INTRODUCTION
OF
COMPUTING

Format Specifier

- The format specifier is used during input and output.

- It is a way to tell the compiler what type of data is in a variable during taking input using scanf() or printing using printf().

Format Specifier

Variable Type	Keyword	Format	Size (in Bytes)
Short Integer	short	%d	2
Integer	int	%d	2 or 4
Long Integer	long	%ld	4
Floating Point	float	%f	4
Double-Precision Floating-Point	double	%f	8
Long Double-Precision Floating-Point	long double	%lf	12
Character	char	%c	1

Character Data Type

To store character data types keyword **char** is used.

Example of char data types:- ‘m’, ‘A’, ‘5’, ‘@’, ‘?’ e.t.c. Note that all are inside the single quotation.

- “a” is not a char data type because it is inside double quotation not in the single quotation.
- ‘abc’ is not a char data type because inside a single quotation only one character must be present.
- ‘’ is not a char data type, a blank single quotation is not a valid character.
- ‘ ’ is a char data type, space is inside a single quotation and space is a valid character.
- m is not a char data type, it is a variable because it is not inside a single quotation.

Character Data Type

signed char	1	-128 to 127	%c
unsigned char	1	0 to 255	%c

Integer Data Type

The keyword **int** is used to declare integer data type.

- If integer data is small then we can use keyword **short** or **short int** and
- to store long integer number we can use **long** or **long int** keyword and
- to store very long number we can use **long long** or **long long int** keyword.

Integer Data Type

Data type	Format specifier	Size (in bytes)
short int	%d	2
signed short int	%d	2
unsigned short int	%u	2
int	%d	4
signed int	%d	4
unsigned int	%u	4
long int	%ld	8
signed long int	%ld	8
unsigned long int	%lu	8
long long int	%lld	8
unsigned long long int	%llu	8

Integer Data Type

NOTE:

- Size of data type int is 2 bytes in 32-bit environment and 4 bytes in 64-bit environment.
- If a signed integer has n bits, it can contain a number between -2^{n-1} to $+(2^{n-1} - 1)$
- In the binary number system, an unsigned integer containing “n” bits can have a value between 0 to 2^n-1

Floating point data type

By default, every floating-point number is treated as a double data type. Float and long double data type are also used for floating-point.

Data type	Format Specifier	Size (in bytes)
float	%f	4
double	%f	8
long double	%Lf	16

void

- As the name indicates this type has no values. Most of the times it is used to indicate that a function does not return any value.

- Void can’t be used for storing and calculation in a program.

CHAPTER 2

Types of Instructions

"Program statements are called instruction."

Instructions are commands.

1. Data Type Declaration Instruction – This instruction is used to declare the type of variables used in a C program.

2. Arithmetic Instruction – This instruction is used to perform arithmetic operations on constants and variables.

3. Control Instruction – This instruction is used to control the sequence of execution of various statements in a C program.

Type Declaration Instruction

Ex.:

```
int bas ;
float rs, grosssal ;
char name, code ;
```

Type Declaration Instruction

There are several subtle variations of the type declaration instruction.

- While declaring the type of variable we can also initialize it as shown below.
  int i = 10, j = 25 ;
  float a = 1.5, b = 1.99 + 2.4 * 1.44 ;

Type Declaration Instruction

- The order in which we define the variables is sometimes important sometimes not. For example,
  int i = 10, j = 25 ;
  is same as
  int j = 25, i = 10 ;

  However,

  float a = 1.5, b = a + 3.1 ;
  is alright, but

  float b = a + 3.1, a = 1.5 ;
  is not. This is because here we are trying to use **a before defining it.**

Type Declaration Instruction

- The following statements would work

  int a, b, c, d ;

  a = b = c = 10 ;

  However, the following statement would not work

  int a = b = c = d = 10 ;

  Once again we are trying to use **b (to assign to a) before defining it.**

Arithmetic Instruction

Ex.:

```
int ad ;
float kot, deta, alpha, beta, gamma ;
ad = 3200 ;
kot = 0.0056 ;
deta = alpha * beta / gamma + 3.2 * 2 / 5 ;
```

Arithmetic Instruction

Integer mode arithmetic statement:

- Ex.: int i, king, issac, noteit ;
- i = i + 1 ;
- king = issac * 234 + noteit - 7689 ;

Arithmetic Instruction

Real mode arithmetic statement:

- Ex.: float qbee, antink, si, prin, anoy, roi ;
- qbee = antink + 23.123 / 4.5 * 0.3442 ;
- si = prin * anoy * roi / 100.0 ;

Arithmetic Instruction

Mixed mode arithmetic statement:

- Ex.: float si, prin, anoy, roi, avg ;
- int a, b, c, num ;
- si = prin * anoy * roi / 100.0 ;
- avg = (a + b + c + num) / 4 ;

Arithmetic Instruction

- C allows only one variable on left-hand side of **=. That is, z = k * l is** legal, whereas **k * l = z is illegal.**

- In addition to the division operator C also provides a modular division operator. This operator returns the remainder on dividing one integer with another.

- Thus the expression 10 / 2 yields 5, whereas, 10 % 2 yields 0.

- Note that the modulus operator **(%)** cannot be applied on a float.

- Also note that on using % the sign of the remainder is always same as the sign of the numerator. Thus -5 % 2 yields –1, whereas, 5 % -2 yields 1.

Arithmetic Instruction

Arithmetic operations can be performed on **ints, floats and chars.** Thus the statements,

```
char x, y ;
int z ;
x = 'a' ;
y = 'b' ;
z = x + y ;
```

Arithmetic Instruction

- No operator is assumed to be present. It must be written explicitly. In the following example, the multiplication operator after b must be explicitly written.

  a = c.d.b(xy) usual arithmetic statement

  a = c * d * b * (x * y) C statement

Arithmetic Instruction

There is no operator in C to perform exponentiation operation. Exponentiation has to be carried out as shown below:

```
# include <math.h>
# include <stdio.h>
int main( )
{
float a ;
a = pow ( 3.0, 2.0 ) ;
printf ( "%f", a ) ;
}
```

Arithmetic Instruction

Here **pow( ) function is a standard library function. It is being used** to raise 3.0 to the power of 2.0. The **pow( ) function works only** with real numbers, hence we have used 3.0 and 2.0 instead of 3 and 2.

#include <math.h> is a preprocessor directive