

C Fundamentals

HoangND1

- ❑ History of C
- ❑ Application Areas Of C
- ❑ About C & C Program Structure
- ❑ First C Program
- ❑ Discuss variables
- ❑ Differentiate between variables and constants
- ❑ List the different data types and make use of them in C programs
- ❑ Discuss arithmetic operators

□ History of C

- ✓ Evolved from two other programming languages
 - BCPL and B
 - “Typeless” languages
- ✓ Dennis Ritchie (Bell Laboratories)
 - Added data typing, other features
- ✓ Development language of UNIX
- ✓ Hardware independent
 - Portable programs
- ✓ 1989: ANSI standard
- ✓ 1990: ANSI and ISO standard published
 - ANSI/ISO 9899: 1990

Application Areas Of C

- ❑ C was initially used for systems programming
- ❑ A system program forms a portion of the operating system of the computer or its support utilities
- ❑ Operating Systems, Interpreters, Editors, Assembly programs are usually called system programs
- ❑ The UNIX operating system was developed using C
- ❑ There are C compilers available for almost all types of PC's

About C & C Program Structure

- ❑ C has 32 keywords
- ❑ These keywords combined with a formal syntax form a C programming language
- ❑ Rules to be followed for all programs written in C:
 - ✓ All keywords are lowercased
 - ✓ C is case sensitive, do while is different from DO WHILE
 - ✓ Keywords cannot be used as a variable or function name

```
main()  
{  
  /* This is a sample Program*/  
  int i,j;  
  i=100;  
  j=200;  
  :  
}
```

First C Program

Source codes

```

1  // hello.c
2  // A first program
3  #include <stdio.h>
4
5  // function main begins pro
6  int main()
7  {
8      printf("Hello world!\n");
9
10     return 0; // indicate that program ended successfully
11
12 } // end function main

```

Single-line comments.

Function main returns an

Left brace { begins function body.

ive to

Function main appears exactly once in every C/C++ program..

Statements end with a semicolon ;.

Corresponding right brace } ends function body.

Name printf belongs to namespace stdio

Keyword return is one of several means to exit function; value 0 indicates program terminated successfully.

Welcome to C++!

First C Program

Anatomy of a C Program

program header comment

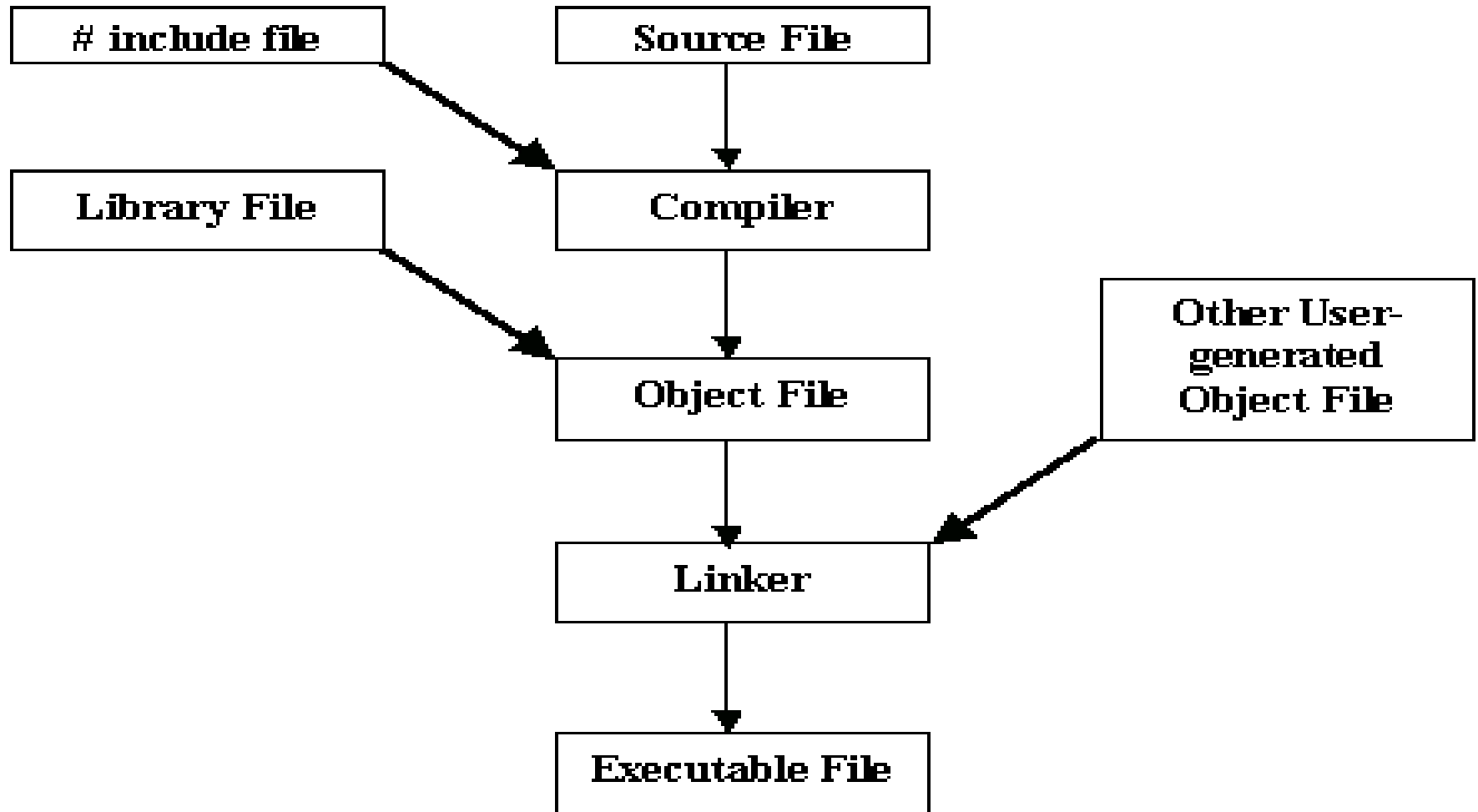
preprocessor directives (if any)

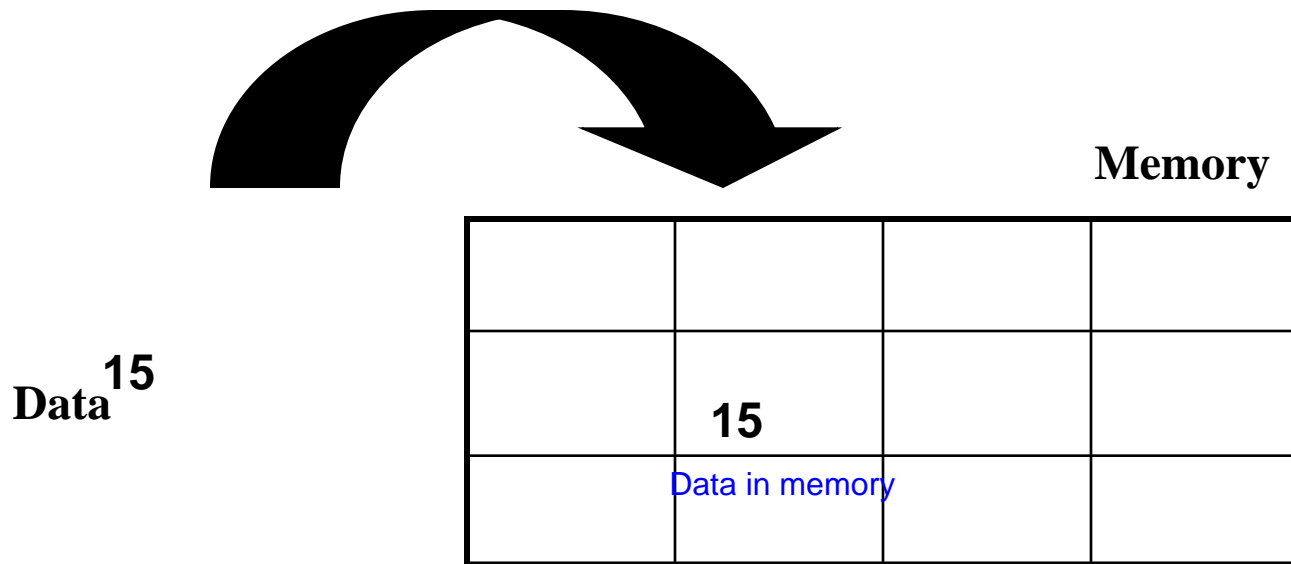
```
int main ( void )  
{  
    statement(s)  
    return 0 ;  
}
```

- ❑ A **comment** is descriptive text used to help a reader of the program understand its content.
- ❑ All comments must begin with the characters `/*` and end with the characters `*/`
- ❑ These are called **comment delimiters**
- ❑ The program header comment always comes first.
- ❑ Look at the class web page for the required contents of our header comment.

- ❑ Lines that begin with a # in column 1 are called **preprocessor directives (commands)**.
- ❑ Example: the **#include <stdio.h>** directive causes the preprocessor to include a copy of the standard input/output header file **stdio.h** at this point in the code.
- ❑ This header file was included because it contains information about the **printf ()** function that is used in this program.

- ❑ Download & Install Dev-C++ from <http://www.bloodshed.net/dev/devcpp.html>
- ❑ Run Dev-C++ and discover its features
- ❑ Creating a Program: Ctrl+N
- ❑ Compiling a Program: Ctrl+F9
- ❑ Running a Program: F9





Each location in the memory is unique

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

- ❑ 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

- ❑ Numeric constants are an uninterrupted sequence of digits (and may contain a period). They never contain a comma.
- ❑ Examples:
 - ✓ 123
 - ✓ 98.6
 - ✓ 1000000

- ❑ Singular!
- ❑ One character defined character set.
- ❑ Surrounded on the single quotation mark.
- ❑ Examples:
 - ✓ 'A'
 - ✓ 'a'
 - ✓ '\$'
 - ✓ '4'

- ❑ A sequence characters surrounded by double quotation marks.
- ❑ Considered a single item.
- ❑ Examples:
 - ✓ "UMBC"
 - ✓ "I like ice cream."
 - ✓ "123"
 - ✓ "CAR"
 - ✓ "car"

- ❑ Some correct identifier names
 - ✓ arena
 - ✓ s_count
 - ✓ marks40
 - ✓ class_one

- ❑ Examples of erroneous identifiers
 - ✓ 1sttest
 - ✓ oh!god
 - ✓ start... End

- ❑ 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

- Different types of data are stored in variables. Some examples 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

A data type describes the kind of data that will fit into a variable

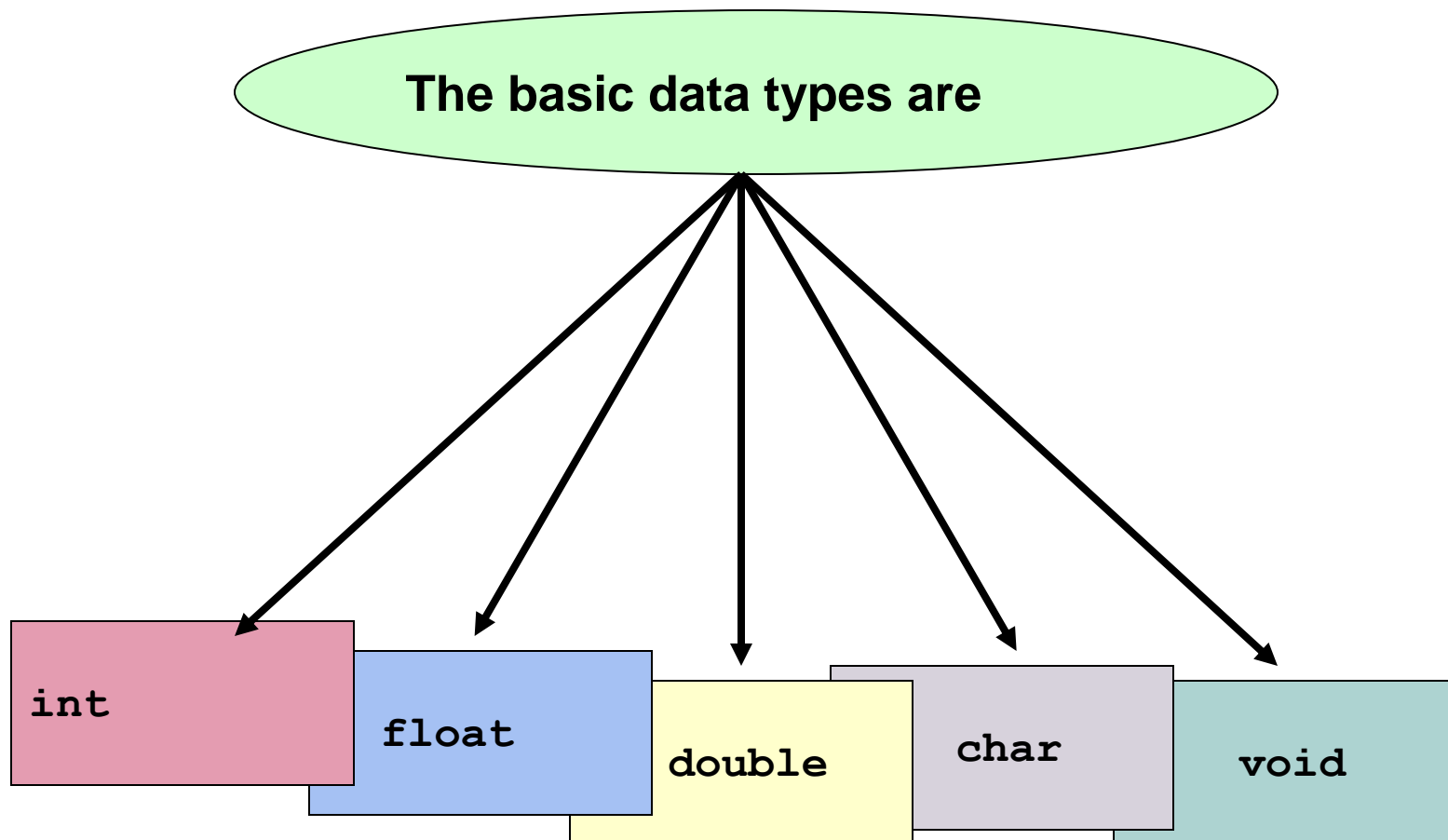
The name of the variable is preceded with the data type

For example, the data type `int` would precede the name `varName`

```
Datatype variableName
```

```
int varName
```

Basic Data Types



- ❑ Stores numeric data

`int num;`

- ❑ 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

- ❑ Stores values containing decimal places

`float num;`

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

- ❑ Stores values containing decimal places

`double num;`

- ❑ Precision of upto 10 digits
- ❑ 64 bits (8 bytes) of memory
- ❑ Examples: 2.0, 3.55, 100000

- Stores a single character of information

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

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

- ❑ 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 /longdouble (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=23123;
```

- ❑ 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)
 - ✓ -2,147,483,647 and 2,147,483,647
- ❑ **long double** occupies 128 bits (16 bytes)

Data Types and their range-1

The size of data type depends Operating System

Sizeof (data type) function return the size of data type

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

Data Types and their range-2

Type	Approximate Size in Bits	Minimal Range
signed short int	8	Same as short int
long int	32	-2,147,483,647 to 2,147,483,647
signed long int	32	0 to 4,294,967,295
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*/
}
```


Q & A