BECE320E & Embedded C Programming

Dr. G. Sudhakaran
SENSE, VIT

Introduction

- C is a procedural programming language initially developed by Dennis Ritchie in the year 1972 at Bell Laboratories of AT&T Labs.
- It was mainly developed as a system programming language to write the UNIX operating system.
- **The main features of the C language include:**
  - General Purpose and Portable
  - Low-level Memory Access
  - Fast Speed
  - Clean Syntax

History of C

1960	• ALGOL	International Group
1967	• BCPL	Martin Richards
1970	• B	Ken Thompson
1972	• Traditional C	Dennis Ritchie
1978	• K&R C	Brain Kernighan and Dennis Ritchie
1989	• ANSI C	ANSI Committee
1990	• ANSI/ISO C	ISO Committee

Why Should We Learn C?

- Many later languages have borrowed syntax/features directly or indirectly from the C language.
- Like syntax of Java, PHP, JavaScript, and many other languages are mainly based on the C language.
- C++ is nearly a superset of C language (Only a few programs may compile in C, but not in C++).
- So, if a person learns C programming first, it will help him to learn any modern programming language as well.
- As learning C help to understand a lot of the underlying architecture of the operating system. Like pointers, working with memory locations, etc.

Features of C Language

- Modularity
- Extensibility
- Elegant syntax
- Case sensitive
- Less memory required
- The standard library concept
- The portability of the compiler
- A powerful and varied range of operators
- Ready access to the hardware when needed

Structure of C Program

```
Documentation section
        (Used for comments)
Link section

Definition section

Global declaration section
(Variables used in more than
one functions)
void main ()
{
        Declaration part
        Executable part
}
Subprogram section
        (User defined functions)
```

Program

```c
// Program for printing a number




void fun();

int a=10;


void main( )
 {
  printf("Value of a inside main function: %d", a);
  fun();
 }

void fun()
{printf("Value of a inside fun function: %d", a);}
```

Comments

- A comment is an explanation or description of the source code of the program
- It helps a programmer to explain logic of the code and improves program readability.
- At run-time, a comment is ignored by the compiler.
- There are two types of comments in C:
  - Single line comment
    - Represented as // double forward slash
    - It is used to denote a single line comment only.
    - Example: `// Single line comment`
  - Multi-line comment
    - Represented as /* any_text */ start with forward slash and asterisk (/*) and end with asterisk and forward slash (*/).
    - It is used to denote single as well as multi-linecomment.
    - Example:
```
/* multi line comment line -1
   multi line comment line -2 */
```

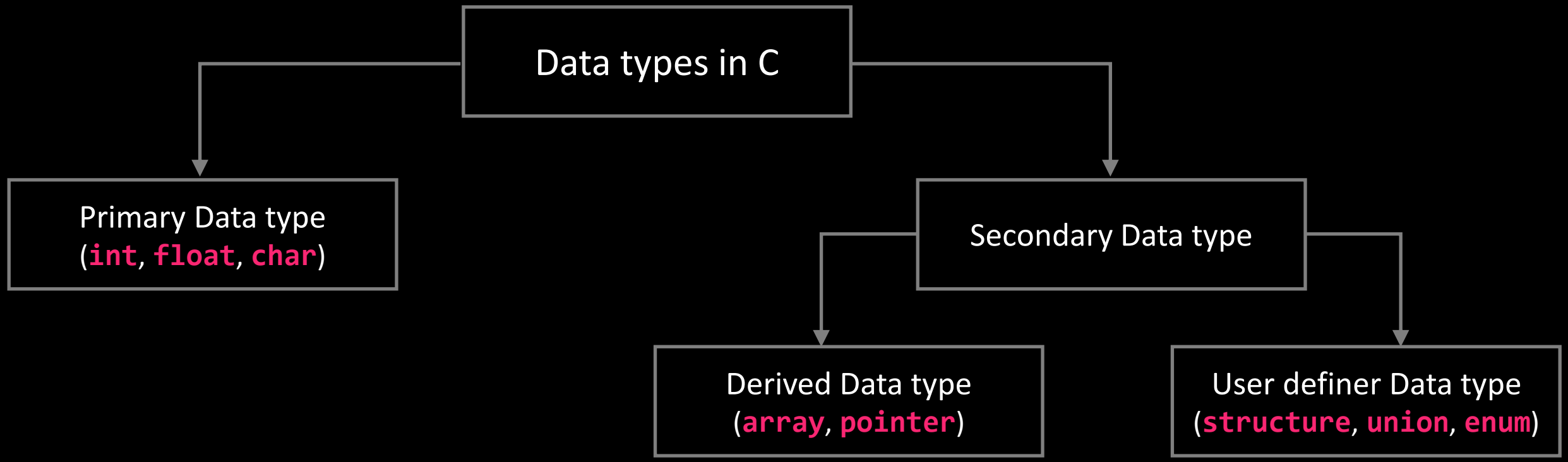
Header files

- A header file is a file with extension .h which contains the set of predefined standard library functions.
- The "#include" preprocessing directive is used to include the header files with extension in the program.

Header file	Description
stdio.h	Input/Output functions (printf and scanf)
conio.h	Console Input/Output functions (getch and clrscr)
math.h	Mathematics functions (pow, exp, sqrt etc…)
string.h	String functions (strlen, strcmp, strcat etc…)

Data Types

- Data types are defined as the data storage format that a variable can store a data.
- It determines the type and size of data associated with variables.



Primary Data Type

- Primary data types are built in data types which are directly supported by machine.
- They are also known as fundamental data types.
  - **int**:
    - **int** datatype can store integer number which is whole number without fraction part such as 10, 105 etc.
    - C language has 3 classes of integer storage namely **short int, int** and **long int**. All of these data types have signed and unsigned forms.
    - Example: **int** a=10;
  - **float**:
    - **float** data type can store floating point number which represents a real number with decimal point and fractional part such as 10.50, 155.25 etc.
    - When the accuracy of the floating point number is insufficient, we can use the **double** to define the number. The double is same as float but with longer precision.
    - To extend the precision further we can use **long double** which consumes 80 bits of memory space.
    - Example: **float** a=10.50;

Primary Data Type (cont…)

- **char**:
  - **Char** data type can store single character of alphabet or digit or special symbol such as ‘a’, ‘5’ etc.
  - Each character is assigned some integer value which is known as ASCII values.
  - Example: **char** a=‘a’;
- **void**:
  - The **void** type has no value therefore we cannot declare it as variable as we did in case of **int** or **float** or **char**.
  - The **void** data type is used to indicate that function is not returning anything.

Secondary Data Type

- Secondary data types are not directly supported by the machine.
- It is combination of primary data types to handle real life data in more convenient way.
- It can be further divided in two categories,
  - Derived data types: Derived data type is extension of primary data type. It is built-in system and its structure cannot be changed. Examples: Array and Pointer.
    - Array: An array is a fixed-size sequenced collection of elements of the same data type.
    - Pointer: Pointer is a special variable which contains memory address of another variable.
  - User defined data types: User defined data type can be created by programmer using combination of primary data type and/or derived data type. Examples: Structure, Union, Enum.
    - Structure: Structure is a collection of logically related data items of different data types grouped together under a single name.
    - Union: Union is like a structure, except that each element shares the common memory.

Variables and Constants

- Variable is a symbolic name given to some value which can be changed.
- x, y, a, $count$, etc. can be variable names.
- $x=5 \quad a=b+c$
- Constant is a fixed value which cannot be changed.
- 5, –7.5, 1452, 0, 3.14, etc.

Tokens

- The smallest individual unit of a program is known as token.
- C has the following tokens:
  - Keywords
    - C reserves a set of 32 words for its own use. These words are called keywords (or reserved words), and each of these keywords has a special meaning within the C language.
  - Identifiers
    - Identifiers are names that are given to various user defined program elements, such as variable, function and arrays.
  - Constants
    - Constants refer to fixed values that do not change during execution of program.
  - Strings
    - A string is a sequence of characters terminated with a null character \0.
  - Special Symbols
    - Symbols such as #, &, =, * are used in C for some specific function are called as special symbols.
  - Operators
    - An operator is a symbol that tells the compiler to perform certain mathematical or logical operation.

Operators

- Arithmetic operators (+, - , *, /, %)
- Relational operators (<, <=, >, >=, ==, !=)
- Logical operators (&&, ||, !)
- Assignment operators (+=, -=, *=, /=)
- Increment and decrement operators (++, --)
- Conditional operators (?:)
- Bitwise operators (&, |, ^, <<, >>)
- Special operators ()

Arithmetic Operators

- Arithmetic operators are used for mathematical calculation.

Operator	Meaning	Example	Description
+	Addition	a + b	Addition of a and b
-	Subtraction	a – b	Subtraction of b from a
*	Multiplication	a * b	Multiplication of a and b
/	Division	a / b	Division of a by b
%	Modulo division- remainder	a % b	Modulo of a by b

Relational Operators

- Relational operators are used to compare two numbers and taking decisions based on their relation.
- Relational expressions are used in decision statements such as if, for, while, etc…

Operator	Meaning	Example	Description
<	Is less than	a < b	a is less than b
<=	Is less than or equal to	a <= b	a is less than or equal to b
>	Is greater than	a > b	a is greater than b
>=	Is greater than or equal to	a >= b	a is greater than or equal to b
==	Is equal to	a = b	a is equal to b
!=	Is not equal to	a != b	a is not equal to b

Logical Operators

- Logical operators are used to test more than one condition and make decisions.

Operator	Meaning
&&	logical AND (Both non zero then true, either is zero then false)
||	logical OR (Both zero then false, either is non zero then true)

a	b	a&&b	a||b
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Assignment Operators

- Assignment operators (=) is used to assign the result of an expression to a variable.
- Assignment operator stores a value in memory.
- C also supports shorthand assignment operators which simplify operation with assignment.

Operator	Meaning
=	Assigns value of right side to left side
+=	a += 1 is same as a = a + 1
-=	a -= 1 is same as a = a - 1
*=	a *= 1 is same as a = a * 1
/=	a /= 1 is same as a = a / 1
%=	a %= 1 is same as a = a % 1

Increment and Decrement Operators

- Increment (++) operator used to increase the value of the variable by one.
- Decrement (--) operator used to decrease the value of the variable by one.

Example

```
x=100;
x++;
```

Explanation

```
After the execution the
value of x will be 101.
```

Example

```
x=100;
x--;
```

Explanation

```
After the execution the
value of x will be 99.
```

Increment and Decrement Operators (cont…)

Operator	Description
Pre increment operator (++x)	value of x is incremented before assigning it to the variable on the left

Example

```
x=10;
p=++x;
```

Explanation

```
First increment value of
x by one then assign.
```

Output

```
x will be 11
p will be 11
```

Operator	Description
Post increment operator (x++)	value of x is incremented after assigning it to the variable on the left

Example

```
x=10;
p=x++;
```

Explanation

```
First assign value of x
then increment value.
```

Output

```
x will be 11
p will be 10
```

Conditional Operators

- A ternary operator is known as conditional operator.
- Syntax: *exp1 ? exp2 : exp3*

Working of the ? : Operator

```
exp1 is evaluated first
if exp1 is true(nonzero) then
        - exp2 is evaluated and its value becomes the value of the expression
If exp1 is false(zero) then
        - exp3 is evaluated and its value becomes the value of the expression
```

Example

```
m=2, n=3;
r=(m>n) ? m : n;
```

Explanation

```
Value of r will be 3
```

Example

```
m=2, n=3;
r=(m<n) ? m : n;
```

Explanation

```
Value of r will be 2
```

Bitwise Operators

- Bitwise operators are used to perform operation bit by bit.
- Bitwise operators may not be applied to float or double.

Operator	Meaning
&	bitwise AND
|	bitwise OR
^	bitwise exclusive OR
<<	shift left (shift left means multiply by 2)
>>	shift right (shift right means divide by 2)

Bitwise Operators

```
8 = 1000 (In Binary) and 6 = 0110 (In Binary)
```

Example: Bitwise & (AND)

```
int a=8, b=6, c;
c = a & b;
printf("Output = %d", c);
```

Output

```
0
```

Example: Bitwise | (OR)

```
int a=8, b=6, c;
c = a | b;
printf("Output = %d", c);
```

Output

```
14
```

Example: Bitwise << (Shift Left)

```
int a=8, b;
b = a << 1;
printf("Output = %d", b);
```

Output

```
16 (multiplying a by a power of two)
```

Example: Bitwise >> (Shift Right)

```
int a=8, b;
b = a >> 1;
printf("Output = %d", b);
```

Output

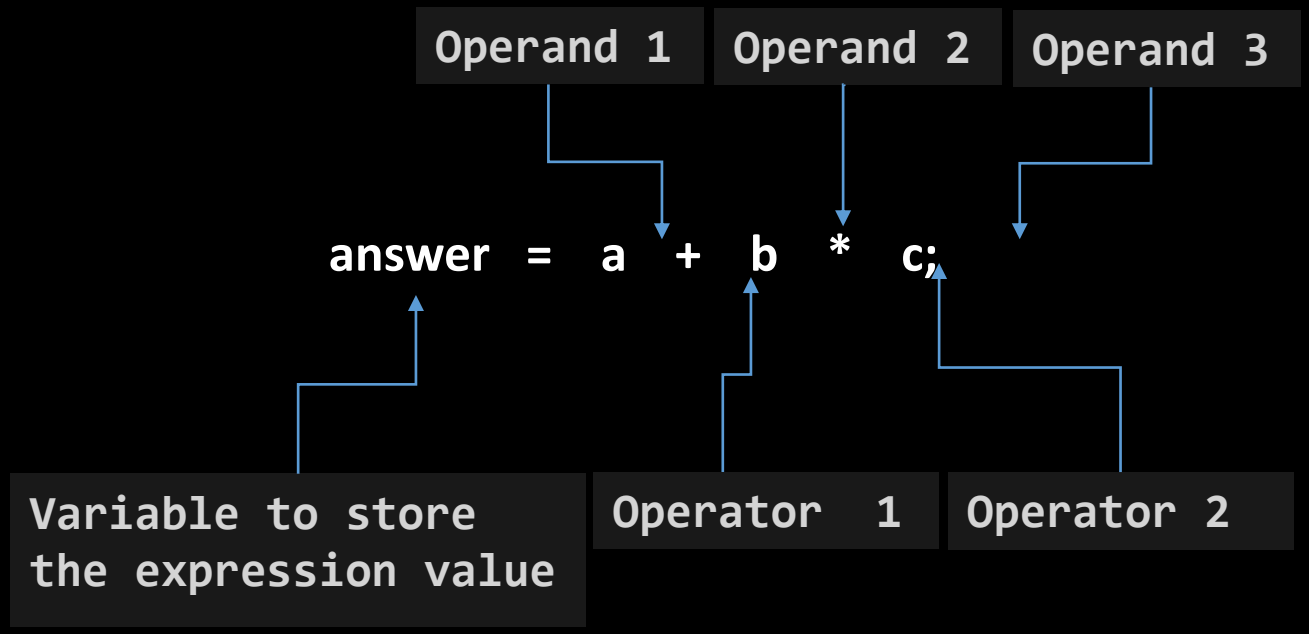
```
4 (dividing a by a power of two)
```

Special Operators

Operator	Meaning
&	Address operator, it is used to determine address of the variable.
*	Pointer operator, it is used to declare pointer variable and to get value from it.
,	Comma operator. It is used to link the related expressions together.
sizeof	It returns the number of bytes the operand occupies.
.	member selection operator, used in structure.
->	member selection operator, used in pointer to structure.

Expressions

- An expression is a combination of operators, constants and variables.
- An expression may consist of one or more operands, and zero or more operators to produce a value.



Evaluation of Expressions

- An expression is evaluated based on the operator precedence and associativity.
- When there are multiple operators in an expression, they are evaluated according to their precedence and associativity.

Operator precedence

- Precedence of an operator is its priority in an expression for evaluation.
- The operator with higher precedence is evaluated first and the operator with the least precedence is evaluated last.
- Operator precedence is why the expression 5 + 3 * 2 is calculated as 5 + (3 * 2), giving 11, and not as (5 + 3) * 2, giving 16.
- We say that the multiplication operator (*) has higher "precedence" or "priority" than the addition operator (+), so the multiplication must be performed first.

Operator associativity

- Associativity is the left-to-right or right-to-left order for grouping operands to operators that have the same precedence.
- Operator associativity is why the expression 8 - 3 - 2 is calculated as (8 - 3) - 2, giving 3, and not as 8 - (3 - 2), giving 7.
- We say that the subtraction operator (-) is "left associative", so the left subtraction must be performed first.
- When we can't decide by operator precedence alone in which order to calculate an expression, we must use associativity.

Type conversion

- Type conversion is converting one type of data to another type.
- It is also known as Type Casting.
- There are two types of type conversion:
  - Implicit Type Conversion
    - This type of conversion is usually performed by the compiler when necessary without any commands by the user.
    - It is also called Automatic Type Conversion.
  - Explicit Type Conversion
    - These conversions are done explicitly by users using the pre-defined functions.

Example: Implicit Type Conversion

```
int a = 20;
double b = 20.5;
printf("%lf", a + b);
```

Output

```
40.500000
```

Example: Explicit Type Conversion

```
double a = 4.5, b = 4.6, c = 4.9;
int result = (int)a + (int)b + (int)c;
printf("result = %d", result);
```

Output

```
12
```

printf()

- printf() is a function defined in stdio.h file
- It displays output on standard output, mostly monitor
- Message and value of variable can be printed
- Let's see few examples of printf
  - printf(" ");
  - printf("Hello World"); // Hello World
  - printf("%d", c); // 15
  - printf("Sum = %d", c); // Sum = 15
  - printf("%d+%d=%d", a, b, c); // 10+5=15

scanf()

- scanf() is a function defined in stdio.h file
- scanf() function is used to read character, string, numeric data from keyboard
- Syntax of scanf
  - `scanf("%X", &variable);`
    - where %X is the format specifier which tells the compiler what type of data is in a variable.
    - & refers to address of "variable" which is directing the input value to a address returned by &variable.

Format specifier	Supported data types	Example	Description
%d	Integer	scanf("%d", &a)	Accept integer value such as 1, 5, 25, 105 etc
%f	Float	scanf("%f", &b)	Accept floating value such as 1.5, 15.20 etc
%c	Character	scanf("%c", &c)	Accept character value such as a, f, j, W, Z etc
%s	String	scanf("%s", &d)	Accept string value such as diet, india etc

getchar and putchar

- getchar function reads a single character from terminal.
- putchar function displays the character passed to it on the screen.

Program

```c
#include <stdio.h>
void main( )
{
    int c;
    printf("Enter a character: ");
    /* Take a character as input */
    c = getchar();
    /* Display the character */
    printf("Entered character is: ");
    putchar(c);
}
```

Output

```
Enter a character: a
Entered character is: a
```

gets and puts

- gets function reads a line from stdin into the buffer pointed to by s until either a terminating newline or EOF (End of File) occurs.
- puts function writes the string 's' and 'a' trailing newline to stdout.

Program

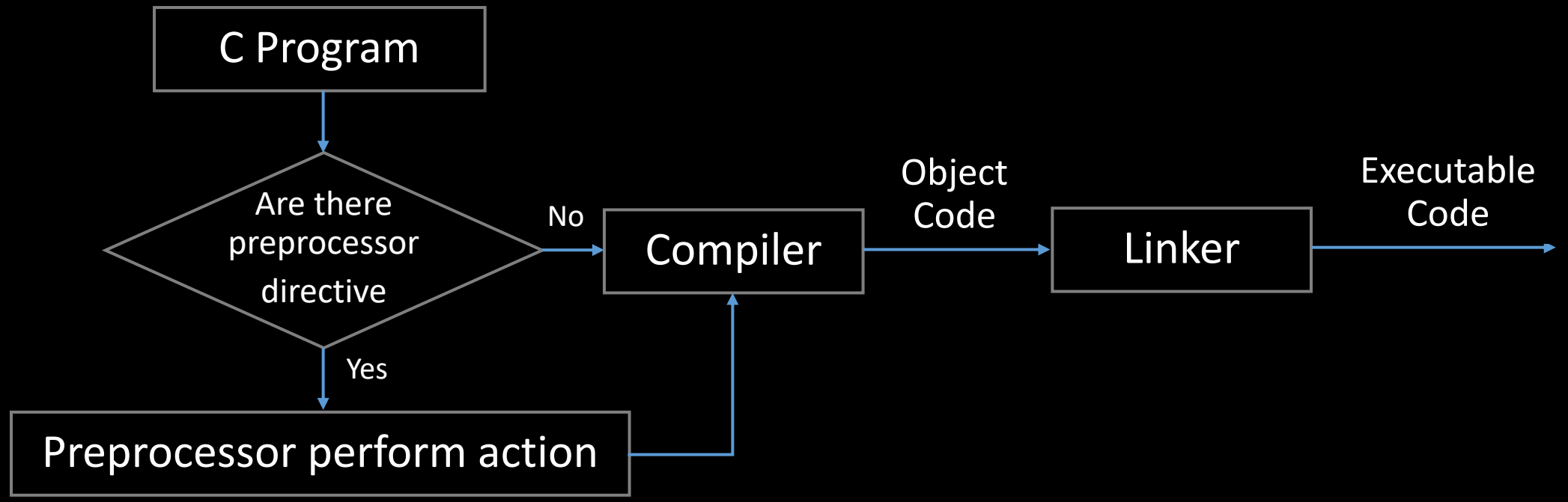
```c
#include <stdio.h>
void main( )
{
    /*Character array of length 100*/
    char str[100];
    printf("Enter a string: ");
    /* Take a string as input */
    gets( str );
    /* Display the string */
    printf("Entered string is: ");
    puts( str );
}
```

Output

```
Enter a string: india
Entered string is: india
```

Preprocessor

- Preprocessors are programs that process our source code before compilation.
- There are a number of steps involved between writing a program and executing a program in C.
- Let us have a look at these steps before we actually start learning about Preprocessors.



Types of Preprocessor

- There are 4 main types of preprocessor directives:
  - Macros
  - File inclusion
  - Conditional compilation
  - Other directives

Macro

- A macro is a fragment of code which has been given a name. Whenever the name is used in program, it is replaced by the contents of the macro.
- Macro definitions are not variables and cannot be changed by your program code like variables.
- The '#define' directive is used to define a macro.
- Do not put a semicolon (;) at the end of #define statements.
- There are two types of macros:
  - Object-like Macros
  - Function-like Macros

Macro

Description	Object-like Macros	Function-like Macros
Definition	The object-like macro is an identifier that is replaced by value.	The function-like macro looks like function call.
Use	It is used to represent numeric constants.	It is used to represent function.
Syntax	#define CNAME value	#define CNAME (expression)
Example	#define PI 3.14	#define MIN(a,b) ((a)<(b)?(a):(b))
Program		

```c
#include <stdio.h>
#define PI 3.14
void main()
{   int r=2;
    float a;
    a=PI*r*r;
    printf("%f", a);
}
```

```c
#include <stdio.h>
#define MIN(a,b) ((a)<(b)?(a):(b))
void main()
{
    printf("%d", MIN(2, 5));
}
```