PROGRAMMING IN C

Chapter 1: Basics of C

Study Notes for Engineering / Polytechnic Students

Topics Covered

1.1 History, Advantages of Structured Programming, Files used in C, Characteristics of C

1.2 Character Set, Tokens, Constants, Variables, Keywords, Data Types

1.3 Operators, Precedence, Associativity, Type Conversion, Typecasting

1.4 Formatted Input and Output

Poly Notes Hub

1.1 History of C, Advantages of Structured Programming, Files Used in C, Characteristics of C

History of C

C is a general-purpose, procedural programming language developed by Dennis Ritchie at Bell Laboratories (AT&T) between 1969 and 1973. It evolved from two earlier languages, BCPL (Basic Combined Programming Language) developed by Martin Richards and B, a simplified version of BCPL created by Ken Thompson. C was originally designed to rewrite the UNIX operating system, which had been written in assembly language, so that it could be ported easily to different machines.

In 1978, Brian Kernighan and Dennis Ritchie published the book "The C Programming Language", which became the informal specification of the language and is often referred to as K&R C. To remove ambiguities and add new features, the American National Standards Institute (ANSI) standardised the language in 1989, giving rise to ANSI C or C89. This was adopted internationally by ISO in 1990 as C90. Later revisions, C99, C11, C17 and C23, added further features such as inline functions, variable-length arrays and improved type support.

C is often called the "mother of modern programming languages" because many later languages such as C++, Java, C#, Python and JavaScript borrow heavily from its syntax and structure.

Advantages of Structured Programming

Structured programming is a programming paradigm that breaks a program into smaller, manageable, logically organised blocks or functions, using constructs such as sequence, selection (if-else) and iteration (loops), instead of relying on unconditional jumps (goto). C strongly supports this style. Its advantages include:

- Improved readability: the program logic can be followed from top to bottom without jumping unpredictably between sections.
- Easier debugging and testing: since the program is divided into independent functions/modules, errors can be isolated and corrected without affecting the rest of the program.
- Code reusability: functions written for one program can be reused in other programs, saving development time.
- Easier maintenance: modifying or upgrading a structured program is simpler because each module performs a specific, well-defined task.
- Better team collaboration: different programmers can work on different functions/modules of the same project simultaneously.
- Reduced complexity: large problems are decomposed into smaller sub-problems, making them easier to understand and solve (a technique known as top-down design).

Files Used in C

When a C program is developed and executed, it passes through several stages, and at each stage a different type of file is produced:

File Type	Extension	Description
Source file	.c	Contains the actual C program code written by the programmer in plain text, human-readable form.

Poly Notes HUB

File Type	Extension	Description
Header file	.h	Contains function declarations (prototypes), macros and definitions that are included into a source file using the #include directive (e.g. stdio.h, math.h).
Object file	.o / .obj	Generated by the compiler after compiling the source file; contains machine code along with unresolved references to library/other functions, but is not yet a complete executable.
Binary executable file	.exe (Windows) / a.out (Linux)	Produced by the linker after combining the object file(s) with required library code; this is the final program that can be directly run by the operating system.

The overall process is: Source code (.c) → Preprocessor (expands headers/macros) → Compiler (translates to object code, .o) → Linker (combines object files and libraries) → Executable file (.exe / a.out).

Characteristics of C

- Middle-level language: combines the features of a high-level language (readability, portability) with low-level features (direct memory/hardware access through pointers).
- Structured language: supports functions, blocks and structured control statements, enabling top-down modular design.
- Portability: a C program written on one machine/OS can be compiled and run on another with little or no modification.
- Rich set of operators and built-in functions: supports arithmetic, logical, relational, bitwise and other operators along with an extensive standard library.
- Fast and efficient: being close to hardware and having a small runtime, C programs execute quickly and use memory efficiently.
- Extensible: new features can be added through user-defined functions and library inclusion.
- Case-sensitive: C treats uppercase and lowercase letters as distinct (e.g. Sum and sum are different identifiers).
- Simple and small in size: it has only 32 keywords, keeping the core language compact.
- Used to develop system software: operating systems (UNIX, parts of Windows/Linux), compilers, and embedded systems are commonly written in C.

Poly Notes Hub

1.2 C Character Set, Tokens, Constants, Variables, Keywords, Data Types

C Character Set

The character set is the set of valid characters that a C compiler can recognise and use to form words, statements and expressions in a program. It includes:

- Letters: uppercase A–Z and lowercase a–z.
- Digits: 0–9.
- Special characters: such as , . ; : ? ' " ! | / \ ~ _ $ % & ^ * ( ) [ ] { } < > + - = # @.
- White spaces: blank space, horizontal tab, newline, carriage return, and form feed, used as separators between tokens.

Tokens

A token is the smallest individual unit of a C program that the compiler can recognise; a C program is essentially a sequence of such tokens. C has six main types of tokens:

1. Keywords – reserved words with predefined meaning (e.g. int, for, return).
2. Identifiers – names given to variables, functions, arrays, etc. (e.g. total, main, myFunc).
3. Constants – values that do not change during program execution (e.g. 10, 3.14, 'A').
4. Strings – a sequence of characters enclosed in double quotes (e.g. "Hello").
5. Operators – symbols that perform operations on operands (e.g. +, -, *, ==).
6. Special symbols/punctuators – such as (), {}, [], ;, and , used to organise the program's structure.

Constants

A constant is a fixed value that cannot be altered by the program during its execution. C constants are classified as follows:

- Integer constants: whole numbers without a decimal point, e.g. 25, -10, 0. They may be written in decimal, octal (prefix 0) or hexadecimal (prefix 0x/0X).
- Floating-point (real) constants: numbers containing a decimal point or expressed in exponential form, e.g. 12.5, 3.14, 2.5e3.
- Character constants: a single character enclosed in single quotes, e.g. 'A', '9', '$'.
- String constants: a sequence of characters enclosed in double quotes, e.g. "Engineering".
- Escape sequences: special backslash-prefixed character constants representing non-printable actions, e.g. \n (newline), \t (tab), \\ (backslash), \' (single quote), \" (double quote), \0 (null character).

Variables

A variable is a named location in memory that is used to store a data value which may change during program execution. Every variable in C must be declared with a specific data type before it is used, which tells the compiler how much memory to allocate and what kind of value it will hold.

Rules for naming a variable (identifier) in C:

- It must begin with a letter (A–Z, a–z) or an underscore (_); it cannot start with a digit.

Poly Notes Hub

- It can be followed by any combination of letters, digits, and underscores.
- No special characters or spaces are allowed (e.g. @, #, % are not permitted).
- A keyword cannot be used as a variable name.
- C is case-sensitive, so Marks and marks are treated as two different variables.
- There is generally no restriction on length, though only the first 31 characters may be treated as significant by some compilers.

```
int age;
float salary;
char grade = 'A';
int rollNo = 101, marks = 89;
```

Keywords

Keywords are reserved words that have a fixed, predefined meaning in the C language. They cannot be used as identifiers (variable names, function names, etc.). Standard C (C89) defines 32 keywords, including:

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

Data Types Used in C

A data type specifies the type of value a variable can hold and how much memory it occupies. C data types are broadly classified as follows:

- Primary (fundamental) data types: int (integer), char (character), float (single-precision floating point), double (double-precision floating point), and void (represents absence of a value, e.g. a function returning nothing).
- Derived data types: built from primary types — arrays, pointers, functions, and structures/unions (structures/unions are also sometimes classified as user-defined).
- User-defined data types: created by the programmer using struct, union, enum and typedef.

Type qualifiers such as signed, unsigned, short and long can modify the range and storage size of the primary integer/character types.

Data Type	**Typical Size**	**Typical Range / Use**
char	1 byte	Stores a single character; range −128 to 127 (signed) or 0 to 255 (unsigned).
int	2 or 4 bytes	Stores whole numbers; commonly −2,147,483,648 to 2,147,483,647 (4-byte).
float	4 bytes	Stores single-precision decimal numbers, about 6–7 significant digits.
double	8 bytes	Stores double-precision decimal numbers, about 15–16 significant digits.

Poly Notes Hub

Data Type	Typical Size	Typical Range / Use
void	—	Represents 'no value'; used for functions that return nothing or generic pointers.

Note: The exact size of a data type (e.g. int being 2 or 4 bytes) is compiler- and machine-dependent; it can always be verified using the sizeof operator.

Poly Notes Hub

1.3 C Operators, Precedence, Associativity, Type Conversion, Typecasting

An operator is a symbol that instructs the compiler to perform a specific mathematical, relational or logical operation on one or more operands. C provides a rich set of operators, which can be classified as follows.

Arithmetic Operators

Used to perform basic mathematical operations between two operands.

Operator	Meaning	Example (a=10, b=3)
+	Addition	a + b = 13
-	Subtraction	a - b = 7
*	Multiplication	a * b = 30
/	Division (quotient)	a / b = 3
%	Modulus (remainder)	a % b = 1

Relational Operators

Used to compare two operands; the result is either 1 (true) or 0 (false).

- == (equal to), != (not equal to), > (greater than), < (less than), >= (greater than or equal to), <= (less than or equal to).

Logical Operators

Used to combine or invert conditions, mainly with relational expressions; also return 1 (true) or 0 (false).

- && (logical AND) – true only if both conditions are true.
- || (logical OR) – true if at least one condition is true.
- ! (logical NOT) – reverses the logical state of its operand.

Assignment Operators

Used to assign a value to a variable. The simple assignment operator is =. C also provides compound (shorthand) assignment operators that combine an arithmetic/bitwise operation with assignment:

- +=, -=, *=, /=, %= (arithmetic compound assignment, e.g. a += 5 means a = a + 5).
- &=, |=, ^=, <<=, >>= (bitwise compound assignment).

Unary and Binary Operators

- Unary operators act on a single operand, e.g. unary minus (-a), increment (++), decrement (--), logical NOT (!), address-of (&), and sizeof.
- Binary operators act on two operands, e.g. addition (a + b), relational (a > b), and most arithmetic/logical/bitwise operators.

Increment and Decrement Operators

These unary operators increase (++) or decrease (--) the value of a variable by 1. Their position relative to the operand affects when the update occurs:

Poly Notes Hub

- Prefix (++a, --a): the variable is incremented/decremented first, then its updated value is used in the expression.
- Postfix (a++, a--): the current value of the variable is used in the expression first, and the variable is incremented/decremented afterward.

```
int a = 5, b;
b = ++a;   // a becomes 6, then b = 6 (prefix)
b = a++;   // b = 6 first, then a becomes 7 (postfix)
```

Conditional (Ternary) Operator

The only ternary operator in C, written as condition ? expression1 : expression2. If the condition is true, expression1 is evaluated and returned; otherwise expression2 is evaluated and returned.

```
int a = 10, b = 20, max;
max = (a > b) ? a : b;   // max becomes 20
```

Bitwise Operators

These operators work directly on the individual bits of integer-type operands.

Operator	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR (exclusive OR)
~	Bitwise complement (one's complement, unary)
<<	Left shift (shifts bits left, fills with 0)
>>	Right shift (shifts bits right)

Special Operators

- Comma operator (,): allows two or more expressions to be evaluated in sequence within a single statement; the overall value is that of the last (rightmost) expression, e.g. x = (a = 5, b = 10, a + b).
- sizeof operator: a compile-time unary operator that returns the size (in bytes) occupied by a variable or data type, e.g. sizeof(int), sizeof(a).
- Pointer operators: & (address-of) returns the memory address of a variable; * (dereference/indirection) accesses the value stored at the address held by a pointer.
- Member access operators: dot (.) accesses a member of a structure/union variable; arrow (->) accesses a member through a pointer to a structure/union.

Operator Precedence and Associativity

When an expression contains more than one operator, precedence determines which operator is evaluated first, and associativity determines the order of evaluation (left-to-right or right-to-left) when two operators of the same precedence appear together.

Category	Operators	Associativity
Postfix	() [] -> . ++ (postfix) -- (postfix)	Left to right
Unary	++ (prefix) -- (prefix) + - ! ~ (type) * & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right

Poly Notes Hub

Category	Operators	Associativity
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	? :	Right to left
Assignment	= += -= *= /= %= &= ^= |= <<= >>=	Right to left
Comma	,	Left to right

*Note: Precedence decides WHICH operator acts first among different operators (e.g. * is evaluated before + in a + b * c); associativity decides direction of evaluation among operators of EQUAL precedence.*

Type Conversion (Implicit Conversion)

Type conversion occurs when the compiler automatically converts one data type into another, according to a set of rules, so that an operation or assignment can take place without explicit instruction from the programmer. This is also called implicit type conversion or coercion.

- In an expression with mixed data types, C converts 'lower' types to the 'higher' type before evaluation, following the hierarchy: char/short → int → unsigned int → long → unsigned long → float → double → long double.
- Assignment conversion: when a value of one type is assigned to a variable of another type, it is automatically converted to the type of the variable on the left-hand side (e.g. assigning a float value to an int variable truncates the decimal part).

```
int a = 5;
float b = 2.0;
float c = a + b;   // a is implicitly converted to float; c = 7.0
```

Typecasting (Explicit Conversion)

Typecasting is the process by which a programmer explicitly and deliberately converts a value from one data type to another using the cast operator, written as (type)expression. Unlike implicit conversion, this is done intentionally by the programmer to control how a value is treated in an expression.

```
int a = 5, b = 2;
float result;
result = (float) a / b;   // forces floating-point division; result = 2.5
// without the cast, a / b would perform integer division and give 2
```

Typecasting is especially important in division operations to avoid unintended truncation, and when passing arguments to functions or storing large values into smaller data types (where the programmer must be aware of potential data loss).

Poly Notes Hub

1.4 Formatted Input and Formatted Output

C provides standard library functions, declared in the header file stdio.h, for reading data from the standard input device (keyboard) and writing data to the standard output device (screen) in a specified, controlled format. These are known as formatted I/O functions because they use format specifiers to indicate the type and form of the data being read or displayed.

Format Specifiers

A format specifier is a code beginning with % that tells the compiler what type of data is being read or printed. Commonly used format specifiers include:

Specifier	Data Type
%d	int (signed decimal integer)
%u	unsigned int
%f	float
%lf	double
%c	char (single character)
%s	string (character array)
%x / %X	hexadecimal integer
%o	octal integer
%ld	long int

Formatted Output — printf()

The printf() function is used to display formatted output on the screen. Its general syntax is:

```
printf("format string", argument1, argument2, ...);
```

The format string may contain plain text to be displayed as it is, along with format specifiers that are replaced, in order, by the values of the corresponding arguments.

```
#include <stdio.h>
int main() {
    int age = 20;
    float marks = 78.5;
    printf("Age = %d, Marks = %.1f", age, marks);
    return 0;
}
// Output: Age = 20, Marks = 78.5
```

Field width and precision can also be specified, for example %5d prints an integer right-justified within a field of at least 5 characters, and %.2f prints a floating-point number rounded to 2 decimal places.

Formatted Input — scanf()

The scanf() function is used to read formatted input entered by the user from the keyboard. Its general syntax is:

```
scanf("format string", &argument1, &argument2, ...);
```

Poly Notes Hub

Each variable in the argument list must generally be preceded by the address-of operator (&), because scanf() needs the memory address of the variable to store the value that is typed in (the exception is when the argument is already an address, such as an array/string name, which decays to a pointer).

```
#include <stdio.h>
int main() {
    int rollNo;
    float marks;
    printf("Enter roll number and marks: ");
    scanf("%d %f", &rollNo, &marks);
    printf("Roll No: %d, Marks: %.2f", rollNo, marks);
    return 0;
}
```

Note: A very common beginner mistake is forgetting the & before a variable name in scanf() (except for strings/arrays), which leads to incorrect input or a runtime crash.

Together, scanf() and printf() form the basic mechanism for interactive, formatted communication between the user and a C program, and are among the most frequently used functions from the stdio.h library.

Poly Notes Hub

Multiple Choice Questions (MCQs)

Answer keys are provided at the end of this section.

1. Who is regarded as the developer of the C programming language?

(a) Ken Thompson
(b) Dennis Ritchie
(c) Bjarne Stroustrup
(d) James Gosling

2. C was originally developed to rewrite which operating system?

(a) Windows
(b) MS-DOS
(c) UNIX
(d) Linux

3. Which file contains function declarations and macros to be included in a C program?

(a) Source file
(b) Object file
(c) Header file
(d) Executable file

4. Which of the following is NOT a characteristic of C?

(a) Case-sensitive
(b) Middle-level language
(c) Purely object-oriented
(d) Portable

5. Which of the following is a valid identifier in C?

(a) 2value
(b) _total
(c) float
(d) my value

6. How many keywords are defined in standard (ANSI) C?

(a) 16
(b) 32
(c) 48
(d) 64

7. Which data type is used to store a single character in C?

(a) int
(b) float
(c) char
(d) void

8. Which of the following is a floating-point constant?

(a) 25
(b) 'A'
(c) 3.14
(d) "text"

Poly Notes Hub

9. Which operator is used to find the remainder of integer division?

(a) /
(b) %
(c) //
(d) \

10. What will ++a do if a = 5, compared to a++?

(a) Both give the same result always
(b) ++a increments first, a++ increments after use
(c) a++ increments first, ++a increments after use
(d) Neither changes the value

11. Which is the only ternary operator in C?

(a) &&
(b) ?:
(c) ::
(d) ->

12. Which operator is used to determine the size (in bytes) of a data type or variable?

(a) size()
(b) len()
(c) sizeof
(d) memsize

13. Which operator has the highest precedence among the following?

(a) + (addition)
(b) * (multiplication)
(c) () (parentheses/function call)
(d) = (assignment)

14. Which format specifier is used to read/print a floating-point (float) value?

(a) %d
(b) %f
(c) %c
(d) %s

15. In scanf("%d", &num);, the & symbol is used to:

(a) Perform bitwise AND
(b) Pass the address of the variable
(c) Declare the variable
(d) Multiply two variables

Answer Key

1. (b) 2. (c) 3. (c) 4. (c) 5. (b) 6. (b) 7. (c) 8. (c) 9. (b) 10. (b) 11. (b) 12. (c) 13. (c) 14. (b) 15. (b)

Poly Notes Hub

Broad / Descriptive Questions

1. Discuss the history and evolution of the C programming language. Why is C often called the 'mother of modern programming languages'?
2. Explain structured programming and describe its advantages over unstructured programming.
3. Describe, with the help of a diagram or flow, the different types of files (source, header, object, and binary executable) involved in compiling and running a C program.
4. List and explain any six important characteristics of the C programming language.
5. What are tokens in C? Explain the different types of tokens with suitable examples.
6. What is a variable? Explain the rules for naming a variable/identifier in C, giving examples of valid and invalid identifiers.
7. Explain the different categories of data types available in C with their approximate memory size and range.
8. What are operator precedence and associativity? Explain with a suitable example how they affect the evaluation of an expression such as a + b * c - d / e.
9. Differentiate between implicit type conversion and typecasting (explicit conversion) in C, with suitable programming examples.
10. Explain the working of the printf() and scanf() functions in C with the help of a program that inputs a student's roll number and marks and displays them in formatted form.

Poly Notes Hub