



# **Variables and Data Types**

---

## **Session 2**



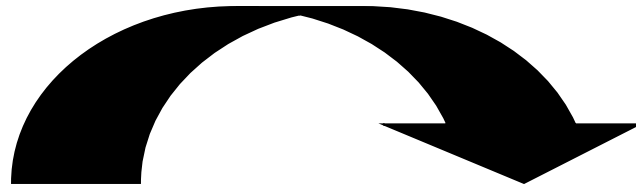
# Objectives

---

- Discuss variables
- Differentiate between variables and constants
- List the different data types and make use of them in C programs
- Discuss arithmetic operators

# Variables

Data **15**



Memory

	<b>15</b>		
	Data in memory		

**Each location in the memory is unique**

**Variables** allow to provide a meaningful **name** for the **location in memory**



# Example

```
BEGIN
    DISPLAY 'Enter 2  numbers'
    INPUT  A,  B
    C = A + B
    DISPLAY  C
END
```

- **A**, **B** and **C** are variables in the pseudo code
- Variable names takes away the need for a programmer to access memory locations using their address
- The operating system takes care of allocating space for the variables



# Constants

---

- A **constant** is a value whose worth never changes
- Examples
  - ◆ 5                      **numeric / integer constant**
  - ◆ 5.3                    **numeric / float constant**
  - ◆ "Black"                **string constant**
  - ◆ 'C'                     **character constant**
- Variables hold constant values



# Identifier Names

---

- The names of variables, functions, labels, and various other user-defined objects are called identifiers
- Some correct identifier names
  - ◆ arena
  - ◆ s\_count
  - ◆ marks40
  - ◆ class\_one
- Examples of erroneous identifiers
  - ◆ 1sttest
  - ◆ oh!god **! is invalid**
  - ◆ start... end
- Identifiers can be of any convenient length, but the number of characters in a variable name that are recognized by a compiler, varies from compiler to compiler
- Identifiers in C are case sensitive



# Guidelines for Naming Identifiers

---

Variable names should begin with an alphabet

The first character can be followed by alphanumeric characters

Proper names should be avoided while naming variables

A variable name should be meaningful and descriptive

Confusing letters should be avoided

Some standard variable naming convention should be followed while programming



# Keywords

---

- Keywords: All languages reserve certain words for their internal use
- Keywords hold a special meaning within the context of the particular language
- No problem of conflict as long as the *keyword* and the *variable name* can be distinguished. For example, having *integer* as a variable name is perfectly valid even though it contains the keyword **int**





# Data Types-1

---

- A data type describes the kind of data that will fit into a variable
- Some examples of different types of data are:
  - ◆ **Numbers**
    - Whole numbers. For example, 10 or 178993455
    - Real numbers. For example, 15.22 or 15463452.25
    - Positive numbers, negative numbers
  - ◆ **Names.** For example, John
  - ◆ **Logical values.** For example, Y or N



# Data Types-2

---

The name of the variable is preceded with the data type

```
Datatype variableName;
```

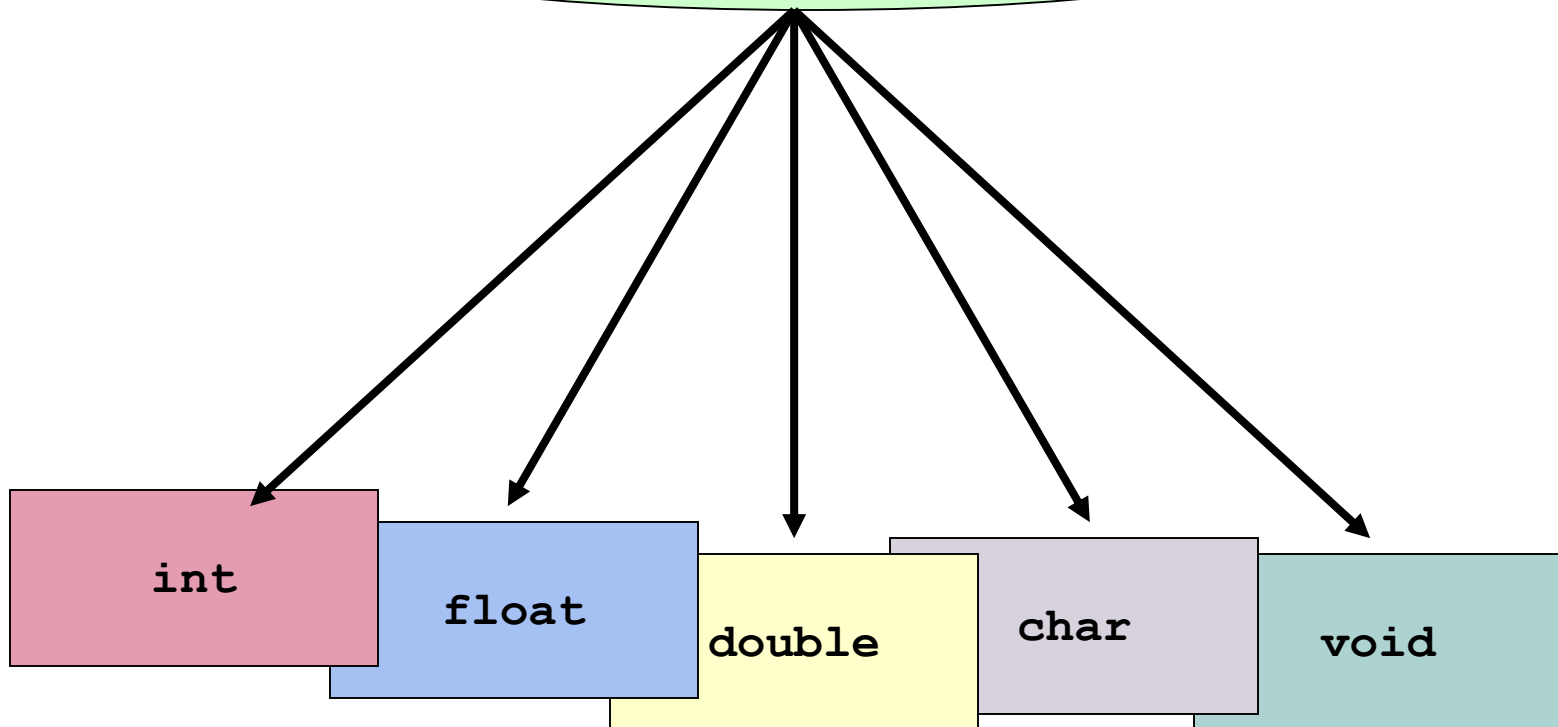
For example,

```
int total;
```



# Basic Data Types

The basic data types are





# Type **int**

---

- Stores numeric data

```
int num;  
num = 2949;
```

- Cannot then store any other type of data like "Alan" or "abc"
- 16 bits (2 bytes)
- Integers in the range -32768 to 32767
- Examples: 12322, 0, -232



# Type float

---

- Stores values containing decimal places

```
float num;  
num = 3.13;
```

- Precision of upto 6 digits
- 32 bits (4 bytes) of memory
- Examples: 23.05, 56.5, 32



# Type double

---

- Stores values containing decimal places

```
double num;  
num = 12.4679;
```

- Precision of upto 10 digits
- 64 bits (8 bytes) of memory
- Examples: 23.0545, 6.54444, -67.9087



# Type **char**

---

- Stores a single character of information

```
char gender;  
gender = 'M' ;
```

- 8 bits (1 byte) of memory
- Examples: 'a', 'm', '\$' '%', '1', '5'



# Type **void**

---

- Stores nothing
- Indicates the compiler that there is nothing to expect





# Derived Data Types

**Data type  
Modifiers**

+

**Basic Data  
types**

=

**Derived data type**

*unsigned*

+

**int**

=

**unsigned int**  
(Permits only  
positive numbers)

*short*

+

**int**

=

**short int**  
(Occupies less  
memory  
space than int)

*long*

+

**int /double**

=

**Long int /long double**  
(Occupies more  
space than  
int/double)



# signed and unsigned Types

---

- **unsigned** type specifies that a variable can take only positive values

```
unsigned int varNum;  
varNum = 56000;
```

- varNum is allocated 2 bytes
- modifier may be used with the **int** and **float** data types
- unsigned int supports range from 0 to 65535

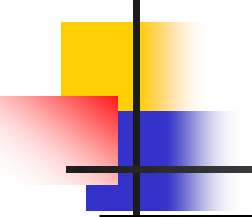


# long and short Types

---

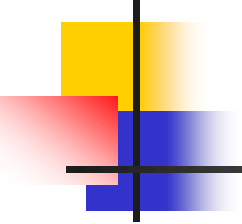
- **short int** occupies 8 bits (1 byte)
  - ◆ allows numbers in the range -128 to 127
- **long int** occupies 32 bits (4 bytes)
  - ◆ -2,147,483,648 and 2,147,483,647
- **long double** occupies 128 bits (16 bytes)

# Data Types and their range-1



Type	Approximate Size in Bits	Minimal Range
char	8	-128 to 127
signed char	8	Same as char
unsigned char	8	0 to 255
int	16	-32,768 to 32,767
signed int	16	Same as int
unsigned int	16	0 to 65,535
short int	8	-128 to 127
signed short int	8	Same as short int

# Data Types and their range-2



Type	Approximate Size in Bits	Minimal Range
unsigned short int	8	0 to 255
long int	32	-2,147,483,647 to 2,147,483,647
signed long int	32	Same as long int
unsigned long int	32	0 to 4,294,967,295
float	32	Six digits of precision
double	64	Ten digits of precision
long double	128	Ten digits of precision



# Sample Declaration

---

```
main ()
{
    char abc;      /*abc of type character */
    int xyz;   /*xyz of type integer */
    float length; /*length of type float */
    double area; /*area of type double */
    long liteyrs; /*liteyrs of type long int */
    short arm;    /*arm of type short integer*/
}
```