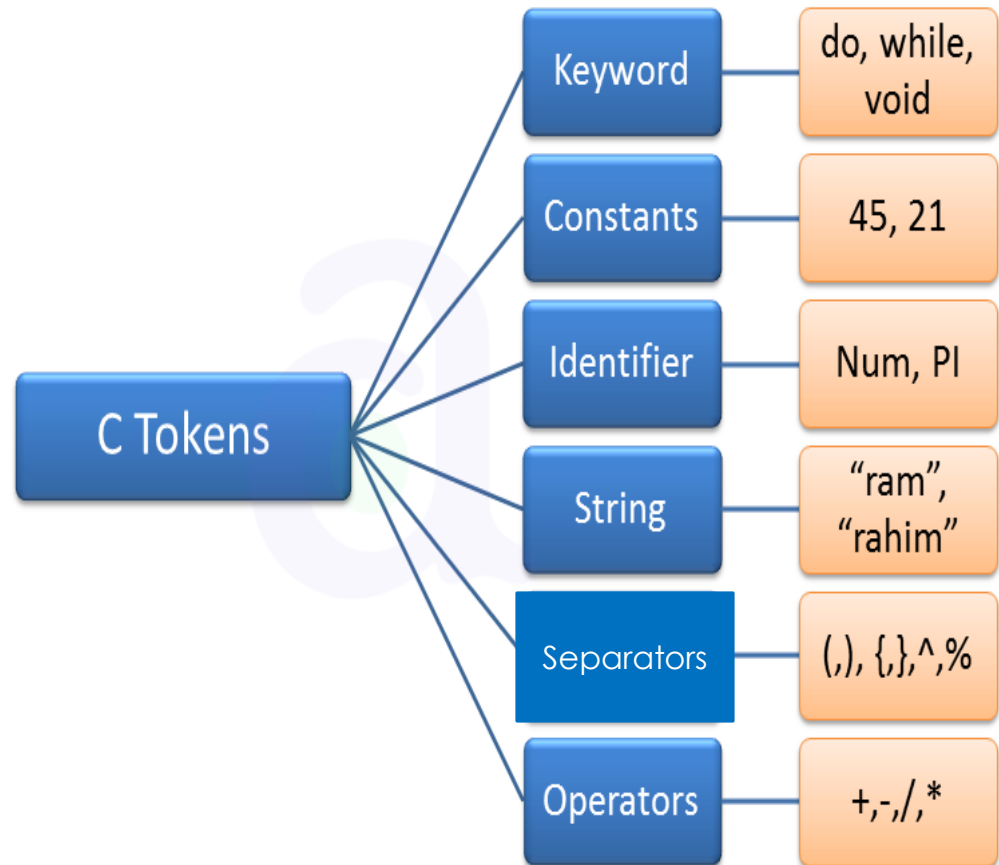
Do certain things based on a true/false or yes/no decisions.

- **Looping Structures**

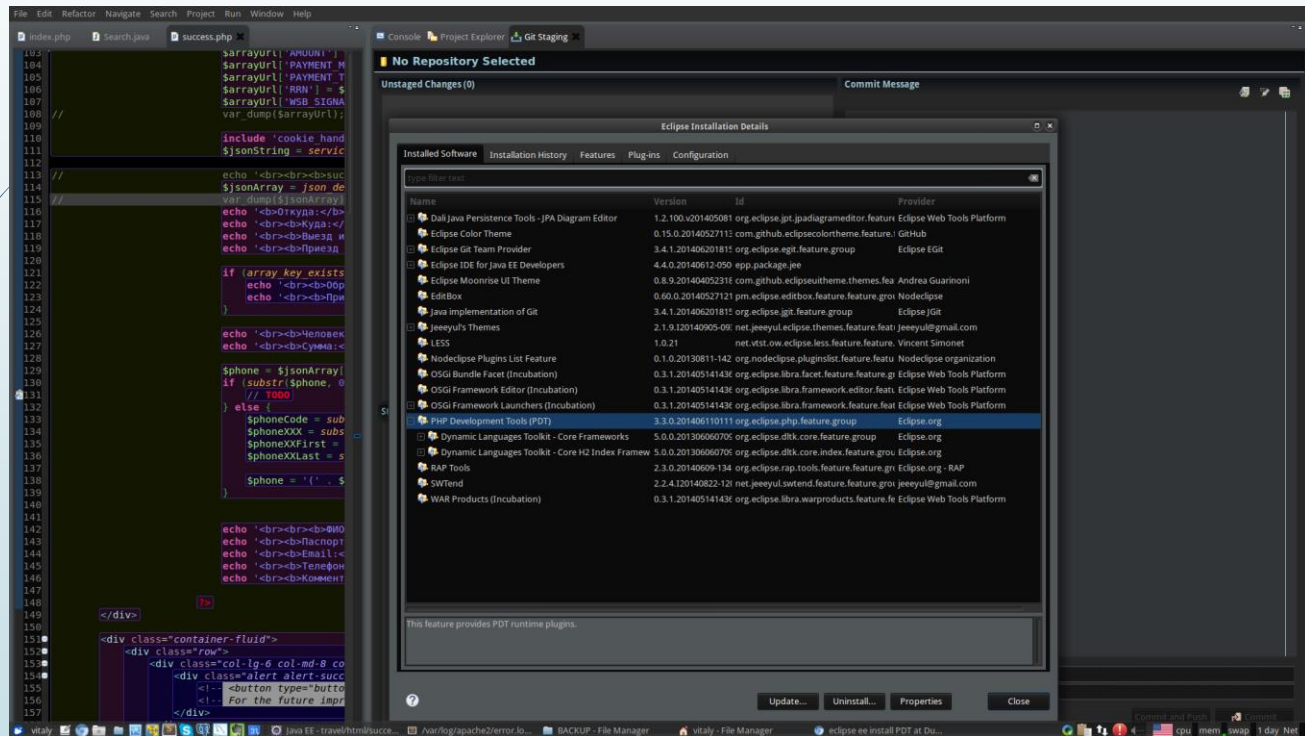
A list of instructions to do more than once.

# Tokens

- Programming tokens are the basic components of a source code.
- C recognizes six types of tokens,
  - Keywords
  - Constants
  - Identifier
  - String
  - Punctuators (Separators)
  - Operators



# Integrated Development Environment (IDE)







# What is an IDE?

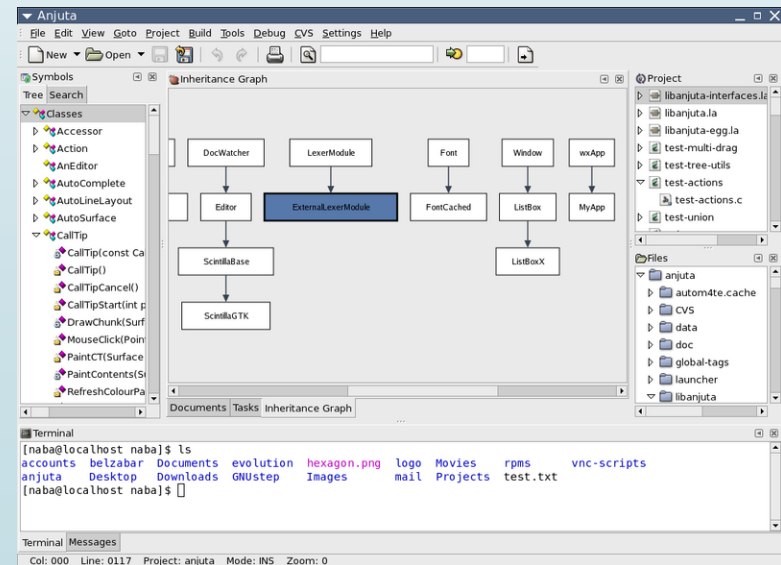
- A collection of development tools exposed through a common user interface.
- Consists of source code editors, build automation tools and a debuggers.
- Comprehensive facilities to computer programmers for software development.
- Contain compiler/interpreter and execution tools.

# Benefits of using IDE

- Maximize the programmer productivity.
- Provide GUI environment.
- Faster.
- Providing instant feedback when syntax errors are found.
- Multiple programming languages within one UI.

## Examples:

- NetBeans
- Eclipse
- Visual Studio
- Dev-C++



# Variables in C

## Variables In C Programming

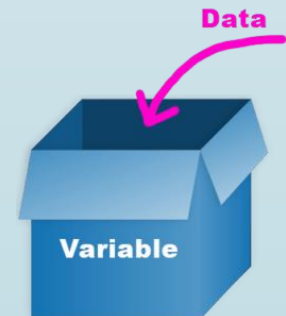
```
int age;  
char b;  
double c;
```



# What are Variables?

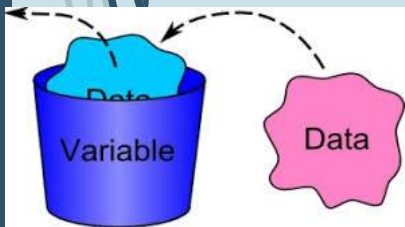
- It is a named memory location in the computer's memory (RAM) where values of different type of data can be stored.
- These values will be stored during the execution of a program (run time).
- The value of variable can be changed.

Ex: The values entered by one user will be different from those entered by another user.



# Storage Locations

- These Storage locations (variables) in memory can be used to hold data for a program.
- A storage location has:
  - **Value:** The **data** stored at that location.
  - **Type:** The **type of data** stored at that location.
  - **Address:** The number which says where **the storage location** is in memory.
  - **Name:** **An identifier** that a program associates with that storage location.



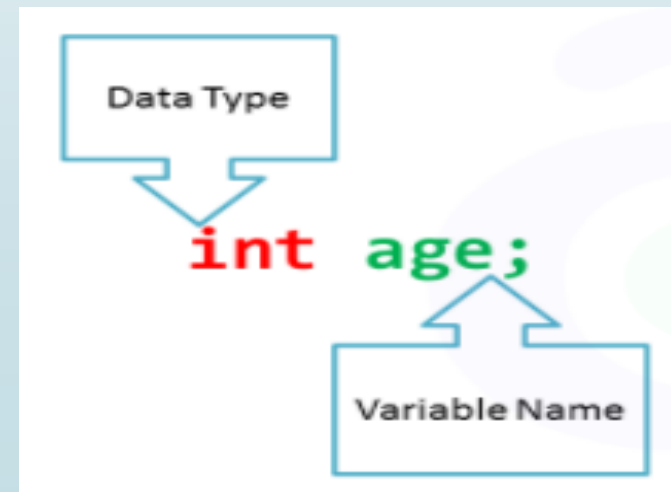
# Variable Declaration


- In C Programming, always have to declare variable before use them.
- **Declaration** – Allocating a space in memory.
  - Specify its type (such as int, Size of the location in memory)
  - specify its name (identifier, such as age, Name for the memory location)

**Syntax:** <data\_type> <identifier> ;

Examples:

- ✓ int age;
- ✓ float length;
- ✓ char letter;
- ✓ double height;



- 
- When a variable is declared, it is designated as **belonging to one of the data types** used in C.
  - The compiler **allocates memory** for the variable according to its **data type**.

### **Declaration**

### **Memory allocates in Bytes**

int a  
char ch  
float payment

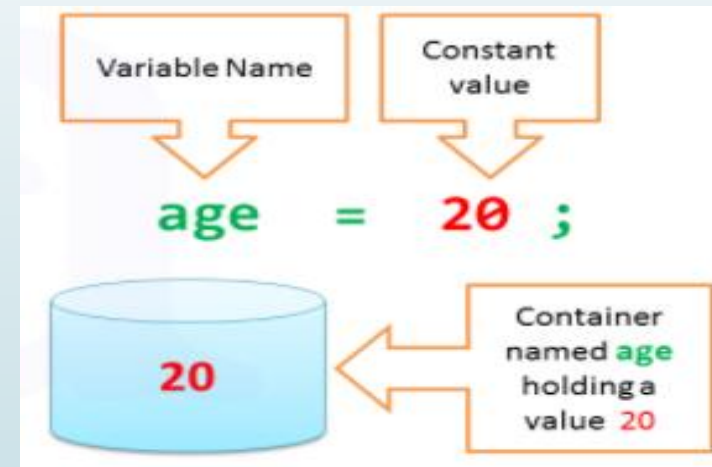
2/4  
1  
4

# Variable Initialization

- After declaration, values can be stored within variables.
- **Initialization** – Assigning an initial value for the variable.
- **Syntax:** <Identifier>=<value or the expression>;

Examples:

- age = 25;
- length = 11.75;
- letter = 'A';
- height = 12.56734;




Sometime this value can be an expression.

- sum = 10 + 20 ;



# Assignment of Values

- The assignment operation can be used to **store a value** in a variable or to **change the value stored** in a variable.
- The **assignment operator** is the equal sign = .
- Right side values will be assigned to left side identifier.

  
age = 25;

- Value will be stored in the memory location identified by the identifier (Name of the variable).

- Both declaration and initialization can be done at once.

`<data_type> <identifier> = <value/expression>;`

```
int x;  
Float y;  
    x = 10;  
    y = 2.5 + 5.5;
```

First declare and  
then assign.

```
int x = 10;  
Float y = 2.5 + 5.5;
```

Both declare and  
assign at the same  
time.

# Lvalues and Rvalues

- Anything that can correctly appear on the left-hand side of an assignment operator is a lvalue.
- An lvalue must specify a storage location where a value must be stored.
- Anything that can legally appear on the right-hand side of an assignment operator is a rvalue.
- All variables (in an expression) which are in a rvalue must have previously been given values.

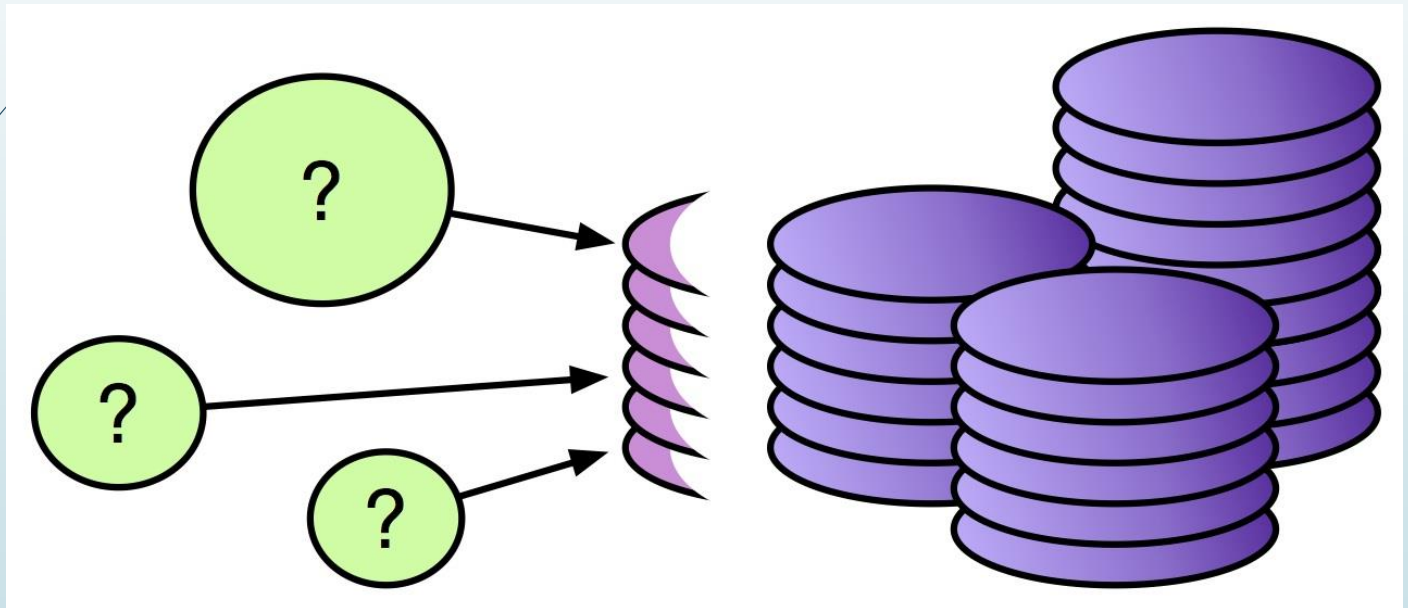
```
sum = age1 + age2 ;
```

- Variable initialization stores the value of the expression (rvalue) into the memory location for the variable (lvalue).
- *Lvalue* – Expressions that refer to a memory location are called "lvalue" expressions.
- *rvalue* – The term rvalue refers to a data value that is stored at some address in memory.
- Variables are lvalues and Numeric literals are rvalues.

CODE:	MEMORY:	
int x = 7;	x	y
int y = 10;	7	10

height	175	0x0041F2
weight	73	0x0041F3
age	24	0x0041F4
	...	0x0041F5
	...	0x0041F6
	...	0x0041F7
	...	0x0041F8

# Data Types in C



# Data Types

- Data type determines the type of data a variable will hold.
- Each data type requires different **amounts of memory** and has some **specific operations** which can be performed over it.
- Two main types of Data Types in C,

## 1. Primitive

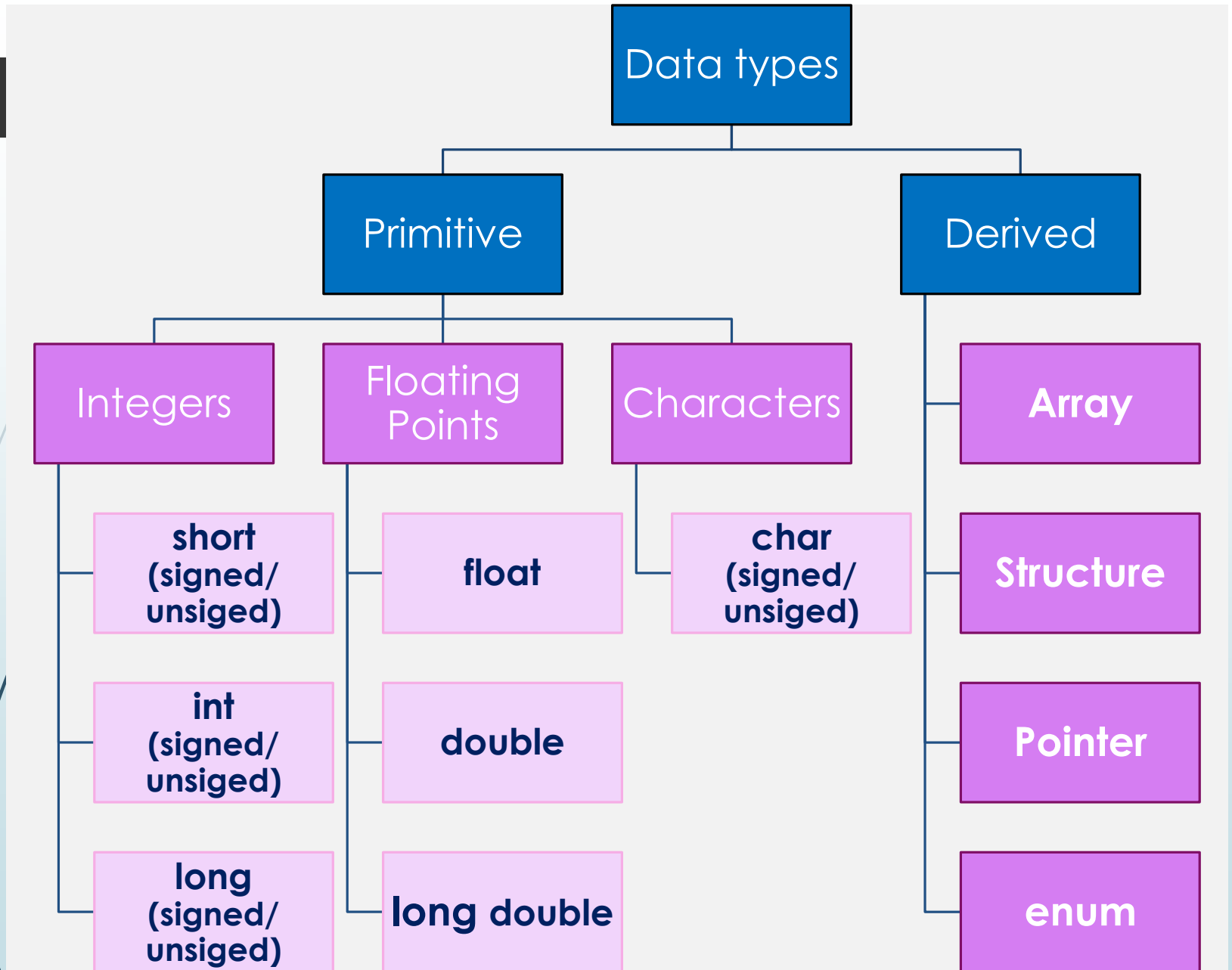
- fundamental data types in C.

Example: integer, floating points, character

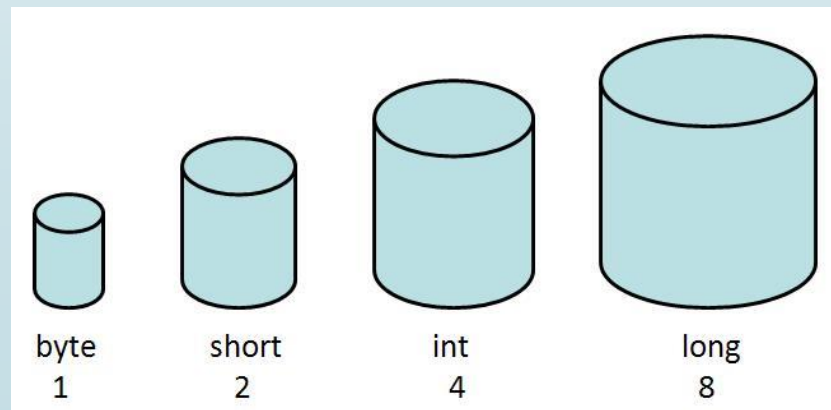
## 2. Non Primitive (Derived)

- Data types made by using primitive data types.


Example: Array, Structure, Pointer etc.



- Depending on the data type, **size of the storage location** and the **range of values** that can have will be determined.
- In mathematics, **Integers** (whole numbers) and **Floating points** (real numbers) form **infinite sets**.
- The **computer has only a finite amount of storage** to represent numbers, therefore there is a **maximum and a minimum values** that can be stored in the computer.







Type	Storage size	Value range	Format specif.
char	1 byte	-128 to 127	%c
unsigned char	1 byte	0 to 255	%c
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647	%d
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295	%u
short	2 bytes	-32,768 to 32,767	%hd
unsigned short	2 bytes	0 to 65,535	%hu
long	8 bytes	-9223372036854775808 to 9223372036854775807	%ld
unsigned long	8 bytes	0 to 18446744073709551615	%lu
float	4 byte	1.2E-38 to 3.4E+38 (precision 6 decimal places)	%f
double	8 byte	2.3E-308 to 1.7E+308 (precision 15 decimal places)	%lf
long double	10 byte	3.4E-4932 to 1.1E+4932 (precision 19 decimal places)	%Lf

Note: The sizes and ranges of datatypes are compiler dependent.

## Integer type

- Integers are used to store whole numbers without decimal points or a fractional part.
  - short - 2 bytes
  - int - 2 or 4 bytes (depending on the computer architecture)
  - long - 4 bytes

## Floating Point Type / Real Numbers

- Floating point data types are used to store numbers with decimal points.
- Have a wider range of numbers than integers.
  - float - 4 bytes
  - double - at least 4 bytes, generally 8 bytes
  - long double - at least as large as double, may be larger
- Many floating point numbers can not be represented exactly in the computer. This representation error, can be magnified by numeric computations.

## Character Type

- char types are used to store only one character.
- char values should be enclosed within single quotes.
- char - 1byte (8bits)
- String values are used to store set of characters(words).
- String values should be enclosed within double quotes.
- Strings use one more byte of storage than the number of characters in the string. (for the ending null character)
- Examples of characters: 'A' '@' '7' 'v' '.' '\'
- Examples of strings:  
"Today"      "2000"      "486-2687"      "Be happy!"

# Exponential Notation

- Exponential notation represents a floating point number as a decimal fraction times a power of 10.
- With **a** being a decimal fraction and **n** integer, the exponential notation **aen** represents  $a \times 10^n$ .
- For example,  
 $1.645e2 \rightarrow 1.645 \times 10^2$  or 164.5

# Data Type Qualifiers

- Qualifiers alters the meaning of **base data types** to yield a **new data type**.
- Manipulate the **range** or the **size** of a particular data type.
- The 4 qualifiers in C : **long, short, signed and unsigned**.
- long and short are called **size qualifiers**.
- Signed and unsigned are called **sign qualifiers**.
- Example: int – when declared normally is of 2/4 bytes. If it is declared as,
  - ✓ **unsigned int** – then its range is from 0 to 65535.
  - ✓ **signed int** – then its range is from (-32767 to 32768).

# Default Values

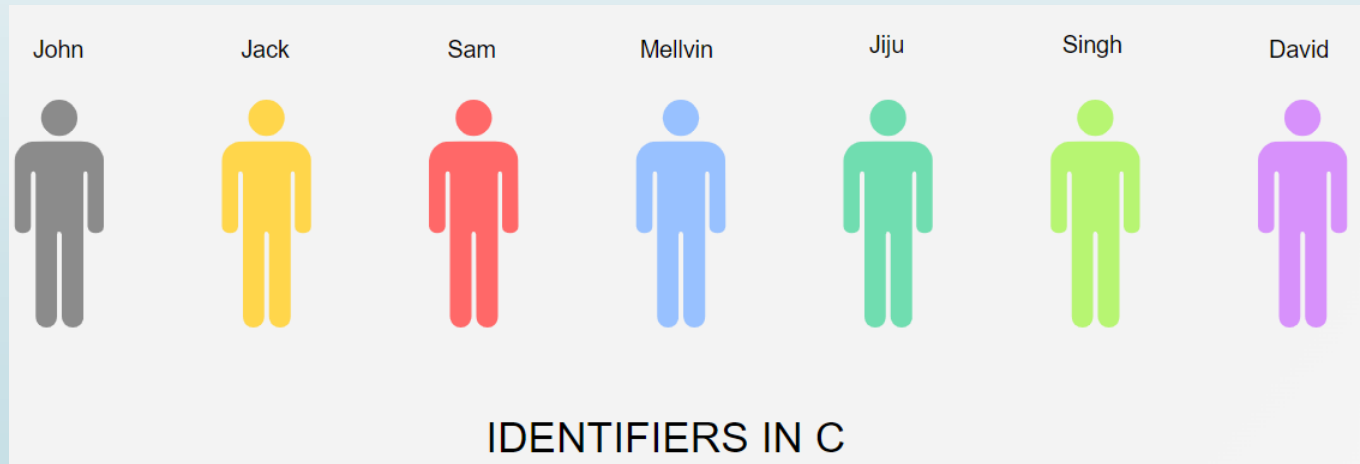
- Each data type in C has a default value (default initialization).
- Used when there is no explicitly set value for the variable.

data type	Default value
char	\u0000
int,short,byte / long	0 / 0L
float /double	0.0f / 0.0d
any reference type	null

- char primitive default value is \u0000, means blank/space character.

# Identifier

- ❖ An identifier is **a name given into a program element** such as constants, variables, objects, functions, classes, user defined data types, and the program itself.



# Rules for making Identifiers in C

- Identifiers are used to name **entities** in C.
- An Identifier can only have alphanumeric characters(a-z , A-Z , 0-9) and underscore(\_).
- The first character of an identifier can only contain alphabet(a-z , A-Z) or underscore (\_).
- Identifiers are **case sensitive** in C.
- **Reserved words** are not allowed.
- No special characters are permitted.  
semicolon, period, whitespaces, slash, comma  
etc.
- An identifier name cannot consist of more than 31 characters.



# Naming Rules

- can be a series of characters consisting of letters, digits and underscores (\_).
- does not begin with a digit.
- may not contain blank spaces.
- may not be a reserved word or keyword (ex: int, return, if, while, for etc.)
- C is case sensitive (uppercase and lowercase letters are different in C).

Ex: total , Total and TOTAL are three different variable names

# Case Sensitive Languages

- Ability to distinguish between uppercase(Capital) and Lowercase(Simple) versions of letters in the language's character set.
- C is Case Sensitive.
- For example **sum** , **SUM** and **Sum** are three different identifiers in C.
- All the keywords in C are Lowercase.

**A ≠ a**

# Reserved words VS Standard Identifiers

## Reserved Word/ Keywords

- Word which has a special meaning in C and cannot be redefined by the programmer.

Examples: **int float else if case return char break void for while etc.**

## Standard Identifier

- Word which has a defined meaning in C but its use can be changed by the programmer.

Examples: **main size open**

```
#include<stdio.h>

int main(){
    int case;
    printf("%d",case);
    return 0;
}
```



```
#include<stdio.h>

int main(){
    int open;
    printf("%d",open);
    return 0;
}
```




```
#include<stdio.h>

int main(){
    int BREAK;
    printf("%d",BREAK);
    return 0;
}
```





# 32 Keywords in C



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

# Exercise

Identify valid and invalid identifiers.

- Age\_23
- My name
- a1\_b2\_c3
- 2things
- regNo
- for
- Pet-s

# Difference between Keywords and Identifiers

keyword	Identifier
Predefined-word.	User defined word.
Must be written in lowercase only.	Can written in lower and upper case.
Has fixed meaning.	Must be meaningful in the program.
Whose meaning has already been explained to the C compiler.	Whose meaning not explained to the C compiler.
Combination of alphabetic characters.	Combination of alphanumeric characters.
Used only for it intended purpose.	Used for required purpose.
Underscore character is not consider as a letter.	Underscore character is considered as a letter.



# sizeof( ) Operator

- `sizeof(variable)`
- sizeof operator can use to get the exact size of a type or a variable on a particular platform.
- It will return the size in unsigned integer format.
- Syntax looks more like a function but it is considered as an operator in C programming.
- It can be applied to any data-type.

# Examples

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

```
int main() {
```

```
    printf("%lu", sizeof(char));  
    printf("%lu", sizeof(int));  
    printf("%lu", sizeof(float));  
    printf("%lu",  
    sizeof(double));
```

```
    return 0;  
}
```

**Output**

1  
4  
4  
8

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

```
int main() {
```

```
    int ivar = 100;  
    char cvar = 'a';  
    float fvar = 10.10;
```

```
    printf("%lu",  
    sizeof(ivar));  
    printf("%lu",  
    sizeof(cvar));  
    printf("%lu",  
    sizeof(fvar));
```

```
    return 0;  
}
```

**Output :**  
2 1 4





# What is the answer?

```
#include<stdio.h>
void main()
{
    int a=2;
    double b = 20.123;
    printf("%lu", sizeof(a+b));
}
```

# Constants

- Constants are non-modifiable values which cannot be changed during the program execution.
- Constants can be of any of the basic data types like an integer constant, a floating constant, a character constant, or a string literal.
- Two simple ways in C to define constants:
  - Using `#define` preprocessor
  - Using `const` keyword
- Syntax:  
`#define length 25`  
`const int length=100;`



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

```
#define LENGTH 10
```

```
int main() {
```

```
    int area;
```

```
    const int WIDTH=20;
```

```
    area = LENGTH * WIDTH;
```

```
    printf("value of area : %d", area);
```

```
    return 0;
```

```
}
```

What is the output?

# ASCII Encoding Scheme

- The American Standard Code for Information Interchange (ASCII) encoding scheme is a system of assigning a number to a character.
- In an ASCII file, each alphabetic, numeric, or special character is represented with a 7-bit binary number.
- There are 256 characters but 128 are used mostly in programs.
- Most computers store characters using the ASCII encoding scheme.
  - ✓ UNIX, DOS based OS
- Some other different encoding schemes are Unicode, EBCDIC.

# ASCII Table

dec	oct	hex	ch
0	0	00	<b>NUL</b> (null)
1	1	01	<b>SOH</b> (start of header)
2	2	02	<b>STX</b> (start of text)
3	3	03	<b>ETX</b> (end of text)
4	4	04	<b>EOT</b> (end of transmission)
5	5	05	<b>ENQ</b> (enquiry)
6	6	06	<b>ACK</b> (acknowledge)
7	7	07	<b>BEL</b> (bell)
8	10	08	<b>BS</b> (backspace)
9	11	09	<b>HT</b> (horizontal tab)
10	12	0a	<b>LF</b> (line feed - new line)
11	13	0b	<b>VT</b> (vertical tab)
12	14	0c	<b>FF</b> (form feed - new page)
13	15	0d	<b>CR</b> (carriage return)
14	16	0e	<b>SO</b> (shift out)
15	17	0f	<b>SI</b> (shift in)
16	20	10	<b>DLE</b> (data link escape)
17	21	11	<b>DC1</b> (device control 1)
18	22	12	<b>DC2</b> (device control 2)
19	23	13	<b>DC3</b> (device control 3)
20	24	14	<b>DC4</b> (device control 4)
21	25	15	<b>NAK</b> (negative acknowledge)
22	26	16	<b>SYN</b> (synchronous idle)
23	27	17	<b>ETB</b> (end of transmission block)
24	30	18	<b>CAN</b> (cancel)
25	31	19	<b>EM</b> (end of medium)
26	32	1a	<b>SUB</b> (substitute)
27	33	1b	<b>ESC</b> (escape)
28	34	1c	<b>FS</b> (file separator)
29	35	1d	<b>GS</b> (group separator)
30	36	1e	<b>RS</b> (record separator)
31	37	1f	<b>US</b> (unit separator)

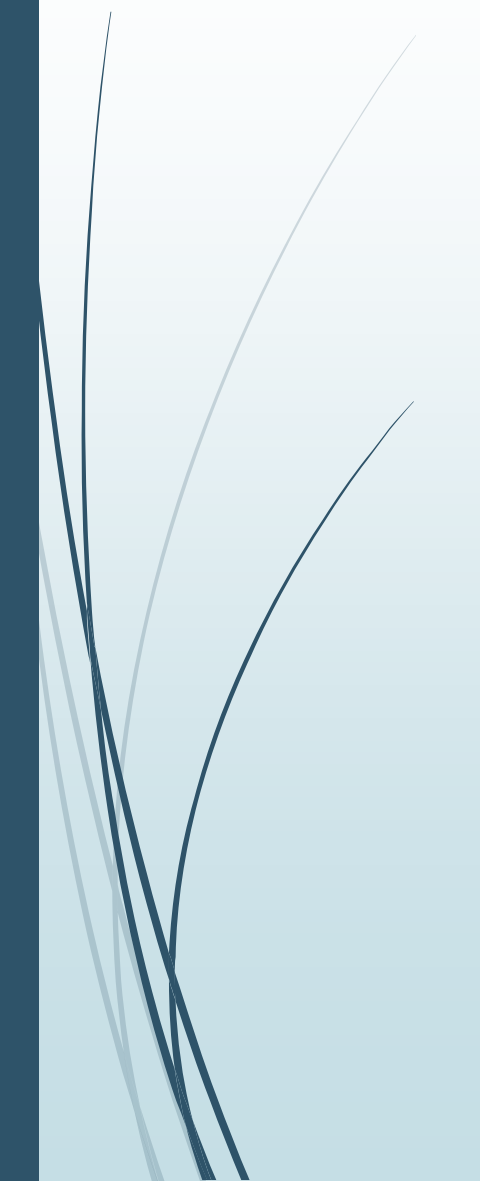
dec	oct	hex	ch
32	40	20	(space)
33	41	21	!
34	42	22	"
35	43	23	#
36	44	24	\$
37	45	25	%
38	46	26	&
39	47	27	'
40	50	28	(
41	51	29	)
42	52	2a	*
43	53	2b	+
44	54	2c	,
45	55	2d	-
46	56	2e	.
47	57	2f	/
48	60	30	0
49	61	31	1
50	62	32	2
51	63	33	3
52	64	34	4
53	65	35	5
54	66	36	6
55	67	37	7
56	70	38	8
57	71	39	9
58	72	3a	:
59	73	3b	;
60	74	3c	<
61	75	3d	=
62	76	3e	>
63	77	3f	?

dec	oct	hex	ch
64	100	40	@
65	101	41	A
66	102	42	B
67	103	43	C
68	104	44	D
69	105	45	E
70	106	46	F
71	107	47	G
72	110	48	H
73	111	49	I
74	112	4a	J
75	113	4b	K
76	114	4c	L
77	115	4d	M
78	116	4e	N
79	117	4f	O
80	120	50	P
81	121	51	Q
82	122	52	R
83	123	53	S
84	124	54	T
85	125	55	U
86	126	56	V
87	127	57	W
88	130	58	X
89	131	59	Y
90	132	5a	Z
91	133	5b	[
92	134	5c	\
93	135	5d	]
94	136	5e	^
95	137	5f	_
96	140	60	`
97	141	61	a
98	142	62	b
99	143	63	c
100	144	64	d
101	145	65	e
102	146	66	f
103	147	67	g
104	150	68	h
105	151	69	i
106	152	6a	j
107	153	6b	k
108	154	6c	l
109	155	6d	m
110	156	6e	n
111	157	6f	o
112	160	70	p
113	161	71	q
114	162	72	r
115	163	73	s
116	164	74	t
117	165	75	u
118	166	76	v
119	167	77	w
120	170	78	x
121	171	79	y
122	172	7a	z
123	173	7b	{
124	174	7c	
125	175	7d	}
126	176	7e	~
127	177	7f	<b>DEL</b> (delete)



# ASCII Representation

Characters	Equivalent ASCII
'A'	65
'Z'	90
'a'	97
'z'	122
'0'	48
'9'	57
' '	32
'+'	43



# Quick Quiz – What is the suitable data type for each value below?

Type	Storage size	Value range
char	1 byte	-128 to 127
unsigned char	1 byte	0 to 255
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	8 bytes	-9223372036854775808 to 9223372036854775807
unsigned long	8 bytes	0 to 18446744073709551615
float	4 byte	1.2E-38 to 3.4E+38 (precision 6 decimal places)
double	8 byte	2.3E-308 to 1.7E+308 (precision 15 decimal places)
long double	10 byte	3.4E-4932 to 1.1E+4932 (precision 19 decimal places)

- 47
- “Alpha”
- -35
- “#”
- ‘45245’
- '45.158925448'
- “4.69”
- 96515234

# Console Inputs and Outputs in C

## **stdin, stdout, stderr**

- When your C program begins to execute, some input/output devices are opened automatically.
- **stdin**  
The “standard input” device, usually your keyboard.
- **stdout**  
The “standard output” device, usually your monitor.
- **stderr**  
The “standard error” device, usually your monitor.
- Some C library I/O functions automatically use these devices.



# Formatted Console Output

- In C, formatted output is created using the `printf()` function.
- `printf()` function outputs data to the standard output stream.
- `printf()` call contains a set of **format specifiers** that describes the output format.
- `printf( format specifiers, other arguments)`

Ex:

- ✓ `int a=10, b=12;`
- ✓ `printf("%d %d", a, b);`

# Formatted Console Output

- The syntax  
*printf( format, arg1, arg2, ... );*  
where the “**format**” is a string containing
  - literals to be printed.
  - conversion specifications/ Format specifiers.
- “**Arguments**” are the set of variables from which the values are coming. Correspond to each conversion specification in format-string

```
printf(“Patient name: %d”, pname);
```

String literals

Conversion  
Specification

Argument  
Variable



# Common Format Specifiers

- Format specifiers defines the type of data to be printed on standard output.
- Start with a percentage % operator and followed by a special character.
- `printf()` works with several types of format specifiers.

# Common Format Specifiers

Format Specifier	Description
%d (%i)	int (Signed integer)
%f	float or double (signed decimal)
%c	char (single character)
%s	Array of char (sequence of characters)
%u	int (Unsigned integer)
%e or %E	Scientific notation of floating point values
%lf	double (Long float)
%x	Hexa decimal representation of Integer
%o	Octal representation of Integer
%p	Pointer value (memory address stored in pointer)



# Escape Sequences

- An escape sequence refers to a combination of characters beginning with a back slash (\) followed by a character.
- Escape sequences represent non-printable and special characters in literal strings.

# Escape Sequences

Escape Sequence	Name	Description
\b	Backspace	Move the cursor back one position on the current line
\n	New Line	Move the cursor to the beginning of the next line
\t	Tab (Horizontal)	Move the cursor to the next horizontal tab position
\'	Single Quote	Output the single quote (') character
\"	Double Quote	Output the double quote (") character
\\	Backslash	Output the backslash (\) character
\?	Question Mark	Output the question mark (?) character
\r	Carriage Return	Move the cursor to the beginning of the current line
\v	Vertical Tab	Move the cursor to the next vertical tab position

# printf( ) Examples

Ex.c

```
1  #include<stdio.h>
2
3  int main()
4  {
5      char ch = 'B';
6      printf("ch = %c\n", ch); //printing character data
7
8      int x = 45, y = 90;
9      printf("x = %d\n", x); //printing integer data
10     printf("y = %i\n", y);
11
12     float f = 12.67;
13     printf("f= %f\n", f); //print float value
14
15     char str[] = "Hello World";
16     printf("string= %s\n", str); //printing string value
17
18     return 0;
19 }
```

# More on floating points

- The size of float (single precision float data type) is 4 bytes. And the size of double (double precision float data type) is 8 bytes.
- Floating point variables has a precision of 6 digits whereas the precision of double is 15 digits.

```
#include<stdio.h>

int main(){

    float x=15.123456789;
    double y=15.123456789;

    printf("value of x:%.5f\n",x);

    printf("\nvalue of x:%.9f\n",x);
    printf("value of y:%.9lf\n",y);

    printf("\nvalue of x:%10.5f\n",x);
    return 0;
}
```

```
value of x:15.12346
value of x:15.123456955
value of y:15.123456789
value of x: 15.12346
```



# Formatted Console Input

- keyboard input is accomplished using the `scanf()` function.
- Calling `scanf()` is similar to calling `printf()`.
- `Scanf()` function reads from the standard input stream.
- `Scanf()` contains a set of **format specifiers** that indicates the type of data that should be entered.

Ex:

- ✓ `int x;`
- ✓ `scanf("%d",&x);`

# Formatted Console Input

Syntax: `scanf(format, arg1, arg2, ...)`

- The format string has a similar structure to the format string in `printf( )`.

(but do not use strings/escape sequences)

- The arguments are the **addresses of the variables** into which the input is store. “&” indicates the memory address.

Ex: `scanf("%f", &payment);`

# Common scanf( ) conversions

Format specifier	Meaning	Description
%d	integer	argument is the address of a int
%f %e	floating point number	The matching argument is the address of a float May be preceded by l to indicate the argument is of the address of a double rather than a float
%s	a word	The matching argument is the address of a char or the name of a char array The caller must insure the array is large enough to store the input string and the terminating \0 character More on this later
%c	Single character	The matching arguments is the address of a char Does not skip over white-space More on this later

# scanf( ) Examples

```
#include<stdio.h>

int main(){
    int age;
    double gpa;

    printf("Input your age: ");
    scanf( "%d", &age );    /* note & */
    printf(" input your gpa: ");
    scanf ("%lf", &gpa );

    return 0;
}
```

# Type Casting

- Type casting is a way of converting a variable from one data type to another data type format.
- If you want to store a 'long' value into a simple integer then you can type cast 'long' to 'int'.
- New data type should be indicated within parenthesis before the constant/variable/expression.

**Syntax: (data\_type)expression**

# Type Casting

- Two type of casting available in C language.

Examples:

- ✓ `int i = (int)3.14` → **Downcasting**,  
information loss
- ✓ `float j = (float)10/2;` → **Upcasting**,  
no information loss

# Implicit Type Casting

- This conversion is done by the compiler.
- When more than one data type of variables are used in an expression, the compiler converts data types to avoid loss of data.

```
#include <stdio.h>
int main() {
    int a = 10;
    char b = 'S';
    float c = 2.88;
    a = a+b;
    printf("Implicit conversion from character to integer : %d\n",a);
    c = c+a;
    printf("Implicit conversion from integer to float : %f\n",c);
    return 0;
}
```

```
Implicit conversion from character to integer : 93
Implicit conversion from integer to float : 95.879997
```

# Explicit Type casting

- This conversion is done by user.
- Data type is converted into another data type forcefully by the user.
- Syntax: (type) expression.

```
#include <stdio.h>
int main() {
    float c = 5.55;
    int s = (int)c+1;
    printf("Explicit Conversion : %d\n",s);
    return 0;
}
```

Explicit Conversion : 6



# Type Casting Example

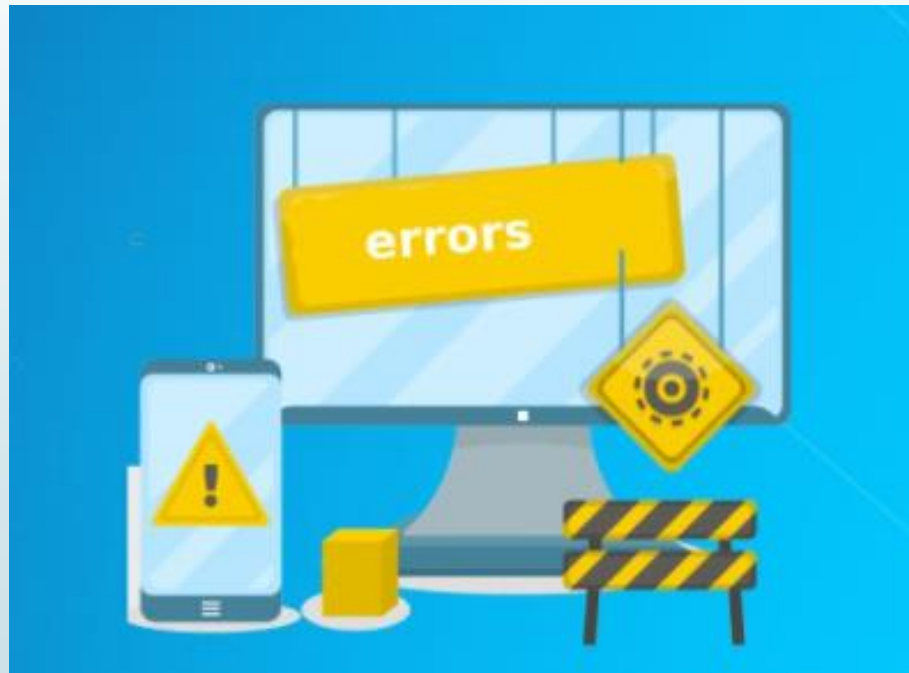
```
#include<stdio.h>
int main(){
    int x=10,y=3;
    float z;

    z = x/y;
    printf("value of z before type casting: %f\n",z);

    z = (float)x/y;
    printf("value of z after type casting: %f\n",z);
    return 0;
}
```

```
value of z before type casting: 3.000000
value of z after type casting: 3.333333
```

# Errors in C



# Syntax Errors

- Errors detected by the compiler are called compile-time errors or Syntax errors.
- It is occurred in the structure of the program statements.

Examples:


- ✓ Not declaring a variable or misspelling the variable name.
- ✓ Missing symbols (; " etc.)
- ✓ Missing curly brace in selection statement.
- Must be fixed before the code execution.

```
#include<stdio.h>
int main(){
    int a=25;
    printf("answer for a/5: %d",(a/5)
    return 0;
}
```

# Logical Errors


- Errors that are detected while the program is executing are called execution or run-time errors (Logical Errors).
- The best way to detect logical errors is by desk checking (informal manual checking).
- During execution, a logic error announces itself by wrong answers, warnings, unexpected termination of the program.

```
#include<stdio.h>
int main(){
    int a=25;
    printf("answer for a/0: %d",(a/0));
    return 0;
}
```




# Some examples of logic errors

- ❖ Value divided by zero.
- ❖ Use of Wrong operators.
- ❖ Assigning value to the wrong variable.
- ❖ Incorrect input or output formatting.

A dark grey arrow points right from the left edge. Below it, several thin, curved lines in shades of blue and grey sweep upwards from the bottom left corner.

**Any**  
**Question**

A stylized blue lightbulb with a question mark inside the glass part. The base of the bulb is composed of three horizontal bars. Ten small blue squares are arranged in a semi-circle above the bulb, representing light rays.

# THANK YOU...!

