Computer Programming: ‘C’

BY: RITU DEVI

Introduction

- Software is a collection of programs and a program is a collection of instructions.
- Process of program development
  - User Requirements
  - Problem analysis
  - Input and output
  - Designing Algorithm
  - Program coding
- Design Methods
  - Top- Down design
  - Bottom-up design
  - Modular approach

- Types of Programming Languages
  - Low level languages: *Machine Level and assembly level language*.
  - High Level languages
- Translators
  - Assembler
  - Compiler
  - Interpreter

Introduction

- C is a general purpose programming language developed at AT&T's Bell Laboratories of USA in 1972 by ***Dennis Ritchie***.
- Many of the ideas of C language were derived and taken from 'B' language.
- As many features came from B it was named as 'C'.
- The purpose of developing C language is for creating system applications that direct interacts to the hardware devices such as drivers, kernals etc.
- C programming is considered as the base for other programming languages, that is why it is known as mother language

Evolution of C

Language	Developed Year	Full name	Who developed	Comments
ALGOL	1958	Algorithmic Language	European and American computer scientists Committee	General
CPL	1963	Combined Programming Language	Mathematical laboratory and University of Cambridge	Difficult to implement and hard to learn
BCPL	1967	Basic combined programming language	Martin Richards at Cambridge university	It deals with some specific problems
B	1970	B Language	Ken Thompson at AT &T labs	Specified to some problems
C	1972	C Language	Dennis Ritchie at AT & T labs	Simple to learn and easy to implement

Features of C Language

- **Simple and easy to use.**
- **Portable** – C programs can be run in many platforms/machines without changes or with change.
- **Structured programming language**
- **Pointers** – C supports pointers, pointers has direct access to memory.
- **Rich functional libraries** – C provides a lot of inbuilt functions that supports many features.
- **Mid level programming language** – It is used to develop system applications such as kernel, driver etc. It also supports the feature of high level language.
- **Extensible** – It adopts new features quickly.
- **Powerful** – C uses Wide variety of 'Data Types', functions and loops and control statements

Structure of a C Program

- C program is a collection of one or more **functions**.
- **Preprocessor directives** are processed through Preprocessor before the C source code passes through compiler.(e.g.-#include, #define).
- It may be possible that some variables have to be used in many functions, so it is necessary to declare them globally. These variables are called **global variables.**

```
Comments
Preprocessor directives
Global variables
main( ) function
{
    local variables
    statements
    ……..
    ……..
}
func1( )
{
    local variables
    statements
    ……..
    ……..
}
func2( )
{
    local variables
    statements
    ……..
    ……..
}
```

Environment For C

- **The steps for the execution of C program are as-**
  1. Program creation
  2. Program compilation
  3. Program execution

- The C programs are written in mostly two environments, UNIX (Compiler is gcc or cc) and MS-DOS.
- **MS-DOS Environment:**
  - Creation, compilation and execution of program can be done using command line or IDE(Integrated Development Environment).

- ***Command Line:***

  (a)Program creation: The program file can be created using any editor and should be saved with .c extension.

  (b) Program compilation: Compiled at DOS prompt by writing

  C:\>tcc filename (in Turbo C)

  C:\>bcc filename (in Borland C)

  (c) Program execution: Compiled file is executed at DOS prompt by writing-

  C:\>filename

- ***Integrated Development Environment:***

  Assuming that you are using a Turbo C or Turbo C++ compiler here are the steps that you need to follow to compile and execute your first C program…

  - Start the compiler at **C>** prompt. The compiler (TC.EXE is usually present in C:\TC\BIN directory).
  - Select **New** from the **File** menu.
  - Type the program.
  - Save the program using **F2** under a proper name (say Program1.c).
  - Use **Ctrl + F9** to compile and execute the program.
  - Use **Alt + F5** to view the output.

Elements of C

Every language has some basic elements and grammatical rules. These basic elements are *character set*, *variables*, *data types*, *constants*, *keywords* (reserved words), *variable declaration*, *expressions*, *statements* etc.

All of these are used to construct a C program

- **Character Set:**

  C language also has a set of characters which include **alphabets**, **digits** and **special symbols**. C language supports a total of 256 characters.

Alphabets	A, B, ..., Y, Z a, b, ..., y, z
Digits	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Special symbols	~ ‘ ! @ # % ^ & * ( ) _ - + = | \ { } [ ] : ; " ' < > , . ? /

ASCII Chart

There are 256 distinct characters used by PCs and Laptops (values range from 0 to 255).

- first 128 are often called ASCII characters
- and the next 128 as Extended ASCII characters

Character Type	No. of Characters
Capital letters	26
Small-case Letters	26
Digits	10
Special Symbols	32
Control Character	34
Graphics Character	128
Total	256

- program that can generate the ASCII chart

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

    for ( ch = 0 ; ch <= 255 ; ch++ )
        printf ( "%d %c\n", ch, ch ) ;
    return 0 ;
}
```

Value	Char	Value	Char	Value	Char	Value	Char	Value	Char	Value	Char
0		22	▬	44	,	66	B	88	X	110	n
1	☺	23	↕	45	-	67	C	89	Y	111	o
2	☻	24	↑	46	.	68	D	90	Z	112	p
3	♥	25	↓	47	/	69	E	91	[	113	q
4	♦	26	→	48	0	70	F	92	\	114	r
5	♣	27	←	49	1	71	G	93	]	115	s
6	♠	28	∟	50	2	72	H	94	^	116	t
7	•	29	↔	51	3	73	I	95	_	117	u
8	◘	30	▲	52	4	74	J	96	`	118	v
9	○	31	▼	53	5	75	K	97	a	119	w
10	◙	32		54	6	76	L	98	b	120	x
11	♂	33	!	55	7	77	M	99	c	121	y
12	♀	34	"	56	8	78	N	100	d	122	z
13	♪	35	#	57	9	79	O	101	e	123	{
14	♫	36	$	58	:	80	P	102	f	124	|
15	☼	37	%	59	;	81	Q	103	g	125	}
16	►	38	&	60	<	82	R	104	h	126	~
17	◄	39	'	61	=	83	S	105	i	127	ᴹH
18	↕	40	(	62	>	84	T	106	j	128	Ç
19	‼	41	)	63	?	85	U	107	k	129	ü
20	¶	42	*	64	@	86	V	108	l	130	é
21	§	43	+	65	A	87	W	109	m	131	â

Value	Char	Value	Char	Value	Char	Value	Char	Value	Char	Value	Char
132	ä	154	Ü	176	▒	198	╞	220	▄	242	≥
133	à	155	¢	177	▒	199	╟	221	▌	243	≤
134	å	156	£	178	▓	200	╚	222	▐	244	⌠
135	c	157	¥	179	│	201	╔	223	▀	245	⌡
136	ê	158	₧	180	┤	202	╩	224	α	246	÷
137	ë	159	ƒ	181	╡	203	╦	225	ß	247	≈
138	è	160	á	182	╢	204	╠	226	Γ	248	°
139	ï	161	í	183	╖	205	═	227	π	249	∙
140	î	162	ó	184	╕	206	╬	228	Σ	250	·
141	ì	163	ú	185	╣	207	╧	229	σ	251	√
142	Ä	164	ñ	186	║	208	╨	230	µ	252	ⁿ
143	Å	165	Ñ	187	╗	209	╤	231	τ	253	²
144	É	166	ª	188	╝	210	╥	232	Φ	254	■
145	æ	167	º	189	╜	211	╙	233	Θ	255	
146	Æ	168	¿	190	╛	212	╘	234	Ω		
147	ô	169	⌐	191	┐	213	╒	235	δ		
148	ö	170	¬	192	└	214	╓	236	∞		
149	ò	171	½	193	┴	215	╫	237	ø		
150	û	172	¼	194	┬	216	╪	238	ϵ		
151	ù	173	¡	195	├	217	┘	239	∩		
152	ÿ	174	«	196	─	218	┌	240	≡		
153	Ö	175	»	197	┼	219	█	241	±		

- **Delimiters**:

  Delimiters are used for syntactic meaning in C. These are as given below

:	colon	used for label
;	semicolon	end of statement
( )	parentheses	used in expression
[ ]	square brackets	used for array
{ }	curly braces	used for block of statements
#	hash	preprocessor directive
,	comma	variable delimiter

Token

A token is the smallest element of a program that is meaningful to the compiler. Tokens can be classified as follows:

- Keywords
- Identifiers
- Constants
- Strings
- Special Symbols
- Operators

Keywords:

There are certain words that are reserved for doing specific tasks. These words are known as keywords and they have standard, predefined meaning in C.

- ✓ They are always written in lowercase.
- ✓ There are only 32 keywords available in C.

Keywords in C Language

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

- **Identifiers**

Identifiers are user defined words and are used to give names to entities like variables, arrays, functions, structures etc.

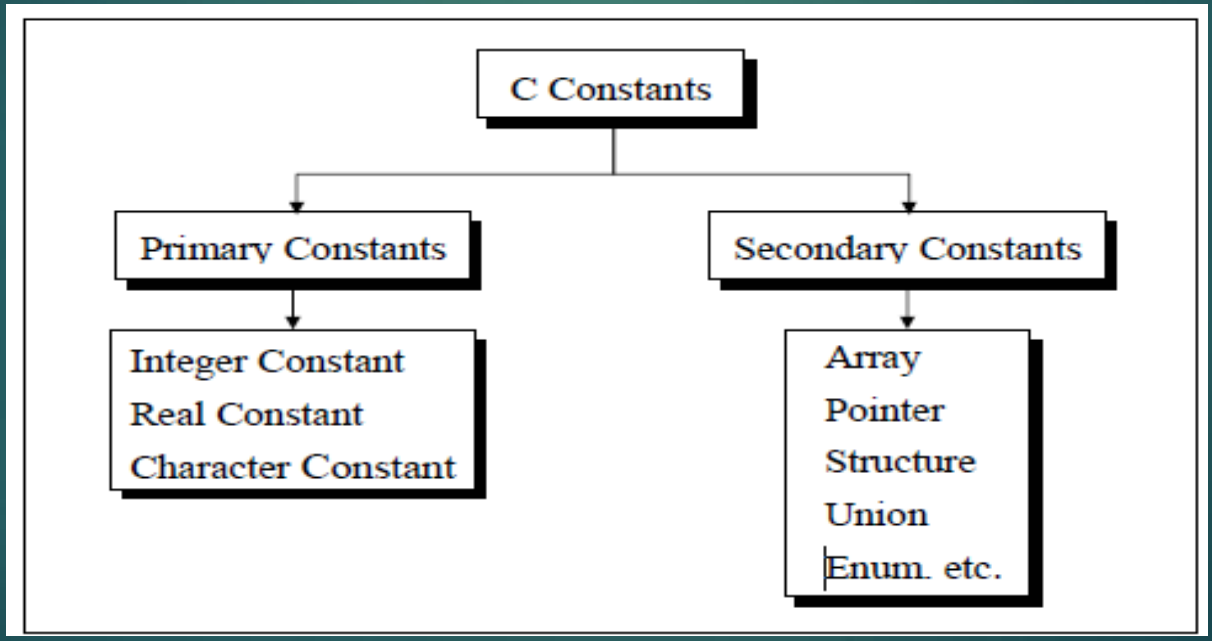
Rules for naming identifiers are:

- ✓ The name should consist of only alphabets (both upper and lower case), digits and underscore.
- ✓ First character should be an alphabet or underscore.
- ✓ The name should not be a keyword.
- ✓ Since C is case sensitive, the uppercase and lowercase letters are considered different. For example code, Code and CODE are three different identifiers.

Constant:

Constant is a value that cannot be changed during execution of the program. C constants can be divided into two major categories:

- Primary Constants
- Secondary Constants



Variables:

Variable is a name that can be used to store values. Variables can take different values but one at a time. These values can be changed during execution of the program. A data type is associated with each variable.

The rules for naming variables are same as that for naming identifiers.

- **Declaration of Variables:** It is must to declare a variable before it is used in the program. The type and range of values that a variable can store depends upon its data type.

  Syntax is: *Data type variablename;*

- **Initialisation of Variables:** When a variable is declared it contains undefined value commonly known as garbage value. Variable initialization means assigning a value to the variable. For example,

  int i =5;

  float x = 8.9, Y = 10.5;

Operators:

- An operator specifies an operation to be performed that yields a value. An operand is a data item on which an operator acts. Some operators require two operands, while others act upon only one operand.
- C is very rich in operators. There are about 45 operators available in C. But there is no operator for exponentiation.
  - Arithmetic operators
  - Assignment operators .
  - Increment and Decrement operators
  - Relational operators.
  - Logical operators
  - Conditional operator
  - Comma operator
  - sizeof operator
  - Bitwise operators.

- **Data Types:**

  There are four fundamental data types in C. Storage representation of these data types is different in memory.

  - **'char'** is used to store any single character.
  - **' int'** is used to store integer value.
  - **'float'** is used for storing single precision floating point number.
  - **'double'** is used for storing double precision floating point number.

  There are two types of type qualifiers-

  - ***Size qualifiers:*** short, long.
  - ***Sign qualifiers:*** signed, unsigned.

Data Type	Range	Bytes	Format
signed char	-128 to + 127	1	%c
unsigned char	0 to 255	1	%c
short signed int	-32768 to +32767	2	%d
short unsigned int	0 to 65535	2	%u
signed int	-32768 to +32767	2	%d
unsigned int	0 to 65535	2	%u
long signed int	-2147483648 to +2147483647	4	%ld
long unsigned int	0 to 4294967295	4	%lu
float	-3.4e38 to +3.4e38	4	%f
double	-1.7e308 to +1.7e308	8	%lf
long double	-1.7e4932 to +1.7e4932	10	%Lf

Note: The sizes and ranges of int, short and long are compiler dependent. Sizes in this figure are for 16-bit compiler.

- **Expression:**

An expression is a valid combination of *operators*, *constants*, *variables* and *function calls*. The expression can be arithmetic, logical or relational. Some examples are as

- x+y+2 -arithmetic operation.
- a = b+c -uses two operators (=) and (+).
- a > b - relational expression
- func(a, b) - function call

- **Comments:**

Comments are used for increasing readability of the program and generally used for documentation purposes.

Comments can be placed anywhere in a program and are enclosed between the delimiters /* and */.

- /*Single line comment*/ or //single line comment
- /*This is multiline

  comment */

Comments can't be nested i.e. we can't write a comment inside another comment

- **Statements:** A statement is an executable part of the program and causes the computer to carry out some action
  - Expression statements
  - Compound statements
  - Selection statements (if, if. .. else, switch)
  - Iterative statements (for, while, do, ..while)
  - Jump statements (goto, continue, break, return)
  - Label statements (case, default, label statement used in goto)

- Normally you are at liberty to give functions whatever names you like, but **"*main*"** is special
  - your program begins executing at the beginning of *main*.
- *main* will usually call other functions to help perform its job, some that you wrote, and others from libraries that are provided for you.

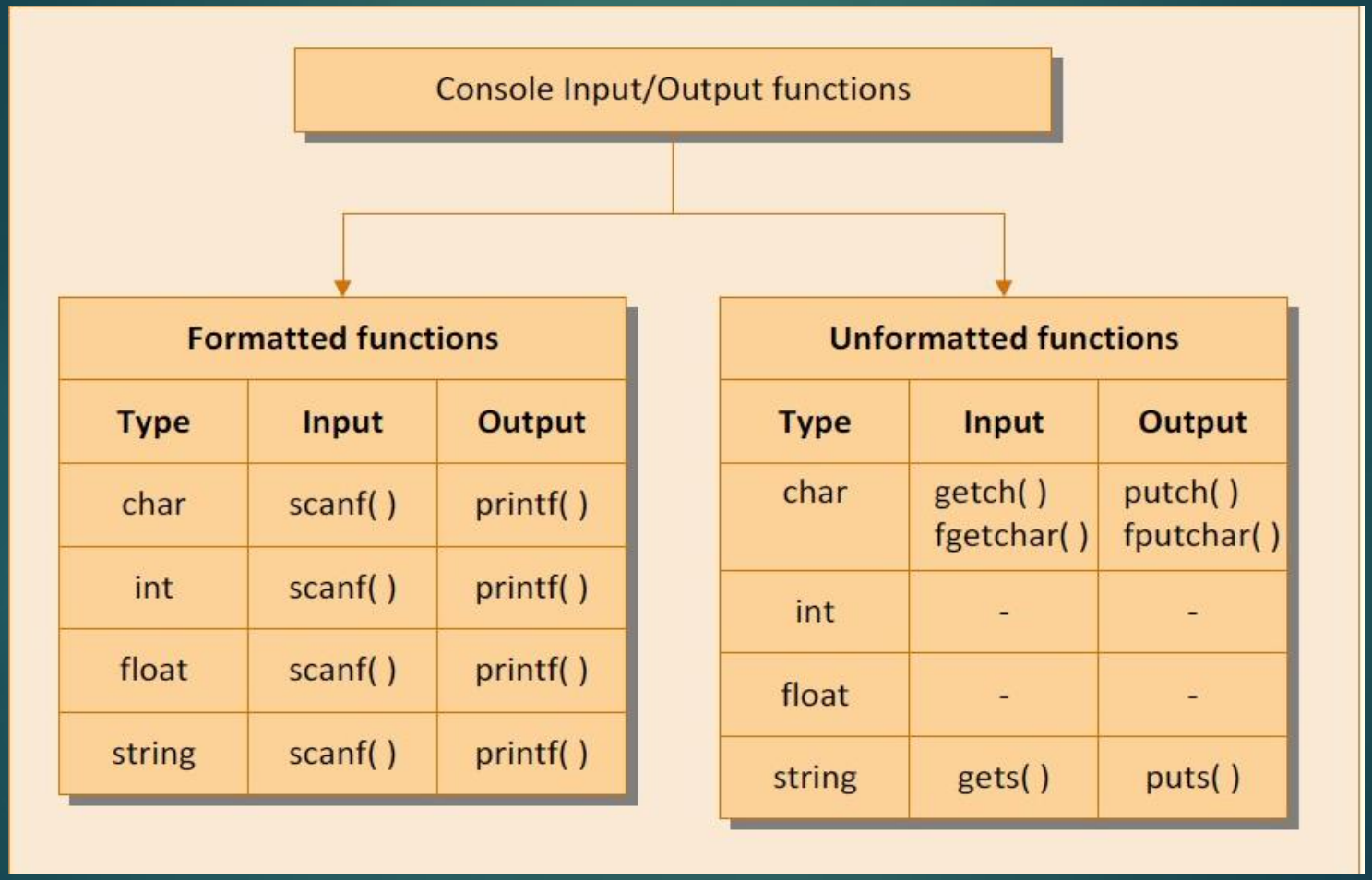
First C Program


```
/* Calculation of simple interest */
/* Author gekay Date: 25/05/2004 */
main( )
{
    int p, n ;
    float r, si ;
    p = 1000 ;
    n = 3 ;
    r = 8.5 ;
    /* formula for simple interest */
    si = p * n * r / 100 ;
    printf  ( "%f" , si ) ;
}
```

Input/Output in C

- There are numerous library functions available for I/O in C.
  a. **Console I/O functions** - Functions to receive input from keyboard and write output to VDU.
  b. **File I/O functions** -Functions to perform I/O operations on a floppy disk or hard disk.
- **Console I/O Functions**

  The screen and keyboard together are called a **console**. Console I/O functions can be further classified into two categories
  - formatted and unformatted console I/O functions.



Formatted functions

Type	Input	Output
char	scanf( )	printf( )
int	scanf( )	printf( )
float	scanf( )	printf( )
string	scanf( )	printf( )

Unformatted functions

Type	Input	Output
char	getch( ) fgetchar( )	putch( ) fputchar( )
int	-	-
float	-	-
string	gets( )	puts( )

Formatted I/O Function: printf()

- The general form of **printf()** is:

  printf("<**format string**>",<**list of variables**>);

  The **format string** can contain:

  - Characters that are simply printed as they are
  - *Conversion specifications (or Format Specifiers)* that begin with a % sign
  - *Escape sequences* that begin with a \ sign

- Some of the examples are

  - printf ("%f", si) ;
  - printf ("%d %d %f %f", p, n, r, si) ;
  - printf ("Simple interest = Rs. %f", si) ;
  - printf ("Principal = %d \nRate = %f", p, r) ;

- **Format Specifiers List:**

Data type		Format specifier
Integer	short signed short unsigned long singed long unsigned unsigned hexadecimal unsigned octal	%d or %I %u %ld %lu %x %o
Real	float double long double	%f %lf %Lf
Character	signed character unsigned character	%c %c
String		%s

optional specifiers

Specifier	Description
w	Digits specifying field width
.	Decimal point separating field width from precision (precision means number of places after the decimal point)
p	Digits specifying precision
-	Minus sign for left justifying the output in the specified field width

Example


```
# include <stdio.h>
int main( )
{
    int  weight = 63 ;
    printf ( "weight is %d kg\n", weight ) ;
    printf ( "weight is %2d kg\n", weight ) ;
    printf ( "weight is %4d kg\n", weight ) ;
    printf ( "weight is %6d kg\n", weight ) ;
    printf ( "weight is %-6d kg\n", weight ) ;
    printf ( "weight is %1d kg\n", weight ) ;
    return 0 ;
}
```

The output of the program would look like this ...

```
Columns        01234567890123456789012345678901234567890
               weight  is  63  kg
               weight  is  63  kg
               weight  is    63  kg
               weight  is      63 kg
               weight  is  63      kg
               weight  is  63  kg
```

Formatted I/O Function: scanf()

- **scanf( )** allows us to enter data from keyboard that will be formatted in a certain way. The general form is:

  scanf("<**format string**>", <**list of addresses of variables** >);

  - For e.g.,

    scanf ("%d %d %f", &p, &n, &r) ;

  - "&" is an 'Address of' operator. It gives the location number used by the variable in memory.

- Note that a blank, a tab or a new line must separate the values supplied through the keyboard .
- All the format specifications that we learnt in printf( ) function are applicable to scanf( ) function as well.

Unformatted I/O Functions

- These types of functions can deal with
  - a single character
  - and a string of characters
- **getch( )** and **getche( )** functions return the character that has been most recently typed.
- The 'e' in *getche( )* function means it echoes (displays) the character that you typed to the screen. As against this *getch( )* just returns the character that you typed without echoing it on the screen.
- **getchar( )** works similarly and echo's the character that you typed on the screen, but unfortunately requires Enter key to be typed following the character that you typed.
- **putch( )** and **putchar( )** form the other side of the coin. They print a character on the screen.

- **gets( )** receives a string from the keyboard.
- The **puts( )** function works exactly opposite to gets( ) function. It outputs a string to the screen.
- main( )

```
{
  char footballer[40] ;
  puts ( "Enter name" ) ;
  gets ( footballer ) ; /* sends base address of array */
  puts ( "Happy footballing!") ;
  puts ( footballer ) ;
}
```

- ***Unformatted console I/O functions work faster since they do not have the overheads of formatting the input or output.***

Escape Sequences:

- Escape sequences are specially sequenced characters used to format output.

```
printf("\nC you later\n");
```

- Here, backslash symbol (\) is considered as an 'escape' character.
- It causes an escape from the normal interpretation of a string, so that the next character is recognized as one having a special meaning.

Escape Seq.	Purpose	Escape Seq.	Purpose
\n	New line	\t	Tab
\b	Backspace	\r	Carriage return
\f	Form feed	\a	Alert
\'	Single quote	\"	Double quote
\\	Backslash		