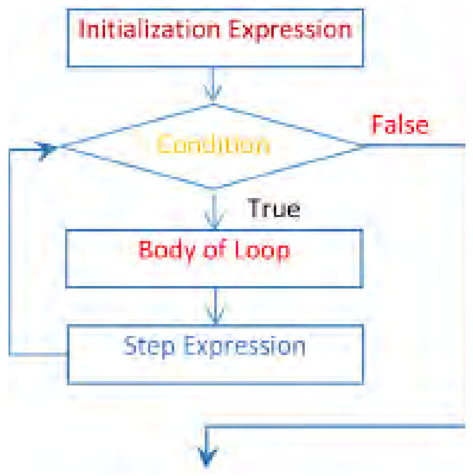
Programming principal and Concept in C Language



Government of Nepal
Ministry of Education, Science and Technology
Curriculum Development Centre
Sanothimi, Bhaktapur

Phone : 5639122/6634373/6635046/6630088
Website- https://www.moecdc.gov.np
Email- info@moecdc.gov.np

Technical and Vocational Stream
Learning Resource Material

Programming principal & concept in C Language

(Grade 9)

Computer Engineering

Government of Nepal
Ministry of Education, Science and Technology

Curriculum Development Centre

Sanothimi, Bhaktapur

Publisher: Government of Nepal
Ministry of Education, Science and Technology
Curriculum Development Centre
Sanothimi, Bhaktapur

Layout by Khados Sunuwar

Preface

The curriculum and curricular materials have been developed and revised on a regular basis with the aim of making education objective-oriented, practical, relevant and job oriented. It is necessary to instill the feelings of nationalism, national integrity and democratic spirit in students and equip them with morality, discipline, self-reliance, creativity and thoughtfulness. It is essential to develop linguistic and mathematical skills, knowledge of science, information and communication technology, environment, health and population and life skills in students. It is also necessary to bring the feeling of preserving and promoting arts and aesthetics, humanistic norms, values and ideals. It has become the need of the present time to make them aware of respect for ethnicity, gender, disabilities, languages, religions, cultures, regional diversity, human rights and social values to make them capable of playing the role of responsible citizens with applied technical and vocational knowledge and skills. This learning resource material for computer engineering has been developed in line with the Secondary Level computer engineering Curriculum with an aim to facilitate the students in their study and learning on the subject by incorporating the recommendations and feedback obtained from various schools, workshops, seminars and interaction programs attended by teachers, students, parents and concerned stakeholders.

In bringing out the learning resource material in this form, the contribution of the Director General of CDC Mr. Yubaraj Paudel and members of the subject committee Dr. Baburam Dawadi, Dr. Sarbim Sayami, Mrs. Bibha Sthapit, Mrs. Trimandir Prajapati is highly acknowledged. This learning resource material is compiled and prepared by Mr. Bimal Thapa, Mr, Rajendra Rokaya, Mr. Suresh Shakya. The subject matter of this material is edited by Mr. Badrinath Timsina and Mr. Khilanath Dhamala. Similarly, the language is edited by Mr. Binod Raj Bhatta. CDC extends sincere thanks to all those who have contributed to developing this material in this form.

This learning resource material contains a wide coverage of subject matters and sample exercises which will help the learners to achieve the competencies and learning outcomes set in the curriculum. Each chapter in the material clearly and concisely deals with the subject matters required for the accomplishment of the learning outcomes. The Curriculum Development Centre always welcomes creative and constructive feedback for the further improvement of the material.

2082 BS

Curriculum Development Centre
Sanothimi, Bhaktapur

Table of Content

Guidelines to Teachers

A. Facilitation Methods

The goal of this course is to combine the theoretical and practical aspects of the contents needed for the subject. The nature of contents included in this course demands the use of practical or learner focused facilitation processes. Therefore, the practical side of the facilitation process has been focused much. The instructor is expected to design and conduct a variety of practical methods, strategies or techniques which encourage students engage in the process of reflection, sharing, collaboration, exploration and innovation new ideas or learning. For this, the following teaching methods, strategies or techniques are suggested to adopt as per the course content nature and context.

Brainstorming

Brainstorming is a technique of teaching which is creative thinking process. In this technique, students freely speak or share their ideas on a given topic. The instructor does not judge students' ideas as being right or wrong, but rather encourages them to think and speak creatively and innovatively. In brainstorming time, the instructor expects students to generate their tentative and rough ideas on a given topic which are not judgmental. It is, therefore, brainstorming is free-wheeling, non-judgmental and unstructured in nature. Students or participants are encouraged to freely express their ideas throughout the brainstorming time. Whiteboard and other visual aids can be used to help organize the ideas as they are developed. Following the brainstorming session, concepts are examined and ranked in order of importance, opening the door for more development and execution. Brainstorming is an effective technique for problem-solving, invention, and decision-making because it taps into the group's combined knowledge and creative ideas.

Demonstration

Demonstration is a practical method of teaching in which the instructor shows or demonstrates the actions, materials, or processes. While demonstrating something the students in the class see, observe, discuss and share ideas on a given topic. Most importantly, abstract and complicated concepts can be presented into visible form through demonstration. Visualization bridges the gap between abstract ideas and concrete manifestations by utilizing the innate human ability to think visually. This enables students to make better decisions, develop their creative potential, and obtain deeper insights across a variety of subject areas.

Peer Discussion

Peer conversation is a cooperative process where students converse with their peers to exchange viewpoints, share ideas, and jointly investigate subjects that are relevant or of mutual interest. Peer discussion is an effective teaching strategy used in the classroom to encourage critical thinking, active learning, and knowledge development. Peer discussions encourage students to express their ideas clearly, listen to opposing points of view, and participate in debate or dialogue, all of which contribute to a deeper comprehension and memory of the course material. Peer discussions also help participants develop critical communication and teamwork skills by teaching them how to effectively articulate their views, persuasively defend their positions, and constructively respond to criticism.

Peer conversation is essential for professional growth and community building outside of the classroom because it allows practitioners to share best practices, work together, and solve problems as a group. In addition to expanding their knowledge horizon and deepening their understanding, peer discussions help students build lasting relationships and a feeling of community within their peer networks.

Group Work

Group work is a technique of teaching where more than two students or participants work together to complete a task, solve a problem or discuss on a given topic collaboratively. Group work is also a cooperative working process where students join and share their perspectives, abilities, and knowledge to take on challenging job or project. Group work in academic contexts promotes active learning, peer teaching, and the development of collaboration and communication skills. Group work helps individuals to do more together than they might individually do or achieve.

Gallery Walk

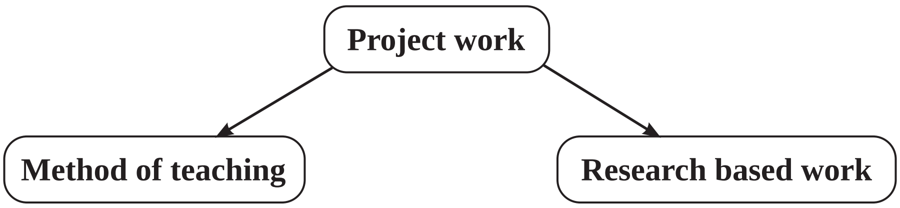
Gallery walk is a critical thinking strategy. It creates interactive learning environment in the classroom. It offers participants or students a structured way to observe exhibition or presentation and also provides opportunity to share ideas. It promotes peer-to-peer or group-to-group engagement by encouraging participants to observe, evaluate and comment on each other’s work or ideas. Students who engage in this process improve their communication and critical thinking abilities in addition to their comprehension of the subject matter, which leads to a deeper and more sophisticated investigation of the subjects at hand.

Interaction

The dynamic sharing of ideas, knowledge, and experiences between people or things is referred to as interaction, and it frequently takes place in social, academic, or professional settings. It includes a broad range of activities such as dialogue, collaboration or team work, negotiation, problem solving, etc. Mutual understanding, knowledge sharing, and interpersonal relationships are all facilitated by effective interaction. Interaction is essential for building relationships, encouraging learning, and stimulating creativity in both in-person and virtual contexts. Students can broaden their viewpoints, hone their abilities, and jointly achieve solutions to difficult problems by actively interacting with others.

Project Work

Project work is a special kind of work that consists of a problematic situation which requires systematic investigation to explore innovative ideas and solutions. Project work can be used in two senses. First, it is a method of teaching in regular class. The next is: it is a research work that requires planned investigation to explore something new. This concept can be presented in the following figure.



Project work entails individuals or teams working together to achieve particular educational objectives. It consists of a number of organized tasks, activities, and deliverables. The end product is important for project work. Generally, project work will be carried out in three stages. They are:

- Planning
- Investigation
- Reporting

B. Instructional Materials

Instructional materials are the tools and resources that teachers use to help students. These resources/materials engage students, strengthen learning, and improve conceptual comprehension while supporting the educational goals of a course or program. Different learning styles and preferences can be accommodated by the variety of instructional resources available. Here are a few examples of typical educational resource types:

- Daily used materials
- Related Pictures
- Reference books
- **Slides and Presentation:** PowerPoint slides, keynote presentations, or other visual aids that help convey information in a visually appealing and organized manner.
- **Audiovisual Materials:** Videos, animations, podcasts, and other multimedia resources that bring concepts to life and cater to auditory and visual learners.
- **Online Resources:** Websites, online articles, e-books, and other web-based materials that can be accessed for further reading and research.

Maps, Charts, and Graphs: Visual representations that help learners understand relationships, patterns, and trends in different subjects.

Real-life Examples and Case Studies: Stories, examples, or case studies that illustrate the practical application of theoretical concepts and principles.

C. Assessment

Formative Test

Classroom discussions: Engage students in discussions to assess their understanding of concepts.
Quizzes and polls: Use short quizzes or polls to check comprehension during or after a lesson.
Homework exercises: Assign tasks that provide ongoing feedback on individual progress.
Peer review: Have students review and provide feedback on each other's work.

Summative Test

Exams: Conduct comprehensive exams at the end of a unit or semester.
Final projects: Assign projects that demonstrate overall understanding of the subject.

Peer Assessment

Group projects: Evaluate individual contributions within a group project.
Peer feedback forms: Provide structured forms for students to assess their peers.
Classroom presentations: Have students assess each other's presentations.

Objective Test

Multiple-choice tests: Use multiple-choice questions to assess knowledge.
True/False questions: Assess factual understanding with true/false questions.
Matching exercises: Evaluate associations between concepts or terms.

Portfolio Assessment

Compilation of work: Collect and assess a variety of student work samples.
Reflection statements: Ask students to write reflective statements about their work.
Showcase events: Organize events where students present their portfolios to peers or instructors.

Observational Assessment

Classroom observations: Observe students' behavior and engagement during class.
Performance observations: Assess practical skills through direct observation.
Field trips: Evaluate students' ability to apply knowledge in real-world settings.

Abbreviation

CRM: Customer Relationship Planning

ERP: Enterprise Resource Planning

GCC: GNU Compiler Collection

GUI: Graphical User Interface

HLL: High-Level Language

HPC: High-Performance Computer

HRM: Human Resource Management

I/O: Input Output

IDE: Integrated Development Environment

LLL: Low-Level Language

MATLAB: Matrix Laboratory

MLL: Machine Level Language

PoS: Point of Sale

QBASIC: Quick Beginners All-purpose Symbolic Code

Principles of Programming

Unit 1

1.1. Introduction to Programming

Programming is the first step into the dynamic world of computer science and problem-solving using code. Programming is fundamentally about creating specific sets of instructions, or algorithms, that a machine can perform. These instructions are expressed in a specific programming language, each with its syntax and constraints. The essential ideas include comprehending variables and data types for storing and manipulating information, control structures such as if statements and loops to guide program flow, and the usage of functions for code modularity.

Programming is more than just writing code; it's a methodical approach to dissecting issues, finding solutions, and implementing them in a way that a computer can comprehend and execute. Aspiring programmers frequently start by creating a development environment, writing simple programs, and then progressing to more sophisticated tasks. The journey entails continual learning, troubleshooting, and skill refinement, which opens the door to a world of possibilities in software development and technical innovation.

Program

A program is a set of instructions in a programming language, that guides a computer through a sequence of actions to produce a desired output, ranging from simple scripts to complex software applications.

Programmer

A programmer is a person who has skills in problem-solving, logic, and algorithmic thinking and creates computer programs using programming languages across various industries like software development, web development, data science, and artificial intelligence.

Programming Language

A programming language (computer language) is an artificial language developed for writing computer programs. A variety of programming languages have their own defined instructions and syntax. Python, Java, C, C++, etc. are programming languages.

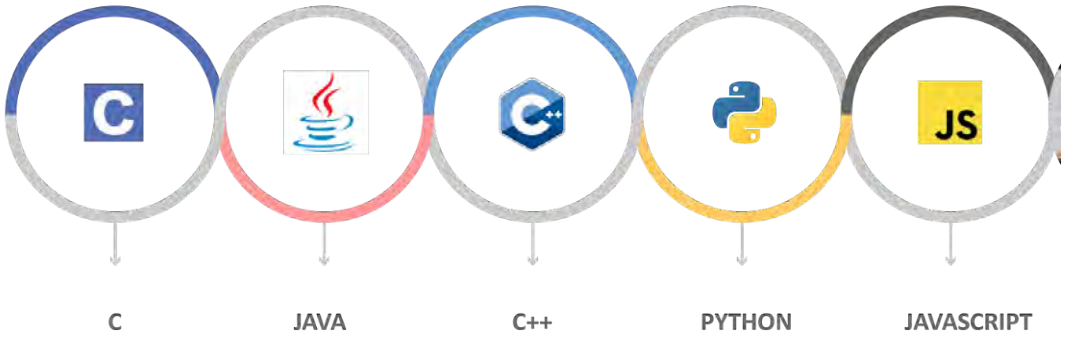


Figure 1Different programming languages

Software

Software is a set of programs, data, and instructions that enable computers to perform specific tasks. Windows 10, Android operating software, Microsoft Office, antiviruses, etc. are software. Software are developed by programmers.

1.2. Categories of programming of Lamguage

Programming languages can be categorized into several types based on their features, application domains, and programming paradigms.

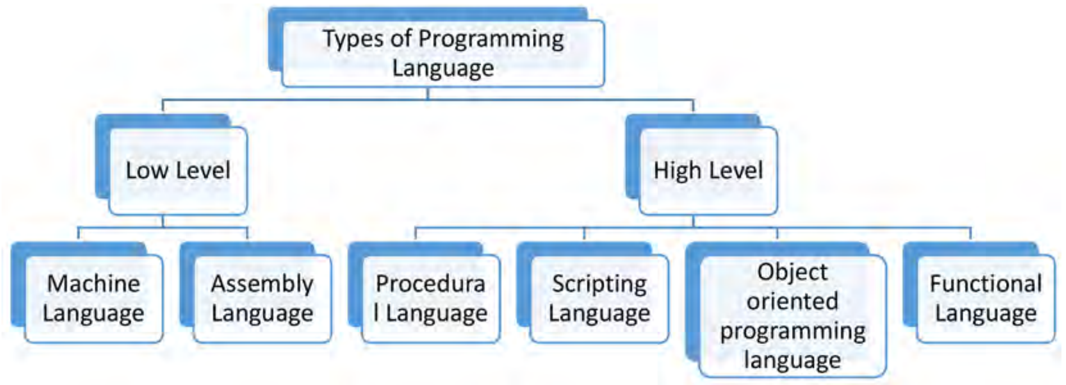


Figure 2 Different types of programming languages

Here are some common categories of programming languages:

- **High-Level Languages:** These languages are designed to be easily understood by humans and are closer to natural language. Examples include Python, Java, and C#.
- **Low-Level Languages:** These languages are closer to machine code and are more challenging for humans to read and write. Assembly language is an example, and machine code is the lowest level.

- **Imperative Languages:** These languages focus on describing how a program operates. C, C++, and Java are examples.
- **Declarative Languages:** These languages emphasize describing what a program should accomplish without specifying how. SQL and HTML are examples.
- **Procedural Languages:** Programs are structured around procedures or routines. C and Pascal are procedural languages.
- **Object-Oriented Languages:** Programs are structured around objects that encapsulate data and behavior. Examples include Java, Python, and C++.
- **Functional Programming Languages:** These languages treat computation as the evaluation of mathematical functions. Haskell and Lisp are examples.
- **Scripting Languages:** These languages are often interpreted and used for automating tasks. Python, JavaScript, and Ruby are popular scripting languages.

1.3. Application of Programming

Programming is a flexible tool used in many fields, such as data analysis, artificial intelligence, machine learning, and software development, is programming. It promotes innovation in IoT devices, cybersecurity, and game development. Scientific research is aided by programming, which also improves efficiency across a range of industries.

1.3.1 Scientific Application

Programming is a versatile technique that may be applied in a variety of domains, including software development, data analysis, artificial intelligence, and machine learning. It promotes innovation in game creation, cybersecurity, and IoT devices. Programming advances scientific research, streamlines operations, and improves efficiency in a variety of industries.

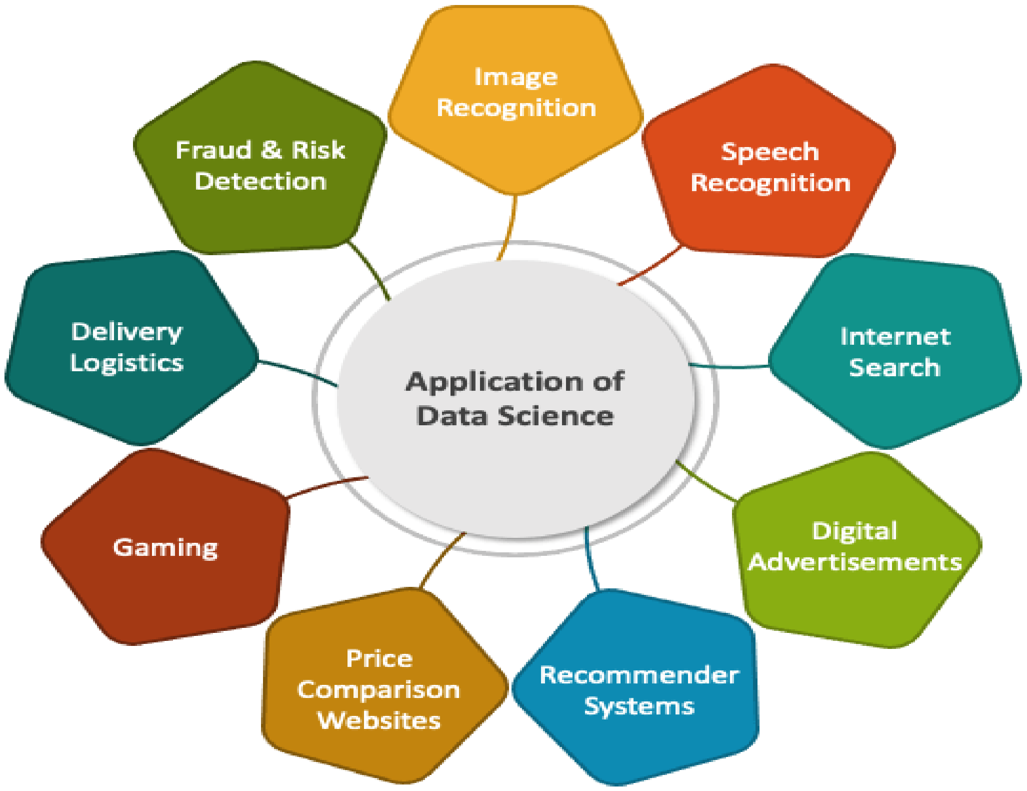


Figure 3 Application of programming languages

Scientific applications involve data analysis, simulation and modeling, numerical computing, machine learning and AI, visualization, high-performance computing (HPC), bioinformatics, chemical modeling, environmental modeling, and data integration and collaboration. These applications use programming languages like Python, R, NumPy, Pandas, MATLAB, Julia, and SciPy for data manipulation, statistical analysis, and visualization. Numerical methods like MATLAB, Julia, and Python are used for solving mathematical problems in scientific research. Python, along with libraries like TensorFlow and scikit-learn, is widely used for implementing machine learning models. High-Performance Computing (HPC) is used for parallel processing. Bioinformatics uses Python and specialized tools for analyzing biological data. Chemical modeling uses Python and specialized tools for predicting chemical properties. Environmental modeling simulates environmental processes and ecosystems.

1.3.2 Business Application

Business applications are software solutions designed to improve efficiency, streamline processes, and support decision-making within a business or organizational context. Key aspects of business applications include Enterprise Resource Planning (ERP) systems, Customer Relationship Management (CRM) software, accounting software, Human Resource Management (HRM) systems, project management software, Business Intelligence (BI) and Analytics, Supply Chain Management (SCM) systems, e-commerce platforms, communication and collaboration tools, document management systems, customer support and help desk software, Point of Sale (POS) systems, security and compliance software, and Business Process Automation (BPA) tools. These applications help businesses manage various functions, streamline processes, and support decision-making. Examples of popular business applications include SAP, Oracle ERP, Salesforce, HubSpot, Zoho CRM, QuickBooks, Xero, FreshBooks, Workday, BambooHR, Workday, BambooHR, Asana, Jira, Tableau, Power BI, SAP SCM, Shopify, Magento, WooCommerce, Microsoft Teams, Slack, Zoom, SharePoint, M-Files, Zendesk, Freshdesk, Salesforce Service Cloud, Square, Lightspeed, Shopify POS, Symantec, McAfee, and Varonis.

1.4. Program Design Tools (Algorithm and Flowchart)

Program design tools, such as algorithms and flowcharts, are essential to the software development process because they assist programmers in planning and visualizing the logic of their systems. They offer an easy approach to creating a computer program in any programming language. Here's an overview of algorithm and flowchart design:

Algorithm

An algorithm is a step-by-step technique or collection of rules used to solve a specific problem or complete a specific activity. It gives a clear description of the answer to a problem.

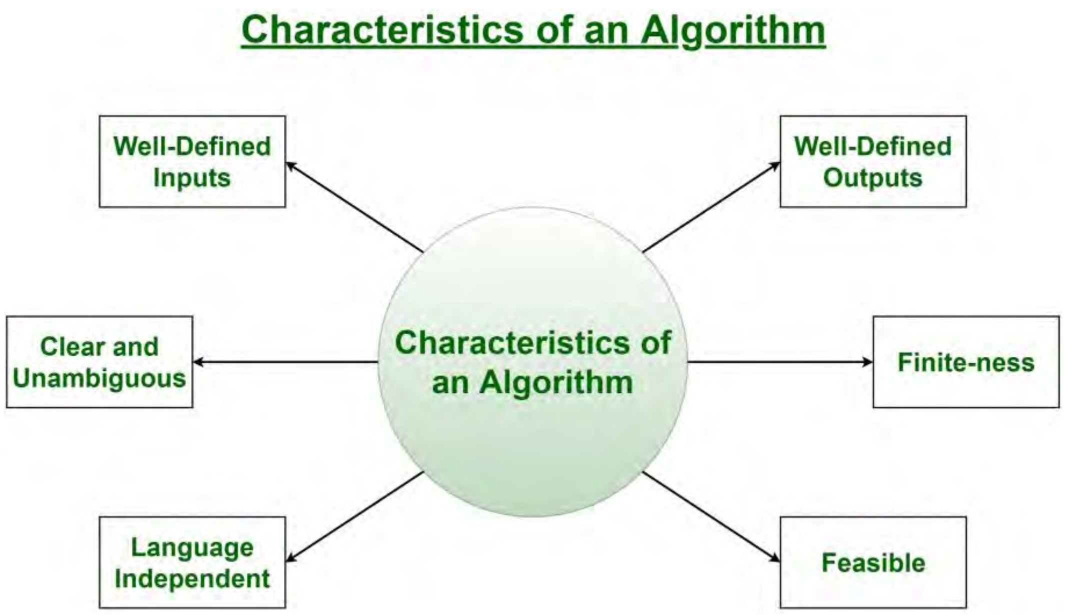


Figure 4 Characteristics of programming language

Characteristics of Good Algorithms

The algorithm must be precise, finite, have defined inputs and outputs, and be effective, ensuring each step is feasible and well-defined.

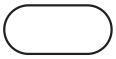
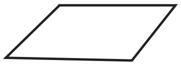
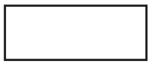
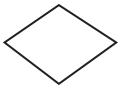
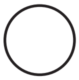
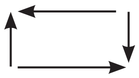
Flowchart

A flowchart is a graphical description of a process or algorithm that uses multiple shapes to represent distinct steps and connectors to depict control flow. It visualizes a program's logical flow.

Common Flowchart Symbols

- **Terminator:** Indicates the start or end of a process.
- **Process:** Represents a specific operation or action.
- **Decision:** Represents a decision point with branching paths.
- **Input/Output:** Signifies input or output operations.
- **Connector:** Connects different parts of the flowchart.
- **Flow Arrows:** Indicate the direction of the flow.

A diagram of the sequence of operations in a computer program is called a flow chart. It uses a few standard symbols. They are:

S.N.	Symbol	Name	Description
1.		Start/End Symbol	Represents START and END of the program.
2.		Input/output Symbol	Represents input and output of data.
3.		Process	Process block. Represents process, formula, or function.
4.		Decision	Decision block. Represents any decision in the program.
5.		Connector	Connector. They are used to link to segments of the flowchart.
6.		Flow Lines	Flow lines. Used to show the direction of flow of the program.

Steps for Algorithm and Flowchart Design

- **Understand the Problem:** Clearly define the problem and understand the requirements.
- **Identify Inputs and Outputs:** Determine what data the program will take as input and what results it should produce as output.
- **Develop the Algorithm:** Write a step-by-step algorithm in pseudocode, focusing on the logical flow and decision points.
- **Draw the Flowchart:** Create a flowchart that visually represents the algorithm, using appropriate symbols and connectors.
- **Review and Refine:** Evaluate the algorithm and flowchart for clarity, correctness, and efficiency. Make adjustments as needed.

Advantages of Algorithm and Flowchart Design

- **Clarity and Understanding:** Provides a clear and structured representation of the solution.
- **Communication:** Facilitates communication between team members and stakeholders.
- **Planning:** Aids in planning and organizing the logic before coding.
- **Debugging:** Helps identify potential issues and logical errors early in the development

process.

- **Documentation:** Serves as documentation for future reference and maintenance.

Practical works

An algorithm is a precise rule (or set of rules) specifying how to solve a specific problem in a finite number of steps. A good algorithm should:

- Be language independent.
- Be simple, complete, unambiguous, and step-wise.
- Have no standard format or syntax.
- Help understand problems.

Example 1

Algorithm for taking online class through Zoom:

```
Step 1 : Start
Step 2 : Click on Zoom program.
Step 3 : Fill up meeting ID and type your name.
Step 4 : Enter your meeting pass code.
Step 5 : Say Hello to everyone
Step 6 : Take your class.
Step 7 : Say Bye Bye, after completing online class.
Step 8 : Close the Zoom program.
Step 9 : Stop
```

Example 2

Algorithm to calculate the sum of any two numbers

```
Step 1: Start
Step 2: Ask any two numbers and stores into a and b
Step 3: c = a + b
Step 4: Display c
Step 4: Stop
```

Example 3

Algorithm to find the greater number among any two supplied numbers

```
Step 1: Start
Step 2: Ask any two numbers and stores into a and b
Step 3: Is a>b?
                If yes, display a
                If no, display b
Step 4: Check more?
                If yes, go to step 2
                If no, go to step 4
Step 5: Stop
```

Example 1

The following flow chart of *Taking online class through Zoom*

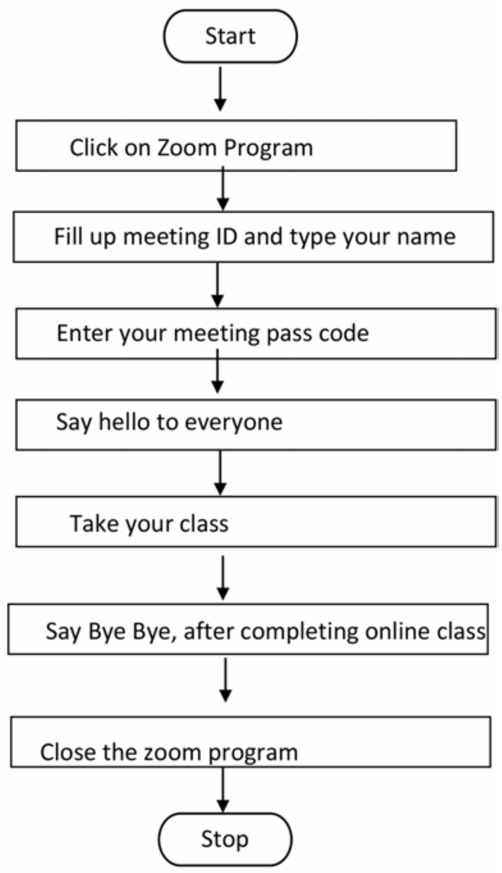

Start
Click on Zoom Program
Fill up meeting ID and type your name
Enter your meeting pass code
Say hello to everyone
Take your class
Say Bye Bye, after completing online class
Close the zoom program
Stop

Figure 5 Flowchart to take online class using zoom

Example 2

Flowchart to find the sum of any two supplied numbers

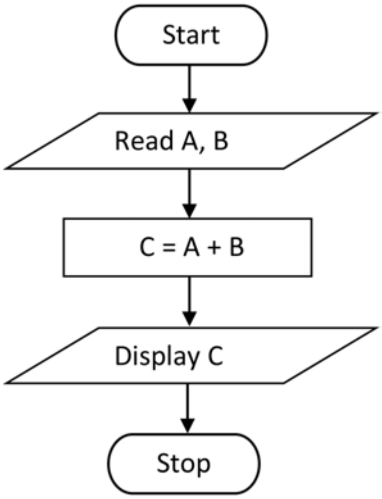


Figure 6 Find sum of two entered numbers

Example 3

Flowchart to find the greater number among any two supplied numbers

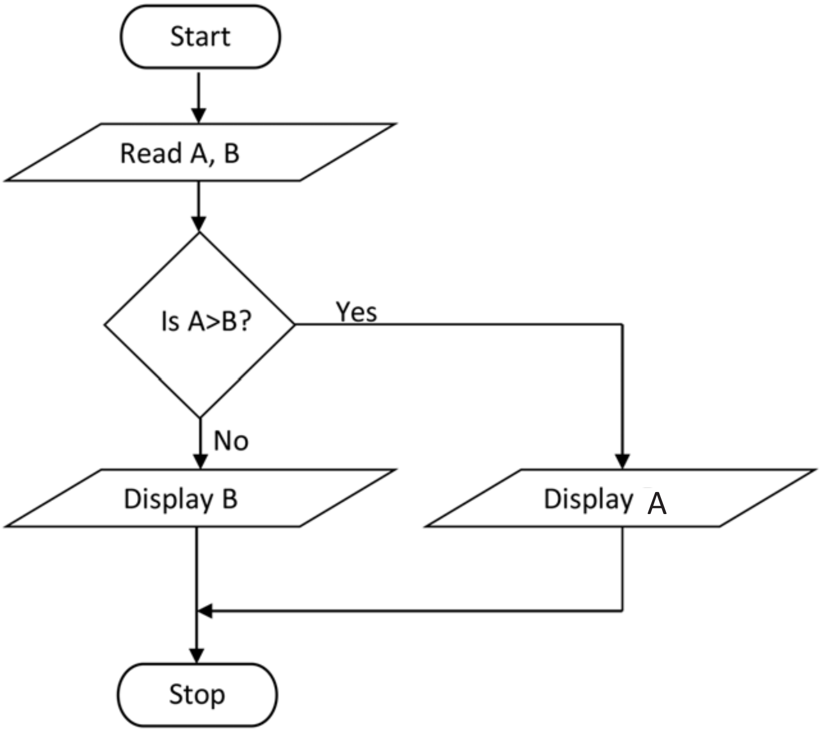


Figure 7 FLowchart to display greatest number among two entered numbers

Exercise

Choose the correct answer from the given alternatives.

1. What is the primary role of a programmer?

   a. Designing computer hardware　　b. Writing code and creating software
   c. Managing computer networks　　d. Testing software applications

2. What is the purpose of a programming language?

   a. To communicate between users and computers
   b. To facilitate communication between programmers
   c. To design computer hardware
   d. To perform mathematical calculations

3. Which programming language is commonly used in scientific computing and data analysis due to its versatility and extensive library support?

   a. Java　　b. Python　　c. C#　　d. Ruby

4. Which of the following shapes in a flowchart represents a process or operation?

   a. Oval　　b. Rectangle　　c. Diamond　　d. Parallelogram

5. What does the arrow in a flowchart represent?

   a. Decision　　b. Data input　　c. Data output　　d. Flow direction

Write short answers to the following questions.

1. What is the primary purpose of writing a program in a programming language?
2. Name a popular programming language for scientific computing and explain its advantages.
3. What is the role of simulations in scientific applications, and how are they created using programming?
4. Define an algorithm. How does it differ from a program?
5. What are the key characteristics of a good algorithm?
6. Define a flowchart. How does it aid in the development of algorithms?
7. Explain the purpose of using different shapes and symbols in a flowchart.

8. Discuss the advantages of using flowcharts in the design and documentation of algorithms.
9. Write an algorithm and draw a flowchart:
    a. To make a cup of coffee.
    b. To display the average of any three numbers.
    c. To calculate the area of a room.
    d. To display the greatest number among any two numbers.
    e. To display whether the given number is even or odd.
    f. To display whether the number is positive, negative or neutral.
    g. To display the sum of all the numbers from 1 to 10.
    h. To display the product of all the numbers from 1 to 20.
    i. To display the factorial of the given number.
    j. To display the factors of the given number.

Write long answers to the following questions.

1. Describe about Business Application and scientific application of C programming.
2. Explain the different types of program design tools with their significance.
3. What is a flowchart? Draw the different symbols that are used in the flowchart along with their meanings.

Fundamentals of C

Unit 2

2.1 Introduction to C Programming

A general-purpose programming language, C is invented by Dennis Ritchie at Bell Labs in the early 1970s. It is a procedural programming language, which means it operates in a linear flow of control, and it is well-known for its efficiency, flexibility, and low-level programming skills.

Features of C

a. **Procedural Language:** C is a programming language that uses a procedural paradigm, organizing code into functions or procedures that perform operations on data.

b. **Low-Level Programming:** C is a programming language that allows for direct access to memory addresses, facilitating efficient manipulation of hardware resources and is commonly utilized in system-level programming and operating system development.

c. **Efficiency:** C is renowned for its efficiency in execution speed and memory usage, enabling precise control over system resources.

d. **Portability:** C programs can be easily ported across different platforms with minimal modifications due to their standardized syntax.

e. **Structured Programming:** C is a programming language that adheres to structured programming principles, enabling developers to create clear and organized code using functions and blocks.

Limitation of C Language

C has several restrictions, yet it is a powerful programming language. Some limitations of C include no run-time type checking, lack of support for object-oriented programming, limited source code reusability, only 32 keywords, no data protection, and inability to handle exceptions (run-time errors).

Application of C Language

C was initially used for system development, specifically the programs that comprise the operating system. Specifically, it generates [optional] code that executes nearly as quickly as code written in assembly language. However, the C programming language is useful for more than only system software development.

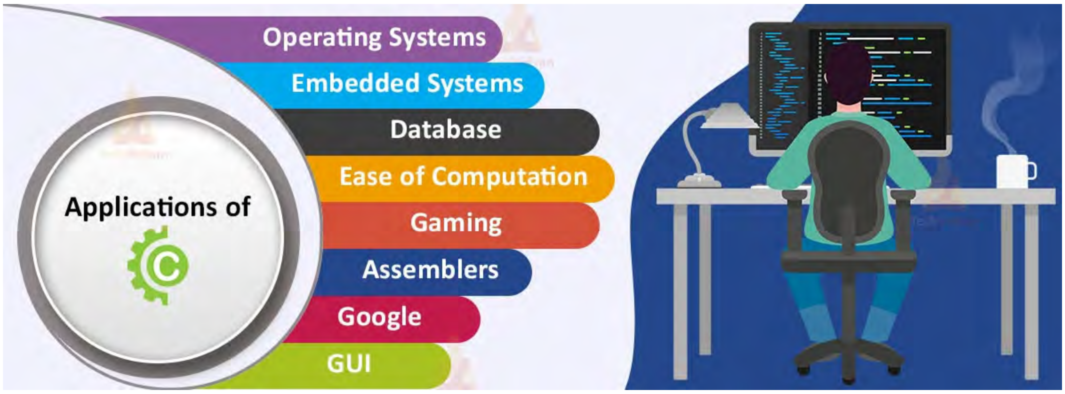


Figure 9 Different applications of c

C language is widely used in a variety of applications, including operating systems, language compilers/interfaces, assemblers, text editors, print spoolers, network devices, modern programs, database management systems, and utilities.

Basic Requirement for Writing Code in CLlanguage

You can write code in C language simply using a text editor like Notepad, Notepad++ or sublime text. Once you finished writing the code in C language, save the program with .c extension and compile the program using any C compiler like GCC.

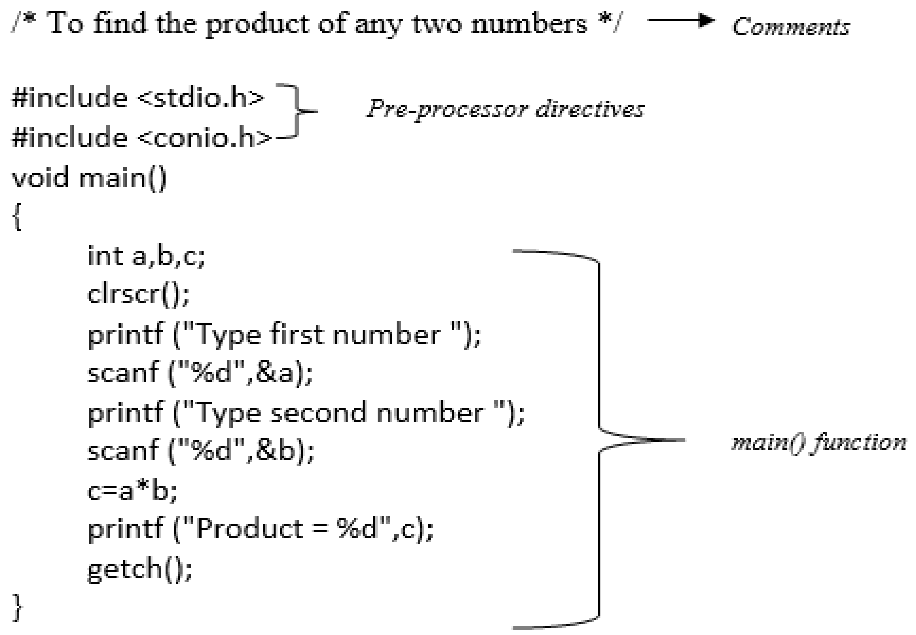
You can use any integrated development environment (IDE) like Code::Blocks, Eclipse, Visual studio, Dev C++ or Turbo C++ to write and compile the code in C language.

2.2 Basic Program Structure (Preprocessor Directive, Header Files, Tokens, Semicolons, Comments, Identifiers, Whitespace, Escape Sequence)

```
Pre-processor directives
    Global Declarations
    main ()
    {
      Local declarations
```

```
        Program Statements
        Calling user defined for (optional)
      }
      user defined functions
      function 1
      function 2
      function 3
```

Example 1



```
/* To find the product of any two numbers */

#include <stdio.h>
#include <conio.h>
void main()
{
        int a,b,c;
        clrscr();
        printf ("Type first number ");
        scanf ("%d",&a);
        printf ("Type second number ");
        scanf ("%d",&b);
        c=a*b;
        printf ("Product = %d",c);
        getch();
}
```

Parts of a C Program

Preprocessor Directive

The C compiler uses the C preprocessor for compilation, which performs code addition, removal, tokenizing, and linking the resulting program with the required programs and libraries.

While writing a program in C, we need to include different header files in the beginning. In the above program, printf() and scanf() functions are used for output and input operation. These functions are defined in the header file <stdio.h>. So, this header file is included at the beginning of the program which contains the code of printf() and scanf() functions. All the code of header files will be added to the program during compilation.

Header Files

In C language, library functions, like scanf(), are defined in header files with.h extension. #include directive is used to include header files, otherwise, compiler fails. Here is the list of some commonly used Header file and their purposes:

Header Files	Purpose	Functions Declared
stdio.h	Used for standard input and output (I/O) operations.	printf(), scanf(), getchar(), putchar(), gets(), puts(), getc(), putc(), fopen, fclose(), feof()
conio.h	Contains declaration for console I/O functions.	clrscr(), exit()
ctype.h	Used for character-handling or testing characters.	isupper(), is lower, isalpha()
math.h	Declares mathematical functions and macros.	pow(), squr(), cos(), tan(), sin(), log()
stdlib.h	Used for number conversions, storage allocations.	rand(), srand()
string.h	Used for manipulating strings.	strlen(), strcpy(), strcmp(), strcat(), strlwr(), strupr(), strrev()

Global Directives: In this section of C program, Global variables and User-defined function are declared.

main () function: C program must start with main() function. This is the entry point of the program.

{ } Parenthesis: In C language, each function is defined inside parenthesis ({ }).

User-defined function: As in QBASIC, we can create different user-defined function as per our requirements.

Output Function in C

The output function is used to show the calculated result or output on the screen. In C language, printf() is one of the output functions defined in <stdio.h> header file.

printf() function

In C Language, printf() function is used to print the valued on the screen. It is defined in <stdio.h> header file. So, the header file <stdio.h> must be added to use this function.

Syntax

```
printf("format string",argument list);
```

format string is the combination of format identifier, escape sequence or string constant.

Escape Sequence

An escape Sequence is a pair of character that is used with printf() function to display non-printing character or an special character on the screen.

Some Examples of Escape Sequence:

\n	-	new line
\t	-	tab
\b	-	backspace
\o	-	null character
\?	-	question mark
\\	-	slash
\'	-	single quote
\"	-	double quote

Tokens

Tokens in C are the basic components for creating a program, including identifiers, keywords, constants, strings, special symbols, and operators, making sentences and programs interconnected.

Semicolons

C programming uses semicolons (;) as statement terminators, marking the end of a statement and separating multiple statements within a program, including variable declarations, assignments, function calls, and executable statements.

Comments

C programming language supports single-line and multi-line comments, providing explanatory statements for programmers to understand flow. Comments are not executed by the compiler and all enclosed characters are ignored. Single-line Comments uses a double slash // and used to comment single line. Multi-line comment starts with a slash asterisk /* and ends with an asterisk slash */ and programmer can place it anywhere in the code, on the same line or several lines.

Example 2

```
// Single Line comment
#include <stdio.h>
void main( )
{
   // This is a single line comment
   printf("This is my first program in C"); //Display output on the screen
   getch( );
}
```

Example 3

```
/* Multiline Comment
    Demo C program
    Use of  printf function
*/
#include <stdio.h>
void main ( )
{
printf("Hello world");
getch( );
}
```

Whitespace

Whitespace in C programming, including spaces, tabs, and newline characters, is used to separate and format code, enhancing readability, while spaces and tabs indent code for visual organization.

```
int main() {
    int x = 5;  // Proper indentation
    if (x > 0) {
        printf("Positive\n");
    }
    return 0;
}
```

Identifier

Identifiers are user-defined names for program elements like variables, types, templates, classes, functions, unions, structures, arrays, or namespaces.

For example

```
float gpa;
double salary;
```

Here gpa and *salary are the name given to the variables. So gpa and salary* are identifiers.

- Identifier should end with either letter or digit.
- Identifiers cannot contain any special characters such as semicolon, period, whitespaces, slash or comma.
- Identifiers are also case sensitive in C. For example empid and EmpID are two different identifiers in C.

For example

Identifiers	Remarks
college_1	Valid
11name	Invalid
price$	Invalid
Hardware_price	Valid
Empid_	Invalid

Getch() Function

getch() function is another input function of C language. This function is defined in the header file <conio.h>. This function is used to ask any one character from keyboard.

Example 4

```
#include <conio.h>
#include <stdio.h>
void main()
{
    char ch;
    clrscr();
    ch=getch();
    printf("The typed character is %c.",ch);
```

```
      getch();
    }
```

Note:

You can see the use of getch() function in every example of C program in this book. The purpose of using this function in the sample program is to let the user to read output on the screen. If such type of function is not used, the output screen will be closed immediately after showing the output and returns to the coding window. In the above program, after showing the output by printf() function, getch() asks a character and get chance to see the output until the character is not typed.

2.3 Variable and Keywords

Variables

While writing a program, programmers must store a variety of values. Values can be stored in memory so that they are easier to access when needed. The term "variable" refers to the memory location where values are stored. The variable's value is not fixed and can change during program execution. Variable names are simply the symbolic representation of a memory location. Before using variables in C, they must be specified using data types.

Types of Variables

There are two types of variables used in C language. They are:

1. Numeric Variables
2. String Variables

Numeric Variables

Numeric variables are those that store numerical values or numbers. Numbers are typically stored as integers, floating-point values, or complex numbers, depending on the needs of the program. Numeric value or data refers to numbers capable of performing mathematical calculations. Numeric variables are declared with the int, float, long int, and double data types. In C, numeric variables are classified into two categories.

Integer Variables

Integer variables are used to hold whole numbers without fractional parts and can be signed or unsigned, with signed variables holding positive or negative values.

Floating-point Variables

The variables which are used to hold decimal values or fractional parts are called floating point variables.

String Variable

String variables hold characters in a linear sequence, cannot perform mathematical calculations, and are declared using char data type, with single and double quote marks for character representation.

Keywords

Keywords, or reserved words, have specific functions in a program and are not used as identifiers for variables or other functions by compilers. C language supports 32 keywords which are given below.

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

2.4 Character Sets, Constants

Character Sets

C language supports 256 characters, including letters, digits, special symbols, and white spaces, as a building block for basic program elements, forming the foundation of every language. The different character set used in C are

Letters	Uppercase A-Z Lowercase a-z
Digits	All digits from 0-9
Special Characters	Symbols: , . : ; ? ' " ! | \ / ~ _$ % # & ^ * - + < > ( ) { } [ ]
White Spaces	Blank space, Horizontal tab, Carriage return, New line, Form feed

Constants

Constants, also known as literals, are fixed values used in a program that remains constant throughout its execution. The different types of constants in C are

Constant	Example
Integer Constant	100, 200, 45 etc.
Real or Floating-point Constant	10.35, 2.7, 45.63 etc.
Character Constant	'A', 'b', 'n' etc.
String Constant	"c program", "Programming" etc.

Symbolic Constant

Symbolic constant is a name for a fixed value in a program, defining a variable that cannot be changed, and can be defined using the const keyword and #define directive.

For example, const float pi=3.14

#define pi 3.14

In the above example, pi has been defined as the constant. So modification or change in the value of pi is not allowed in the entire program.

2.5 Data Types and Formate Specifiers

The data type describes the type of data that the variable can store, such as integer, floating-point, character, and so on. All variables must be declared using data types before they can be used in the application. In C, data types are classified as primary, derived, enumeration, and void.

Primary Data Type

Primary data types are sometimes known as primitive, fundamental, or inbuilt data types. These are the data types that are already predefined in computer languages. The C programming language supports the basic data types char, int, float, and double, as well as the modifiers signed, unsigned, short, and long. varied data formats have varied ranges for storing numbers. These ranges may differ from compiler to compiler. The following is a list of ranges, as well as the memory requirements and format specifiers for the 32-bit compiler.

Data Type	Memory (bytes)	Range	Format Specifier
short int	2	-32,768 to 32,767	%hd

unsigned short int	2	0 to 65,535	%hu
unsigned int	4	0 to 4,294,967,295	%u
int	4	-2,147,483,648 to 2,147,483,647	%d
long int	8	-2,147,483,648 to 2,147,483,647	%ld
unsigned long int	8	0 to 4,294,967,295	%lu
long long int	8	-(2^63) to (2^63)-1	%lld
unsigned long long int	8	0 to 18,446,744,073,709,551,615	%llu
signed char	1	-128 to 127	%c
unsigned char	1	0 to 255	%c
float	4		%f
double	8		%lf
long double	16		%Lf

Derived Data Types

A derived data type is formed by using one or more basic data types in combination. Array, structure, union, and pointer are called derived data types.

Enumeration Data Types

Enumerated data type (enumeration) is a user-defined data type that can be used to assign some limited values. These values are defined by the programmer at the time of declaring the enumerated type. enum keyword is used to declare enumerated types of data.

Void Data Type

Void means empty or no value. The data type void indicates that no value is available. Void is also used to indicate when a function does not return a value or no argument.

Type of Specifier

The input and output data are formatted by using a specific pattern. These patterns are generated by using specific tokens in C programs. The tokens which are used to format data in a program are called specifiers. Some specifier starts with the % operator followed by a special character for identifying the type of data. Some specifiers are special characters that are written after backslash (\ ). The different types of specifiers are

Format Specifier

Format specifiers are used for input and output purposes. Format specifier allows the compiler to recognize what type of data is in a variable while taking input using the scanf() function and printing using the printf() function. Format specifier start with a percentage % operator and followed by a special character for identifying the type of data. The different types of format specifiers are

Format Specifier	Description
%d	Integer Format Specifier
%f	Float Format Specifier
%c	Character Format Specifier
%s	String Format Specifier
%u	Unsigned Integer Format Specifier
%ld	Long Int Format Specifier
%lu	unsigned long int
%lld	long long int
%llu	unsigned long long int
%lf	Double

Escape sequence is used to format the output. Escape sequence begins with a backslash '\' followed by a character or characters. Some of the common escape sequence are

Escape Sequence	Meaning
\a	Alarm or Beep
\b	Backspace
\f	Form Feed
\n	New Line
\r	Carriage Return
\t	Tab (Horizontal)
\v	Vertical Tab
\\	Backslash

\'	Single Quote
\"	Double Quote
\?	Question Mark
\nnn	octal number
\xhh	hexadecimal number
\0	Null

2.6 Input/Output/Statements

Every program should prompt the user for input and return meaningful results. C also has many input/output functions that are responsible for accepting user input through input devices and presenting output via output devices. C provides two types of input/output functions: formatted I/O and unformatted I/O.

Formatted I/O

There are two types of formatted I/O functions in C.

a. printf ( )

It is the most commonly used formatted I/O function in C. It is used to print integer, float, character, string, octal and hexadecimal values.

Syntax:

printf("format specifier",list of variables)

where

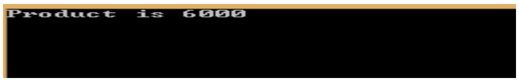
- format specifier can be %d (for integer variable), %f(for floating point variable), %c(for character variable), %s(for string variable).

Example 5

// C program to display the product of three numbers.

```
#include <stdio.h>
# include <conio.h>
  int main( )
  {
    int a,, b, c, p;
    a = 10;
    b= 20;
    c=30;
    p = a*b*c;
printf("Product is %d",p);
getch ( );
return 0;
}
```

Output



```
Product is 6000
```

b. scanf ( )

It is used to ask the input from the user and accepts data through the keyboard. The data entered by the user is stored in the variable.

Syntax

scanf("format specifier", list of variables)

where

- format specifier can be %d (for integer variable), %f(for floating point variable), %c(for character variable), %s(for string variable)

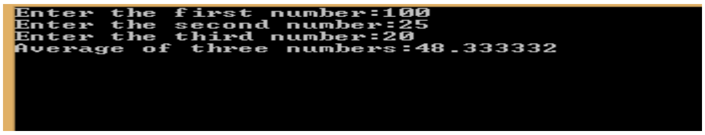
Example 6

// C program to calculate the average of any three numbers given by the user.

```
#include <stdio.h>
# include <conio.h>
void main ( )
{
    float x, y,z, avg;
    printf("Enter the first number:");
    scanf("%f", &x);
    printf("Enter the second number:");
    scanf("%f", &y);
    printf("Enter the third number:");
```

```
        scanf("%f", &z);
        avg=(x+y+z)/3;
        printf("Average of three numbers:%f", avg);
    getch (  );
    }
```

Output



```
Enter the first number:100
Enter the second number:25
Enter the third number:20
Average of three numbers:48.333332
```

Unformatted I/O

This type of I/O function does not require any format specifier There are three types of unformatted I/O functions.

1. Character I/O
2. String I/O
3. File I/O

1. Character I/O

a. getchar( )

This function is used to read one character at a time until and unless the user presses the "Enter Key". It can only accept and store CHAR type input from the user.

Syntax

```
Variable_Name = getchar( )
```

Example 7

```
#include<stdio.h>
#include<conio.h>
void main( )
{
    char x;
    x=getchar( )
    getch ();
}
```

b. putchar( )

This function use to display one character in the screen at a time.

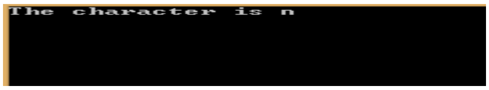
Syntax:

```
putchar( variable name )
```

Example 8

```
#include<stdio.h>
#include<conio.h>
void main( )
{
    char x;
    x= 'n';
    printf("The character is ");
    putchar(x);
  getch ();
}
```

Output



```
The character is n
```

getch( ), getche() and putch()

The getch function read any alphanumeric character from the standard input device. The character entered is not displayed by the getch( ) function until the enter is pressed. The getche function gets a character from the console and echoes to the screen. The putch function print any alphanumeric character given by the user on the screen.

Syntax of getch:

```
    Variable_Name = getch( )
```

Syntax of getche:

```
    Variable_Name = getche( )
```

Syntax of putch:

```
    putch(variable name )
```

Example 9

```
#include<stdio.h>
#include<conio.h>
int main( )
{
    char x,y;
      printf("Enter the first character ");
    x=getch();
      printf("\nEnter the second  character ");
    y=getche();
      printf("\nThe first character is ");
    putch(x);
      printf("\nThe second character is ");
    putch(y);
  getch();
return 0;
}
```

2. String I/O

a. gets()

This function is used to accept string from the user.

Syntax

```
gets(string variable)
```

Example 10

```
#include<stdio.h>
#include<conio.h>
int main( )
{
    char name[100];
      printf("Enter the name: ");
    gets(name);
      printf(" My name is %s",name);
    getch ();
  return 0;
}
```

b. puts( )

This function is used to print the string on the screen.

Syntax

```
puts(string variable)
```

Example 11

```
#include<stdio.h>
#include<conio.h>
int main( )
{
    char name[100];
      printf("Enter the name: ");
    gets(name);
      printf(" My name is");
      puts(name);
    getch ();
  return 0;
}
```

3. File I/O

A file is the collection of related information. A file is created to store data and information for permanent period. Some of the file I/O function are

File I/O function	Description
fopen( )	Opens an existing file
fclose()	Close the file

fscanf()	Reads the data from the file
fprintf()	Writes data in the file
getc()	Reads a character from the file
putc()	Writes a character to the file

Worked out example

Write a program for the following:

1. To calculate the area of four walls of a room. [A=2H(L+B)]

```
#include <stdio.h>
# include <conio.h>
void main( )
{
   float l, b, h, a;
         printf("Enter the length");
            scanf("%f", &l);
         printf("Enter the breadth");
            scanf("%f", &b);
         printf("Enter the height");
            scanf("%f" ,&h);
               a=2*h*(l+b);
         printf("Area of four walls of the room is %f",a);
   getch (  );
}
```

2. To calculate total amount when rate and quantity is given [Amount=Rate x Quantity]

```
#include <stdio.h>
# include <conio.h>
int main( )
{
      float rate,quantity,amount;
```

```
        printf("Enter the rate:");
          scanf("%f", &rate);
        printf("Enter the quantity:");
          scanf("%f", &quantity);
          amount=rate*quantity;
        printf("Total amount is  %f",amount);
      getch (  );
    return 0;
  }
```

3. To display the sum of two integer numbers.

```
#include <stdio.h>
# include <conio.h>
void main( )
{
    int x, y, sum;
        printf("Enter the first number");
          scanf("%d" &x);
        printf("Enter the second number");
          scanf("%d" &y);
          sum=x+y;
        printf("Sum of two numbers= %f",sum);
    getch (  );
}
```

4. To convert the given integer (Seconds) into hour, minute and seconds.

```
#include <stdio.h>
#include <conio.h>
void  main()
{
    int sec, h, m, s;
    printf("Input seconds: ");
      scanf("%d", &sec);
      h = (sec/3600);
      m = (sec -(3600*h))/60;
      s = (sec -(3600*h)-(m*60));
    printf("H:M:S - %d:%d:%d\n",h,m,s);
  getch();
}
```

2.7 Operators in C (Arithmetic Operator, Relational Operator, Logical Operator, Bitwise Operator, Assignment Operator)

Operators are the signs or symbols that execute particular arithmetic and logical operations on constants and variables. The constant and variable on which the operator acts are referred to as operands. C contains a significant number of built-in operators. The different types of operators in C are:

- Arithmetic operators
- Relational operators
- Logical operators
- Assignment operator
- Unary operator
- Short hand operator
- Conditional operator
- Bit wise operator

Arithmetic Operators

Arithmetic operators refer to the operators used to accomplish many sorts of mathematical computations such as addition, subtraction, multiplication, and division. The different arithmetic operators used in the C are

Operator	Description	Example(A=10, B=20)
+	Adds two operands.	A + B = 30
−	Subtracts second operand from the first.	A − B = -10

*	Multiplies both operands.	A * B = 200
/	Divides numerator by de-numerator.	B / A = 2
%	Modulus Operator and remainder of after an integer division.	B % A = 0

Relational Operators

The operators which are used to compare two variables or two values are called relational operators. The different types of relational operators are

Operator	Description	Example(A=10,B=20)
= =	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.	(A == B) is False.
!=	Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.	(A != B) is True.
>	Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true.	(A > B) is False.
<	Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true.	(A < B) is True.
>=	Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true.	(A >= B) is False.
<=	Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true.	(A <= B) is True.

Logical Operators

Logical operators are used to combine and test two or more conditions to make decisions. These operators are used to perform logical operations on the given variables and the logical result will be either true or false. The different types of logical operators are

Operator	Description	Example (A=10,B=20)
&&	Called Logical AND operator. If all the conditions are true, then the result becomes true.	(A>20 && B<30) is false.
||	Called Logical OR Operator. If any one condition is true, then the result becomes true.	(A>20 || B<30) is true.
!	Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.	!(A>20) is true.

The truth table of AND operator is

1st condition	2nd condition	Result
F	F	F
F	T	F
T	F	F
T	T	T

The truth table of OR operator is

1st condition	2nd condition	Result
F	F	F
F	T	T
T	F	T
T	T	T

The truth table of NOT operator is

1st condition	Result
F	T
T	F

Assignment Operator

Assignment operator is used to assign or store value to the variable. The assignment operator used in C is "=".

Example

z=500 [constant value 500 is assigned to or stored in variable z]

t= ‘n’ [character n is assigned to or stored in variable t]

name= "navya" [sting value navyais assigned to or stored in variable name]

Unary Operator

The operators which are used to increases or decreases the value of variable by 1 is called unary operator. ++ and --are the examples of unary operators.

Operator	**Description**	**Example**
++	Increases the value of variable by 1	A++ equivalent to A=A+1
--	Decreases the value of variable by 1	A-- equivalent to A=A-1

Short hand Operator

Short hand operators are the combination of arithmetic and assignment operators. +=, -=, *=, /= and %= are the examples of short hand operator.

Operator	Example	Meaning
+=	a += 10	a = a + 10
-=	a -= 10	a = a – 10
*=	a *= 10	a = a * 10
/=	a /= 10	a = a / 10
%=	a %= 10	a = a % 10

Conditional Operator

The conditional operator also known as ternary operator and denoted by ‘?’ is similar to the if else statement as it does follow the same algorithm of if else statement but the conditional operator takes less space and helps to write the if-else statements in the shortest way possible.

The syntax of a conditional operator is:

Variable-=expression 1 ? expression 2: expression 3

The above syntax can be visualized into if-else statement as:

```
if(Expression1)
{
    variable = Expression2;
}
   else
{
    variable = Expression3;
}
```

Bitwise operators

Bitwise Operators are used for manipulating data at the bit level, also called bit level programming. These operators are used to perform bit operations on given two variables. The different types of bit wise operators are

Operators	Meaning of operators
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
~	Bitwise complement (NOT)
<<	Shift left
>>	Shift right

Expression

An expression is the combination of variables, constants and operators. It is written to perform any calculation or to evaluate any condition. The result may be in the form of numbers, string or logical value or data. Some of the examples of expression are : p= a*b, 15+6, x+y+z, if a>b && a>c etc.

There are three different ways of writing expression.

- Infix notation
- Postfix notation
- Prefix notation

It is the usual way of writing expression in which operators are written between the operands. The expression in which the operator is written between operands is called infix expression. For example P+Q is an infix expression.

Postfix Notation

It is also known as reverse polish notation in which operators are written after their operands. The expression in which the operator is used after operands is called postfix expression. For example PQ+ is an postfix expression.

Prefix Notation

It is also known as polish notation in which operators are written before their operands. The expression in which the operator is used before operands is called a prefix expression. For example +PQ is an prefix expression.

Infix	**Postfix**	**Prefix**
A * B + C / D	A B * C D / +	+ * A B / C D
A * (B + C) / D	A B C + * D /	/ * A + B C D
A * (B + C / D)	A B C D / + *	* A + B / C D

Points to Remember

- C is a high-level Structured Programming Language.
- C has only 32 keywords.
- C is a case-sensitive programming language.
- C language is used to develop Operating System, Language Compilers/Interface, Text Editors, Network Devices, Modern Programs, DBMS, Utilities etc.
- C supports two data types: Basic and Derived
- Basic data type includes int, char, float etc.

Practical works (Specify the activities)

a. Write a program to calculate the volume of the sphere. [v=4/3πr3]

b. Write a program to display the average of any three numbers.

c. To display the sum, product and difference of any two numbers.

d. To calculate the volume of the sphere.[V=4/3 πr3]

e. To calculate the area of four walls of the room.[A=2H(L+B)]

f. To calculate simple interest when principal, time and rate of interest is given.

g. To calculate the area and perimeter of the room.[A=LB, P=2(L+B)]

h. To convert the temperature given in centigrade to Fahrenheit.[F=9C/5+32]

i. To calculate the total surface area of a room. [TSA=2(LB+BH+HL)]

j. To calculate the area of a triangle when three sides are given by the user.

k. [s=a+b+c/2, A=√s(s-a)(s-b)(s-c)]]

l. To calculate the square and cube of a given number.

m. To calculate the area of a cylinder.[A=2ΠR(R+H)]

In the Computer Lab

- Go to the computer lab and open the computer.
- Open BORLAN turbo C software and write any program.
- Run the program and see the output by giving different inputs in the program.
- Save the program by giving the File name.

Exercise

Choose the correct answer from the given alternatives.

1. Who invented C Language.?

   a. Charles Babbage   b. Grahambel

   c. Dennis Ritchie   d. Steve Jobs

2. Which language is a predecessor to C language?

   a. FORTRAN   b. D Language   c. BASIC   d. B Language

3. Which level of language is C?

   a. Low Level   b. High Level   c. Low + High   d. None

4. Which program outputs "Hello World." ?

   a.

```
main()
  {
   scanf("Hello World..");
  }
```

   b.

```
main()
  {
   printf("Hello World..");
  }
```

   c.

```
main()
  {
   print("Hello World..");
  }
```

   d.

```
main()
  {
    scan("Hello World..");
}
```

5. What is the present C Language Standard.?

a. C99 ISO/IEC 9899:1999 b. C11 ISO/IEC 9899:2011

c. C05 ISO/IEC 9899:2005 d. C10 ISO/IEC 9899:2010

6. C language was invented in the year…….

a. 1999 b. 1978 c. 1972 d. 1990

7. What is the output of the C statement.?

```
int main()
{
   int a=0;
   a = 5<2 ? 4 : 3;
     printf("%d",a);
   return 0;
}
```

a. 4 b. 3 c. 5 d. 2

8. What is the output of C Program.?

```
int main()
{
   int a=0;
   a = printf("4");
    printf("%d",a);
   return 0;
}
```

a. 04 b. compiler error c. 40 d. 41

9. Find a correct C Keyword.

a. Float b. Int c. Long d. double

10. Types of Integers are…..

a. short b. int c. long d. All the above

Write short answers to the following questions.

1. What is C? Write any four features of C.
2. List the data types supported by the 'C' language.
3. Explain the structure of the C program.
4. Differentiate between int and float data types in C.
5. What is an operator? Write different types of operators with examples.
6. Define variable? List the different data types in C.
7. Explain the significance of comments in a program.
8. Describe the importance of data types in programming.

Write long answers to the following questions.

1. Describe the basic structure of a C program. Include information about the preprocessor directives, main function, and the significance of header files.
2. Discuss the concept of variables in C programming. Explain the different data types available in C and how they are used in variable declaration. Provide examples.
3. Discuss the significance of input and output operations in C programming. How do these operations contribute to the functionality of a program? Provide examples to illustrate.
4. How operators can be used to perform arithmetic and logical operators in c? Describe with an appropriate example.

Control Flow Statements — Unit 3

3.1 Decision Making Statement

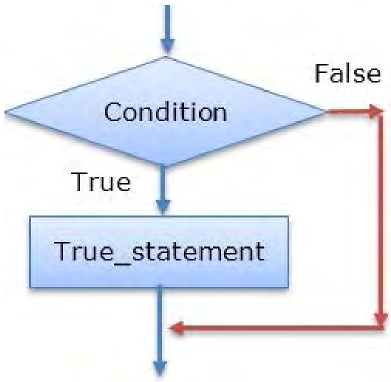
The order of execution of code in a C program is normally from the first statement to the last statement in sequential order. While programming, a programmer may have to make a decision whether to execute the next block of code or not. In other words, a programmer may have to change the flow of a program to execute a specific part of the program. The program control, i.e., the flow of a program in a C program can be controlled or changed by using decision-making/controlling statements like if, if.. else, and switch statements. These decision-making statements evaluate one or more conditions and decide which part of the program is to be executed. The decision making statement is also known as the conditional branching statement.

3.1.1 if Statement

The if statement is a simple decision-making statement. It is used to evaluate the specified condition and decide whether a block of statements is to be executed or not. It executes the certain block of statements, if the specified condition is satisfied or true.

Syntax:

```
if (condition)
{
  // Block_statements
}
Or,
  if (condition)
       statement;
```



Where,

- **condition** is the Boolean expression that is to be evaluated which returns either True or False.
- **Block_statements** enclosed in the curly braces are executed if the result of evaluation of the condition is True.

Note:

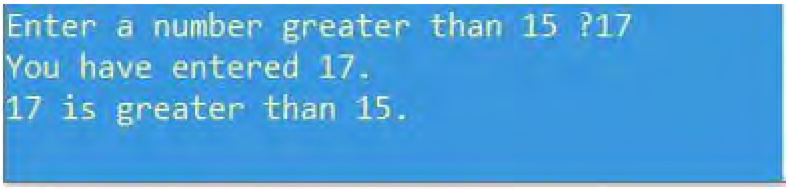
- When statements are not enclosed in the curly braces, it executes the first statement (i.e., which is after if (condition)) by default if the result of evaluation of the condition is True.

Example 12

// program that checks whether a number is greater than 15 or not.

```
#include<stdio.h>
void main()
{
  int a;
    printf("Enter a number greater than 15 \?");
      scanf("%d", &a);
        if (a>15)
    {
      printf("You have entered %d.\n", a);
      printf("%d is greater than 15.\n", a);
    }
}
```

Output



```
Enter a number greater than 15 ?17
You have entered 17.
17 is greater than 15.
```

A user has entered 17 which is greater than 15, so the condition present in the if statement is true. Therefore, the block of statements enclosed in the curly brackets is executed.

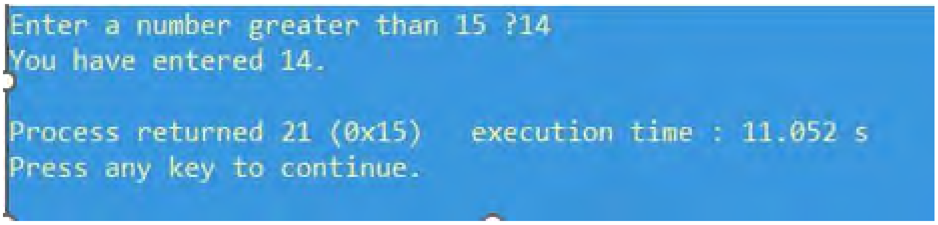
Example 13

// program that checks whether a number is greater than 15 or not.

```
#include<stdio.h>
void main()
{
    int a;
        printf("Enter a number greater than 15 \?");
          scanf("%d", &a);
            if(a>15)
      printf("%d is greater than 15.\n", a);
      printf("You have entered %d.\n", a);
  }
```

Here, the statements are not enclosed in the curly brackets, so it executes the first statement of the program only when the condition is satisfied.

Output



```
Enter a number greater than 15 ?14
You have entered 14.

Process returned 21 (0x15)   execution time : 11.052 s
Press any key to continue.
```

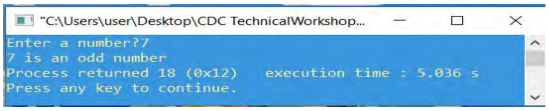
Example 14

// program that checks and displays whether a number is odd or even.

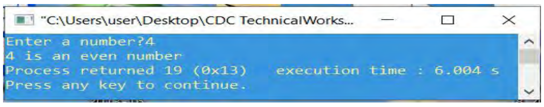
```
  #include<stdio.h>
  void main()
  {
    int a, result;
      printf("Enter a number?");
        scanf("%d", &a);
        result = a%2;
          if(result = =0)
      printf("%d is an even number.", a);
        if(result= =1)
    printf("%d is an odd number.", a);
  }
```

In the above program, **if statement** is used twice for testing even number and odd number separately.

Output



```
"C:\Users\user\Desktop\CDC TechnicalWorkshop...
Enter a number?7
7 is an odd number
Process returned 18 (0x12)   execution time : 5.036 s
Press any key to continue.
```



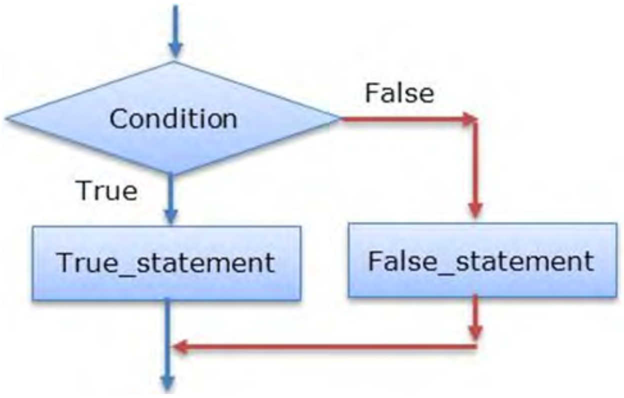
```
"C:\Users\user\Desktop\CDC TechnicalWorks...
Enter a number?4
4 is an even number
Process returned 19 (0x13)   execution time : 6.004 s
Press any key to continue.
```

3.1.2 if..else Statement

The **if statement** decides which block of statements is to be executed when the specified condition is true but it does not tell us what will be executed if the specified condition is false. The **if..else statement** decides which block of statements to execute if the specified condition is true or false. The if ..else statement consists of if block i.e., True_Statements block and **else block** i.e., False_Statements block. The if..else statement executes the True_Statements block if the condition is true (I.e. non zero) and executes **False_Statements block** if the condition is **false** (zero).

Syntax

```
if (condition)
{
  True_Statements;
}
  else
{
  False_Statements;
}
```



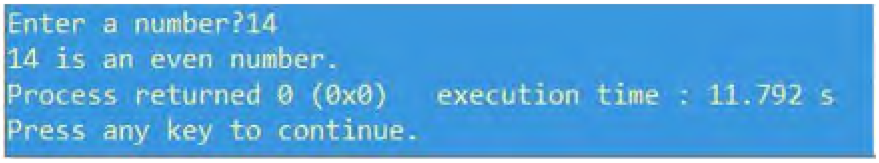
Example 15

// Program that checks and displays whether a number is odd or even.

```
#include<stdio.h>
int main()
{
  int a, result;
      printf ("Enter a number?");
        scanf ("%d", &a);
```

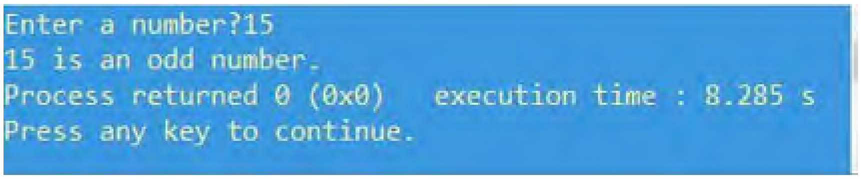
```
          result = a%2;
            if (result = = 0)
          printf ("%d is an even number.", a);
      else
        printf ("%d is an odd number.", a);
      return 0;
    }
```

Output



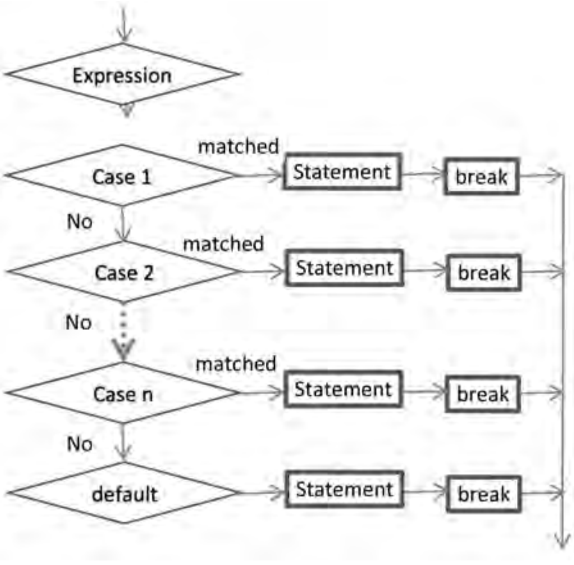
```
Enter a number?14
14 is an even number.
Process returned 0 (0x0)   execution time : 11.792 s
Press any key to continue.
```

Here, since the condition becomes true when a user input 14 as a number, so it executes True_Statements block and displays **14 is an even number**.



```
Enter a number?15
15 is an odd number.
Process returned 0 (0x0)   execution time : 8.285 s
Press any key to continue.
```

Since the condition becomes false when
Statements block and displays **15 is an o**

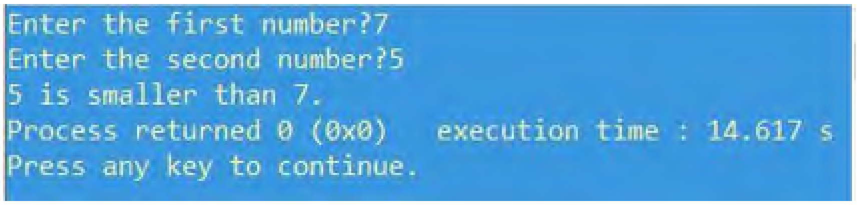


Example 16

// program that displays a smaller num

```
    #include<stdio.h>
    int main()
    {
      int a, b;
        printf("Enter the first number?'
          scanf("%d", &a);
        printf("Enter the second numbe
          scanf(%d", &b);
            if(a<b)
        {
        printf("%d is smaller than %d.", a, b);
    }
        else
        {
          printf("%d is smaller than %d.", b, a);
        }
      return 0;
    }
```

Output



```
Enter the first number?7
Enter the second number?5
5 is smaller than 7.
Process returned 0 (0x0)   execution time : 14.617 s
Press any key to continue.
```

3.1.3 Switch Case Statement

The switch case statement is a multiway branch statement. It evaluates the expression and executes the statements of the one of the matching code blocks among the multiple cases.

Syntax

```
switch (expression)
{
case value1:
      Statements_Block1
  break;
case value2:
      Statements_Block2
  break;
      ……… …..
      ……… …..
  default:
        Default_Statements_Block
    }
```

Where,

- The switch expression must be of **an integer** or **character** type. The integer or character value of expression is compared with the case values.
- The case value must be **an integer** or **character** constant.
- The break statement used inside the switch is the optional that terminates the statement sequence of the switch. If the break statement is not used then all the cases will be executed even the matched case found.
- The default clause inside the switch statement is optional.

Working of Switch Case Statement

- The switch expression is evaluated which yields a character or integer value.
- The character or integer value of the expression is matched with the first case value. If the expression value is matched with the case value then it executes the associated statements block.
- If the **break statement** is used in the **case** then the execution of the switch case is terminated.
- If the first case value is not matched then the expression value is compared with the next case value. If the matching case value is found it executes the corresponding statements block and if not, then the expression value is matched with the next case value and so on.
- If none of the case value is matched with the expression value then it executes block of statements of the default clause.

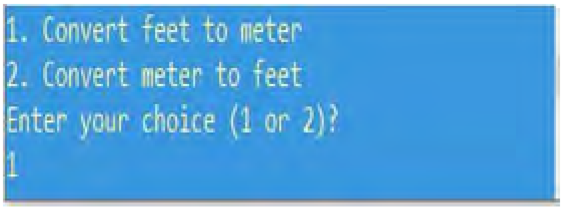
Example 17

// length conversion program

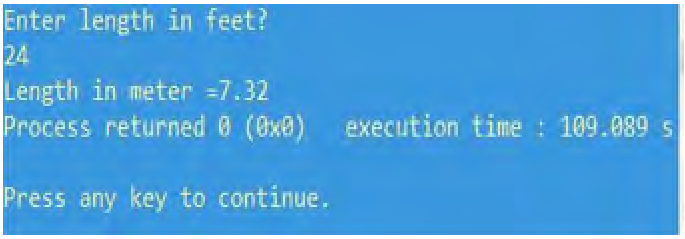
```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int choice;
    float ft, m;
```

```
        printf("1. Convert feet to meter\n");
        printf("2. Convert meter to feet\n");
        printf("Enter your choice (1 or 2)?\n");
        scanf("%d", &choice);
        switch (choice)
        {
            case 1:
                  system("cls");
                  printf("Enter length in feet?\n");
                  scanf("%f", &ft);
                  m=ft/3.28;
                  printf("Length in meter =%0.2f",m);
                  break;
            case 2:
                  system("cls");
                  printf("Enter length in meter?\n");
                  scanf("%f", &m);
                  ft=m*3.28;
                  printf("Length in feet=%0.2f",ft);
                  break;
          default:
                  printf("Enter either 1 or 2");
          }
        return 0;
    }
```

Output



```
1. Convert feet to meter
2. Convert meter to feet
Enter your choice (1 or 2)?
1
```



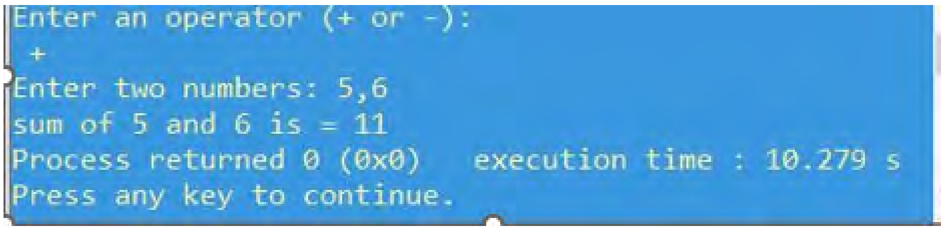
```
Enter length in feet?
24
Length in meter =7.32
Process returned 0 (0x0)   execution time : 109.089 s

Press any key to continue.
```

Example 18

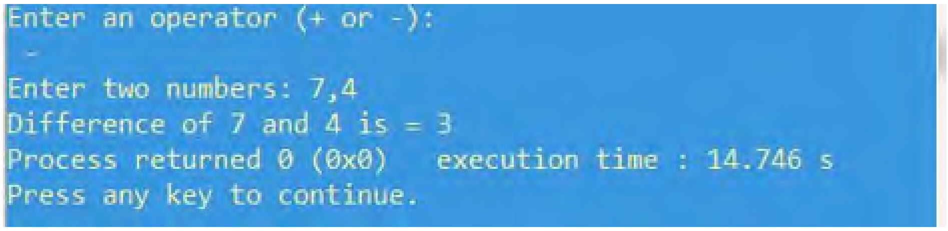
// Program that add or subtract two numbers

```
#include <stdio.h>
int main() {
    char operator;
    int num1, num2, sum, diff;
        printf("Enter an operator (+ or -):\n ");
        scanf("%c", &operator);
        printf("Enter two numbers: ");
        scanf("%d %d",&num1, &num2);
        switch(operator)
        {
             case '+':
             sum=num1+num2;
             printf("sum of %d and %d is = %d",num1, num2, sum);
        break;
             case '-':
             diff=num1-num2;
             printf("Difference of %d and %d is = %d",num1, num2, diff);
        break;
    default:
        printf("This program can add or subtract only.");
    }
  return 0;
}
```

Output



```
Enter an operator (+ or -):
 +
Enter two numbers: 5,6
sum of 5 and 6 is = 11
Process returned 0 (0x0)   execution time : 10.279 s
Press any key to continue.
```



```
Enter an operator (+ or -):
 -
Enter two numbers: 7,4
Difference of 7 and 4 is = 3
Process returned 0 (0x0)   execution time : 14.746 s
Press any key to continue.
```

3.2 Loop Statements

The loop statements enable programmers to reuse the same code again and again once the code is written. The loop statements (also known as iteration control statements) can control the flow of a program and repeat the certain part of the program for a finite number of times or as long as the condition is satisfied or until the condition becomes false. The **for loop**, **while loop** and **do while loop** are the three types of loop statements that are used in C language.

3.2.1 For Loop Statement

The **for loop** statement executes a block of code of a program repeatedly until the specified condition is false. It is used when number of iterations are known.

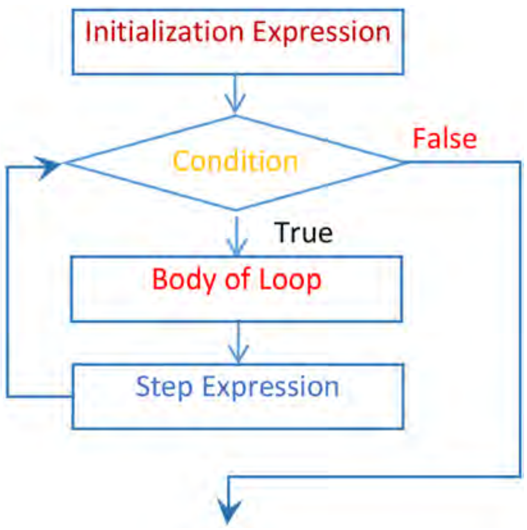
Syntax

```
for (initial_expr; condition; step_expr)
  {
    // Loop Body (statements that are to be executed repeatedly)
  }
```

Where,

- The **initial_expr** is an initialization expression.
- The **conditon** is evaluated before each execution of the for loop body.
- The **step_expr** is the step expression for increasing or decreasing value

Flowchart of for loop statement



Working of for loop

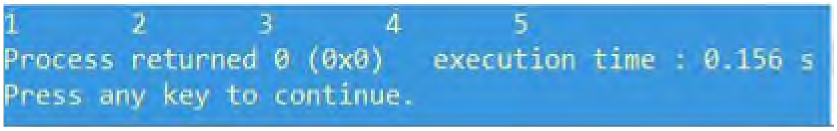
1. The **initial_expr** is executed first. It is executed only once in the looping. It allows a programmer to declare and initialize a loop control variable.
2. The specified condition is evaluated. If the condition is true then it executes the **body of the loop**.
3. After the execution of the loop body, the step_expr generates the new value for the loop control variable and transfers the program control back to the **condition part** for further evaluation.
4. If the condition is unsatisfied then it does not execute the the **body of the loop** and transfers the program control to the next statement that is just after the for loop.

Example 19

// program that generates counting numbers from 1 to 5

```
#include <stdio.h>
int main() {
int i;
    for (i = 1; i <= 5; i++)
    {
        printf("%d\t", i);
      }
    return 0;
  }
```

Output

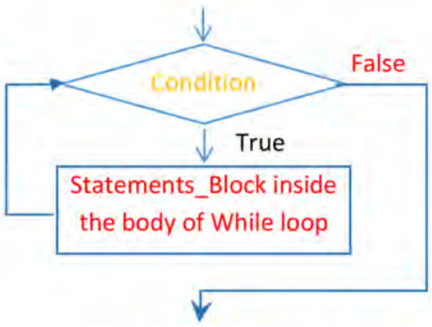


```
1          2          3          4          5
Process returned 0 (0x0)   execution time : 0.156 s
Press any key to continue.
```

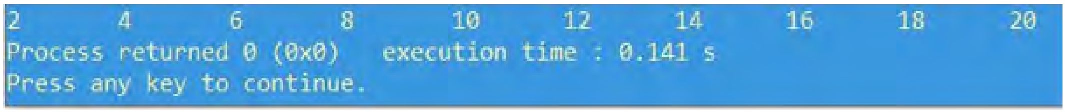
Example 20

// program that generates even numbers from 2 to 20

```
#include <stdio.h>
int main() {
  int n=2;
    for (n ; n <= 20; n=n+2)
    {
      printf("%d\t", n);
    }
  return 0;
}
```



Output



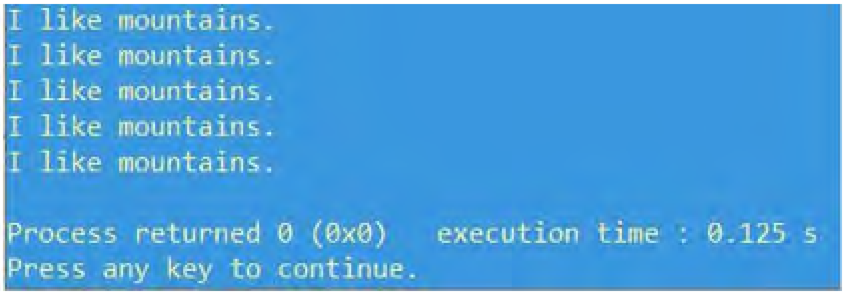
```
2         4         6         8         10        12        14        16        18        20
Process returned 0 (0x0)   execution time : 0.141 s
Press any key to continue.
```

Example 21

// program that displays 'I like mountains' five times

```
#include<stdio.h>
int main()
{
    char str [] = "I like mountains.";
    for (int p=1;p<=5;p++)
    {
      printf("%s\n", str);
    }
    return 0;
}
```

Output



```
I like mountains.
I like mountains.
I like mountains.
I like mountains.
I like mountains.

Process returned 0 (0x0)   execution time : 0.125 s
Press any key to continue.
```

3.2.2 While Loop Statement

The while statement (Iteration statement) executes a statement or block of statements as long as the given condition is True. It is an entry controlled loop that evaluates the condition before processing a statement or block of statements.

Syntax

```
while (Condition){
  Statements_Block
}
```

Where,

- **Condition** is the **boolean expression** that returns either true(I.e., non zero) or false (I.e., zero).
- **Statements_Block** may be a statement or a block of statements that is repeated as long as when the condition is evaluated as true.

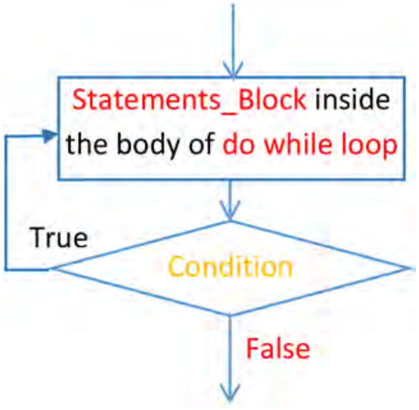
Working of while loop statement

- The **while condition** is evaluated and if the **condition** is **true**, it executes a statement or block of statements.
- After the execution of statements of the body loop, the program control returns back to the while condition part.
- If the **condition** is **false** then it does not execute the statement(s) inside the body of the loop and transfers the program control to the next statement that is just after the while loop.

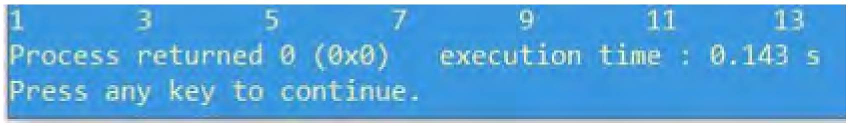
Example 22

// program that generates odd numbers from 1 to 13

```
#include <stdio.h>
int main() {
    int n=1;
      while (n<=13)
      {
          printf("%d\t", n);
          n=n+2;
      }
    return 0;
}
```



Output



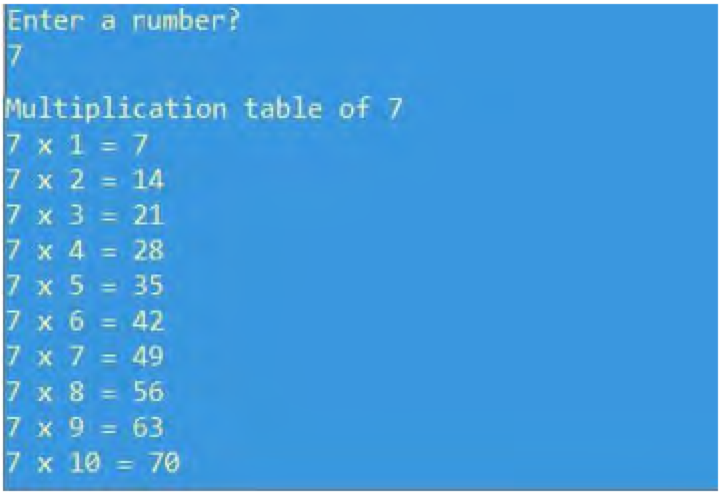
```
1          3          5          7          9          11         13
Process returned 0 (0x0)   execution time : 0.143 s
Press any key to continue.
```

Example 23

// Program that displays the multiplication table of any number.

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int num, count=1;
      printf("Enter a number?\n");
      scanf("%d", &num);
      system("cls");
      printf("Multiplication table of %d\n",num);
      while (count<=10)
      {
        printf("%d x %d=%d\n", num, count, num*count);
        ++count;
    }
  return 0;
}
```

Output



```
Enter a number?
7

Multiplication table of 7
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70
```

3.2.3 do-while Loop Statement

The do while loop statement is the exit control loop statement. It first executes the block of code and then it evaluates the condition. It executes a block of code repeatedly as long as the condition is true or until the condition becomes false.

Syntax

```
do {
    // Block of code that is to be executed
} while (condition);
```

Example 24

// program to display numbers from 5 to 1.

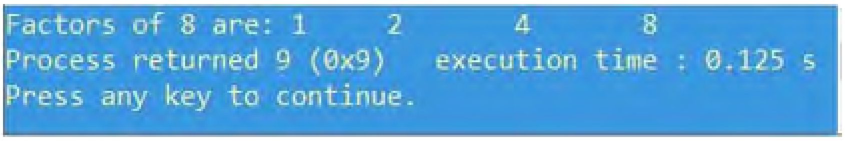
```
#include <stdio.h>
void main(){
    int x=5;
      do {
          printf("%d \t", x);
          --x;
      } while (x>=1);
}
```

Example 25

// Program that displays the factors of 8

```
#include<stdio.h>
void main()
{
    int num=8, r = 0, j=1;
    printf("Factors of %d are: ", num);
    do {
        r = num%j;
        if (r==0)
        printf("%d \t", j);
        j++;
    } while (j<=num);
}
```

Output



```
Factors of 8 are: 1      2       4       8
Process returned 9 (0x9)   execution time : 0.125 s
Press any key to continue.
```

3.2.4 Nested Loops Statement

A loop can be placed inside another loop. The structure where a loop is kept inside another loop is known as nested loop. A loop that encloses another loop is known as the outer loop and the loop that is inside the outer loop is known as the inner loop. For each iteration of the outer loop, the inner loop will be executed. A **for loo**p can be placed inside another f**or loop**. Similarly, a **while loop** or **do while loop** can be placed inside another **while loop** or **do while loop**. A **do while loop** can be placed inside a **for loop** and **vice versa**.

The syntax for a nested **for loop** statement is as follows:

```
for (initial_expr1; condition1; step_expr1)
{
    for (initial_expr2; condition2; step_expr2)
{
```

```
// Inner loop
// Statements2 that are to be executed repeatedly
}
// Outer loop
// Statements1 that are to be executed repeatedly
}
```

The syntax for a nested **while loop** statement is as follows:

```
while (condition1) {
while (condition2) {
statement(s)_2;
}
statement(s)_1;
}
```

The syntax for a nested **do...while loop** statement is as follows:

```
do{
do{
statement(s)_2;
} while(condition2);
statement(s)_1;
} while( condition1);
```

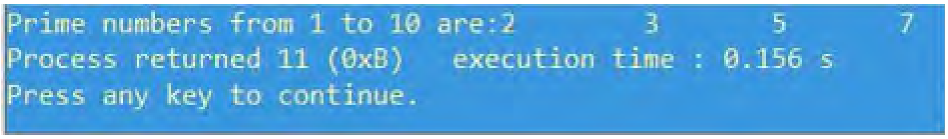
Example 26

// Program that displays prime numbers from 1 to 10

```
#include<stdio.h>
void main() {
  int num, c, r, j;
    printf("Prime numbers from 1 to 10 are:");
      for (num=1; num<=10; num++)
      {
  c=0, r=0;
```

```
        for (int j=1; j<=num; j++)
          {
            r = num % j;
            if (r == 0)
            c=c+1;
          }
        if (c == 2)
        printf("%d\t", num);
      }
    }
```

Output:



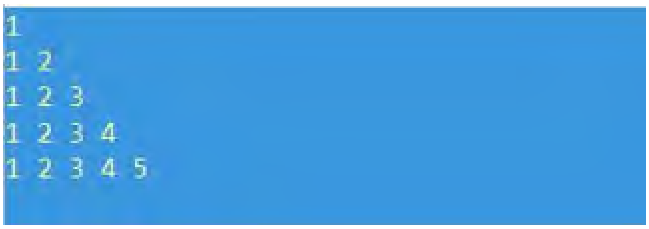
```
Prime numbers from 1 to 10 are:2        3       5       7
Process returned 11 (0xB)   execution time : 0.156 s
Press any key to continue.
```

Example 27

//program that displays patterns of numbers

```
#include <stdio.h>
int main()
{
   int i = 1;
   while (i <= 5) {
      int j = 1;
      while (j <= i) {
         printf("%d ", j);
         j = j + 1;
      }
      i = i + 1;
      printf("\n");
   }
   return 0;
}
```

Output:



```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

3.3 Jump Statement

As you have already learned that the flow of program can be changed by using the conditional branching statements like if, if..else, and switch statements. The transferring of program control (i.e., flow of program) from one statement to another in a program can also be done without any condition. The transferring of program control in a program without any condition is known as unconditional branching/ jumping. The unconditional branching is done by using jump statements like break, continue, goto and return. These jump statements change the normal flow of a program from one part of the program to another unconditionally. These jump statements are used to transfer or jump from one part of a program to another changing the normal flow of program.

3.3.1 Break Statement

The break statement terminates the execution of the switch or loop statement. When the break statement is encountered, it jumps or transfers the program control to the next statement after the switch or loop statement. It is used to bypass the rest of the statements in the loop or switch.

Syntax:

```
break;
```

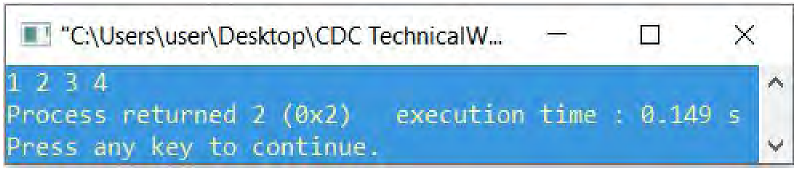
Example 28

// Program that shows the use of break statement

```
#include <stdio.h>
void main(){
  int i;
    for (i = 1; i<= 10; i++) {
      if (i == 5) {
      break;
    }
```

```
        printf("%d ", i);
        }
      return 0;
    }
```

Output



```
"C:\Users\user\Desktop\CDC TechnicalW...
1 2 3 4
Process returned 2 (0x2)   execution time : 0.149 s
Press any key to continue.
```

3.3.2 Continue Statement

The continue statement is used to skip the remaining statement(s) in the current iteration of a loop and execute the next iteration. When the continue statement is encountered, the program control transfers or jumps to the beginning of the loop (i.e. the next iteration) without executing the statement(s) below the continue statement.

Syntax

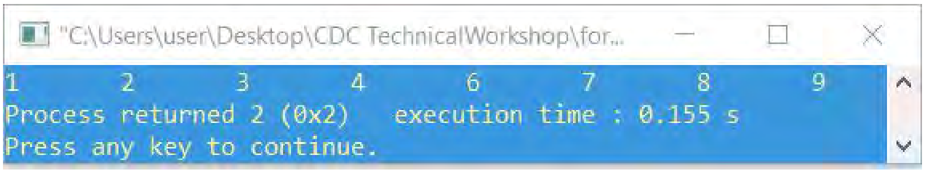
```
continue;
```

Example 29

// program that shows the use of continue statement

```
#include <stdio.h>
void main(){
    for (int i=1; i<10; i++){
    if (i == 5) {
      continue;
      }
    printf("%d\t", i);
  }
}
```

Output



```
"C:\Users\user\Desktop\CDC TechnicalWorkshop\for...
1       2       3       4       6       7       8       9
Process returned 2 (0x2)   execution time : 0.155 s
Press any key to continue.
```

3.3.3 Goto Statement

The goto statement jumps or transfers the program control from anywhere in a program to the specified part in the program.

Syntax:

```
goto label;
……..
……..
label:
```

Where,

- **label** is a user-defined identifier that specifies the location in a program where the program control has to be jumped.

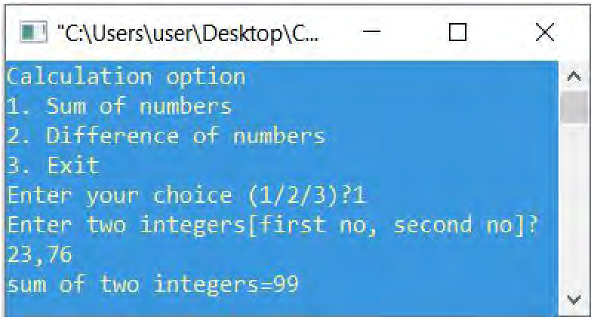
Example 30

// program that shows the use of goto statement

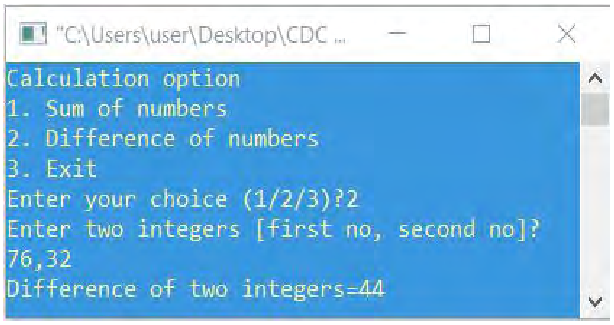
```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
void main(){
    int a,b, choice;
    int t, d;
    top:
        system("cls");
        printf("Calculation option\n");
        printf("1. Sum of numbers\n");
        printf ("2. Difference of numbers\n");
        printf ("3. Exit\n");
        printf("Enter your choice (1/2/3)\?");
        scanf("%d", &choice);
        if (choice==1)
          goto sum;
          else if (choice==2)
          goto diff;
```

```
            else if (choice==3)
            goto buttom;
        else
            printf ("wrong option\n");
        getch();
    goto top;
sum:
            printf("Enter two integers[first no, second no]\?\n");
            scanf("%d, %d",&a,&b);
            t=a+b;
            printf("sum of two integers=%d\n",t);
            getch();
            goto top;
diff:
            printf("Enter two integers [first no, second no]\?\n");
            scanf("%d, %d",&a,&b);
            d=a-b;
            printf("Difference of two integers=%d\n", d);
            getch();
            goto top;
            buttom:
            printf("See you again");
}
```

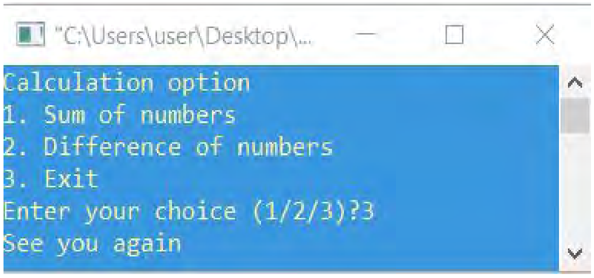
Output



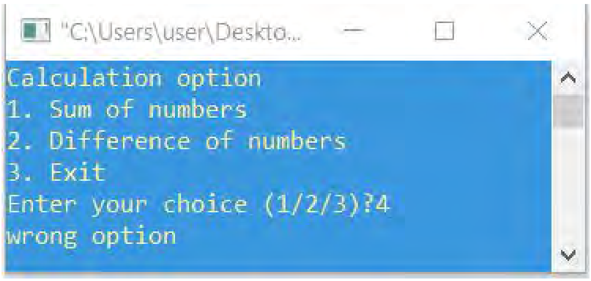
```
Calculation option
1. Sum of numbers
2. Difference of numbers
3. Exit
Enter your choice (1/2/3)?1
Enter two integers[first no, second no]?
23,76
sum of two integers=99
```



```
Calculation option
1. Sum of numbers
2. Difference of numbers
3. Exit
Enter your choice (1/2/3)?2
Enter two integers [first no, second no]?
76,32
Difference of two integers=44
```



```
Calculation option
1. Sum of numbers
2. Difference of numbers
3. Exit
Enter your choice (1/2/3)?3
See you again
```



```
Calculation option
1. Sum of numbers
2. Difference of numbers
3. Exit
Enter your choice (1/2/3)?4
wrong option
```

3.3.4 Return Statement

The **return statement** terminates the execution of a function and returns the program control back to the caller function with a value. If a function has a void return type, then the function does not have to return any value. A non-void return type function must return a value. The return statement can be used at any point within a function to terminate the function and return a value. Multiple return statements can be used within a function, each with a different return value.

Syntax:

return [expression];

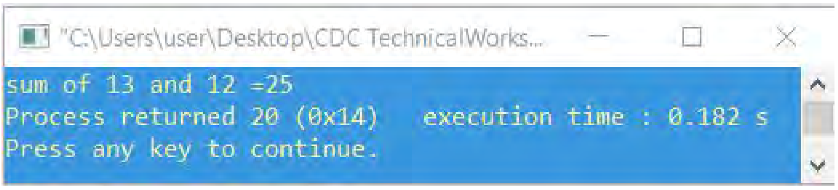
Where,

- **expression** yields a value that has to be returned to a caller function.

Example 31

```
#include<stdio.h>
void main(){
    int x=13, y=12, t=0;
    t = sum(x, y);
    printf("sum of %d and %d =%d", x, y, t);
    }
        int sum (int a, int b) {
        return a+b;
}
```

Output



```
sum of 13 and 12 =25
Process returned 20 (0x14)   execution time : 0.182 s
Press any key to continue.
```

Exercise

Choose the correct answer from the given alternatives.

1. Which is the decision making statement?

   a. If statement b. Do while loop statement
   c. Return statement d. For loop statement

2. Which of the following is post test loop?

   a. Switch case statement b. For loop statement
   c. While loop statement d. Do while loop statement

3. Which one of the following is the jump statement?

   a. Goto b. Break c. Return d. All of the above

4. …… statement is used in the loop or switch to bypass the rest of the code.

   a. Goto b. Continue c. Return d. Break

5. The continue statement is used to ……..

   a. Skip the remaining statements in the current iteration of a loop and execute the next iteration.
   b. Skip the remaining statements in the current iteration of a loop and execute the next statement after the loop statement.
   c. Transfer the program control to the specified part in a program.
   d. None of the above

Write short answers to the following questions.

1. What is flow of program?
2. Why decision-making statement are essential?
3. List any two decision making statements.
4. What is the if statement?
5. Explain the switch statement with example.
6. Define loop. Why it is necessary in programming?
7. List any two loop statements.

8. Write the use of for loop statement.
9. Write the uses of the while loop statement.
10. Write the uses of the do while loop
11. What is nested loop?
12. What is jump statement?
13. List the jump statements used in C.
14. Write the uses of the following statements:

    a. break statement   b. continue statement   c. goto statement

Write long answers to the following questions.

1. Write the use of the if statement.
2. Write the use of the switch statement.
3. Write the use of the loop statement.
4. Write the steps of the working of the for loop.
5. Write the steps of the working of the while loop statement.
6. Differentiate between while and do while loop.
7. Debug the following C programs

a) // check whether a number is lesser than 20 or not.

```
#include<stdio.h>
int main()
{
  int a;
    printf("Enter a number greater than 15 \?");
    scanf("%d", &&a);
      if(a>20)
      {
          printf("You have entered %d.\n", a);
          printf(" is lesser than 20.\n", a);
      }
}
```

b) // Program that checks and displays whether a number is odd or even.

```
#include<stdio.h>
#include<conio.h>
void main()
{
  clrscr();
  int a, result;
      printf ("Enter a number?");
      scanf ("%d", &a);
      result = a / 2;
    if (result = 0)
      printf ("%f is an even number.", a);
    else
      printf ("%f is an odd number.", a);
  return 0;
}
```

8. Write the output of the following programs

a)

```
#include <stdio.h>
void main() {
for (int i=1; i<10; i++){
   if (i == 5) {
      continue;
      }
      printf("%d\t", i);
      }
  }
```

b)

```
#include <stdio.h>
void main() {
int i;
```

```
    for (i = 1; i<= 10; i++) {
      if (i == 5) {
      break;
      }
      printf("%d ", i);
      }
    return 0;
}
```

c)
```
#include <stdio.h>
int main()
{
    int i = 1;
    while (i <= 5) {
    int j = 1;
        while (j <= i) {
             printf("%d ", j);
             j = j + 1;
             }
        i = i + 1;
      printf("\n");
    }
  return 0;
}
```

Practical Works

1. Write the following programs in C:

a) Write a program to add any two numbers.

b) Write a program to display average of any three numbers.

c) Write a program to display the area of a rectangle ground.

d) Write a program to display the perimeter of a rectangle ground.

e) Write a program to display whether an integer input by a user is positive or negative.

f) Write a program to display whether an integer input by a user is odd or even.

g) Write a program to display the greater numbers between any two numbers.

h) Write a program to display the greatest numbers among any three numbers.

i) Write a program to display 'I like my country.' ten times.

j) Write a program to display factors of a number.

k) Write a program to display factorial of a number.

l) Write a program to generate a series of numbers from 1 to 20.

m) Write a program to generate a series of numbers from 5 to 25.

n) Write a program to generate a series of numbers from 100 to 80.

o) Write a program to display even numbers from 2 to 20.

p) Write a program to display first 12 odd numbers.

2. Write the programs in C to generate following numeric patterns:

a)
```
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

b)
```
5 4 3 2 1
5 4 3 2
5 4 3
5 4
5
```

c)
```
5
5 4
5 4 3
5 4 3 2
5 4 3 2 1
```

d)
```
9 7 5 3 1
9 7 5 3
9 7 5
9 7
9
```

e)
```
1 3 5 7 9
1 3 5 7
1 3 5
1 3
1
```

Unit 4 Functions in C

4.1 Introduction to Function

Every C programs consist of functions. Functions are the essential component of C programming. Functions divides a lengthy task into small and manageable group of tasks.

A function in C is a small and manageable block of code that performs a specific task in a program. It is a standalone block of code that can be called from anywhere in the program. When a function is called, it may require to pass or provide data, known as parameters to the function.

Features of Function

A function in C has the following features:

Modularity: A complex task can be divided into many small modules.

Reusability of Code: Once a function is defined in a C program, it can be reused in the program.

Removing Code Redundancy: Once a function is defined in a C program, it is not required to define the function continuously.

Reducing Complexity: A function helps to decrease the length of a program.

4.2 Advantages of Functions

- Functions help break down complex programming into small chunks, making a program more readable and manageable.
- Once a function is defined in a program it can be reused in different parts of the program. It means a function provides code reusability.
- Functions in a program help to reduce the size of a program.
- Functions in a program make debugging easy.
- Functions help in code modularity, which means that the entire code is divided

into separate blocks, each of which is self-contained and performs a different task. This makes each block implementation and debugging much easier.

4.3 Types of Functions

In C programming, there are two types of functions. They are:

a) Library functions

b) User-defined functions

4.3.1 Library Function

The library functions are built-in functions in C programming. They are already defined in header files. They can be used in any C program whenever there is a need to use them. The printf(), scanf(), sqrt(), gets, getch(), etc. are library functions. To use the library functions in programs the associated header files are required to include in programs.

4.3.2 User-Defined Function

A user-defined function is a function that is created by a user to perform some specific tasks in a program. Every C program has a user-defined function 'main ()' function. The main () function is the entry of any C program. When a C program is run, the operating system calls the main () function and the execution of the program starts.

4.4 Functional Aspects

A function in C programming has three general aspects. They are declaration, defining and calling.

4.4.1 Declaration of Function

A user-defined function can be declared in a program. The declaration of function simply means function prototype. The declaration of a function specifies function name, parameters (also known as formal parameters) and return type. The function declaration tells the compiler about the function that the function be used later in the program.

The syntax for declaring a function is as follows:

return_type function_name (**parameters**);

Where,

- **return_type** refers to the data type of value that the function returns.
- **function_name** is the name of function that is to be used in a program.
- **parameters** refer to the variables that are to be used in a function.

Examples

- int **sum** (int, int);
- It declares the **sum** function having two integers as parameters and this function returns an integer value.
- void **diff** (int a, int b);
- It declares the **diff** function having two integers a and b as parameters and this function does not return any value.

4.4.2 Defining of Function

According to the declaration of a function, a user-defined function is required to create in a program. Creating a function as the declaration of the function is known as defining a function. The defining of a function specifies function name, parameters (i.e. formal parameters) with their data types, body of function and the data type of return value.

The syntax for defining a function in C programming is as follow:

```
return_type function_name (parameters) {
    body_of_function;
}
```

Where,

- **return_type** indicates the type of a value that is returned by a function. When a **return_type** is void it does not return any value to the calling function.
- **function_name** refers to the name of function that is to be created.
- **parameters** refer to the list of variables that hold the arguments passed by the calling function. A function may not have any parameters.
- **Body_of_function** refers to the code of a program that is to be executed inside the function on calling.

For example

```
int sum (int a, int b) {
    int t;
    t = a+b;
    return t;
}
```

Here, it defines an integer function named 'sum' with two integer parameters. The sum function returns an integer value.

```
void sum (int a, int b) {
    int t;
    t=a + b;
    printf ("sum of %d and %d = %d", a, b, t);
{
```

Here, it defines a function named sum with two integer parameters and this function does not return any value.

4.4.3 Calling of Function

To use a function in a program, the function is required to call. Calling a function means using the function in a program. When a function is called, the program control is transferred from the calling function to the called function. The called function performs the specific tasks and after the completion of tasks, it returns the program control back to the calling function.

A function can be called in a program just by using the function name followed by the arguments (also known as actual parameters) in parentheses. When a function is called you need to pass or supply arguments (if any) to the function and are stored in parameters (i.e. variables) of the called function.

Syntax of Function Call

```
function_name (argument_list);
```

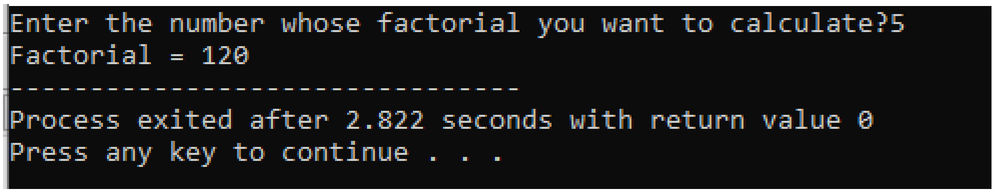
Note: The methods of calling functions i.e. call by value and call by reference are described after the pointers in last chapter.

4.5 Concept of Recursive Functions

A function can be called within the same function in a program. When a function is called within the same function, it is known as recursion of the function. The function that calls itself is known as recursive function.

Example 32

```
#include <stdio.h>
int fact (int);
// function prototype
int main()
{
    int n, f;
    printf("Enter the number whose factorial you want to calculate?");
    scanf("%d",&n);
    f = fact(n);
    printf("Factorial = %d",f);
  }
    int fact(int n)
  {
    if (n==0)
  {
    return 0;
  }
    else if ( n == 1)
  {
    return 1;
  }
    else
  {
  return n*fact(n-1);
  }
}
```



```
Enter the number whose factorial you want to calculate?5
Factorial = 120
--------------------------------
Process exited after 2.822 seconds with return value 0
Press any key to continue . . .
```

Exercise

Choose the correct answer from the given alternatives.

1. Which of the following statements is true about functions in C?

   a. They cannot return values.

   b. They must be declared before they are used.

   c. They cannot take arguments

   d. They are used for repetitive tasks.

2. What is the correct syntax for declaring a function that takes two integer arguments and returns a void (no value)?

   a. void function(int x, int y);

   b. int function(x, y);

   c. void function();

   d. int function x, y);

3. How are arguments passed to a function in C by default?

   a. By reference　　b. By value

   c. It depends on the data type　　d. None of the above

4. What is the purpose of a function prototype?

   a. To define the function body

   b. To declare the function's name, return type, and arguments without the body

   c. To provide comments for the function

   d. To call the function

5. Which keyword is used to return a value from a function in C?

   a. Print　　b. Cutput　　c. Return　　d. Send

6. Can a function call itself in C?

   a. No, this is not allowed.

   b. Yes, this is called recursion.

   c. It depends on the compiler.

d. Only if the function is declared as global.

7. What is the difference between local variables and global variables within functions?
   - a. Local variables are available throughout the program, while global variables are only accessible within the function.
   - b. Global variables are created dynamically, while local variables are statically allocated.
   - c. Local variables are accessible only within the function where they are declared, while global variables can be accessed from anywhere in the program.
   - d. There is no difference.

Write short answers to the following questions.

1. Define user defined functions.
2. Why user defined functions are necessary in c?
3. List in how many ways a function can be declared.
4. How arguments are passed while calling by value?
5. How parameters are passed while calling by reference?

Write long answers to the following questions.

1. What is a function in C?
2. Write any four advantages of functions.
3. Define library function. List any two library functions.
4. Define user defined function.
5. Write any two differences between library function and user-defined function.
6. Write any two difference between call by value and call by reference.
7. What is recursive function?
8. Evaluate the importance of user defined functions over inbuilt functions.
9. Compare and contrast between call by value and call by reference.
10. Discuss different ways of declaring functions in c programming.

Practical Works

Write the following programs in C

1. Write a function to add any two numbers input by a user in the main function.
2. Write a function to display sum of first ten counting numbers.
3. Write a function to display area of a circle whose length and breadth are 19 and 14 respectively.
4. Write a function to display the greater number between any two numbers input by a user in the main function.
5. Write a function to display the average of any three numbers.

Unit 5 Array & String

5.1 Introductionn to the Concept of Array and String

In previous programs we have store a single value at a time in one variable. Some time we need to store multiple values in as single variable. To store multiple similar value in a single variable at a time we need to declare an array. We can declare an array to store integers, float and strings. The array of characters is known as string. Array and string are discussed below:

Array

An array is a special variable which is used to store, organize and manage a group of similar types of value in a single variable name. It is useful for storing more than one value of same data type. Like other variable name an array has also unique name with different subscript or index numbers.

A variable with index number is known as array variable. Each array variable stores value individually. The stored value in the array variable can be manipulated efficiently.

Note: Array can be defined as the collection of heterogeneous data type.

Features of Array

» All the array elements share the common name

» The elements of array stored in contiguous memory location

» The lowest address corresponds to the first element and the highest address to the last element

» It is simple and easy method to handle a large volume of similar data.

Advantages of using array

i. Efficient Access and Retrieval

Storing data in an array makes easier access and retrieval of any stored element. Array allows us to access any element directly using its index. This makes them very efficient for operations that require frequent retrieval of elements. Elements of array are stored in contiguous memory location. Which enhances retrieval speed of the element stored in it.

ii. Memory Efficiency

Arrays are memory-efficient as they allocate a fixed amount of memory for a specified number of elements. This avoids the need for separate memory allocation for each element. Array can store multiple values in a single variable and allocate necessary memory to hold elements minimizing wasted space.

iii. Simplicity and Ease of Use

Array are relatively simple to declare and use. It doesn't require any complex structures and methods for declaring and using an array. Accessing elements is also easier as they can be accessed with a numeric index.

iv. Versatility

Arrays supports versatile data types. They can store elements of different data types including integers, floats, character, strings. We can declare different arrays types for these different data types.

v. Wide range of Application

Arrays are essential for representing matrices, vectors, and other numerical data structures. Arrays are used in numerous sorting and searching algorithms, such as bubble sort, selection sort, insertion sort, and binary search.

String

As like number, alphanumeric value such as letters, number, signs or symbols are also stored in array. String is the series of alphanumeric characters and special symbols stored in an array. The string can be defined as the one-dimensional array of characters terminated by a null ('\0'). The character array or the string is used to manipulate text such as word or sentences. Each character in the array occupies one byte of memory, and the last character must always be 0. The termination character ('\0') is important in a string since it is the only way to identify where the string ends. The difference between a character array and a C string is that the string in C is terminated with a unique character '\0'.

5.2 Declaration and Initialization of Array

Declaration of an Array

In C programming, when you want to group multiple items of the same type together, you can use an array. Here's how you do it:

First, you decide what type of data you want to store in the array, like numbers or letters.
Then, you give a name to your array so you can refer to it later.
Next, you have to specify how many items that an array has to store.
you want your array to hold. This is called the size of the array.
For example, if you want to store 5 numbers, you'd declare an array like this:
If you want to store 5 age we can declare:
int age[5];
if you want to declare 10 fee it can be declared as:
float fee[10];

Initialization of an Array

Initialization of an array refers to defining an array by giving its data type name and setting initial values for the elements of an array. The syntax for initialization of an array is:

Datatype arrayname[size]={element1, element2,…,element};
Eg: int age[5]={4,5,66,7,3};

Accessing Elements of an Array

Elements of an array can be accessed by the index on which they are stored. The index of an array begin from zero 0. For instance, the in above example, array age[5] stores the age of 5 person. And the age are 4,5,66,7,3. And these are stored from index 0 to index 4.

For accessing first element 0 index is used as shown below:
printf("%d",age[0]); will display 4. Similarly printf("%d",age[3]); will display 7.

Accessing Array Elements Using an Array

For accessing all elements we can use an array:

```
for(i=0;i<5;i++)
{
     printf("%d\t",age[i]);
}
```

Here all elements of array age will be displayed using loop. The value of I will start from 0 and will go to upto 4. The result will be displayed as:

```
4	5	66	7	3
```

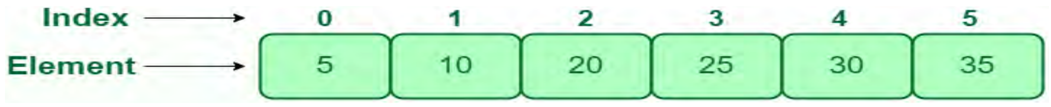
Because age[0]=4

age[1]=5

age[2]=66

age[3]=7

age[4]=3



5.3 Introduction to One-Dimensional Array

One dimensional array is a group of elements having the same data type which are stored in a linear arrangement under a single variable name. It is also known as single arrays and has only one dimension or a single row.

Syntax of One-Dimensional Array in C

The syntax of a *one-dimensional array* in C programming language is as follows:

data Type array name [arraySize];

where

- data_type: is a type of data of each array block.
- array_name: is the name of the array using which we can refer to it.
- array_size: is the number of blocks of memory array going to have.

Eg: int age[10];

Here, int is the data type for integer, age is the name of variable and 10 is the size of array.

Make students to declare 4 one dimensional arrays for:

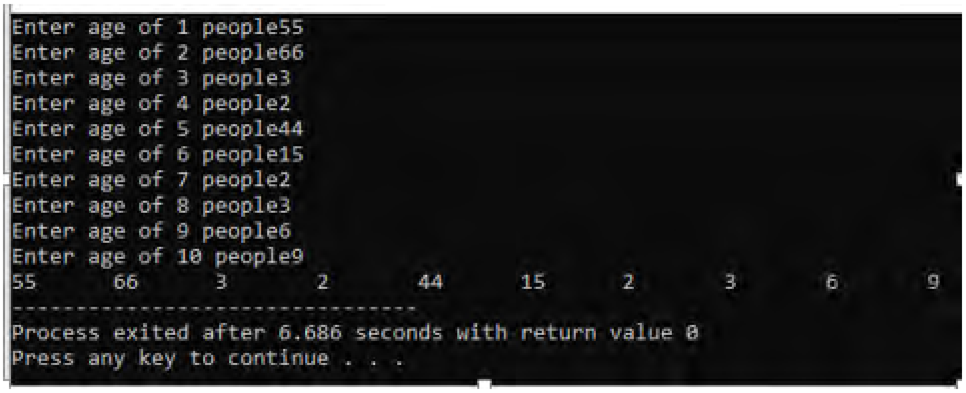
- Storing salary of 10 employee;
- Storing age of 8 students.
- Storing fee of 25 students.
- Storing mark of 5 subjects.

Entering array elements and displaying them.

Example 33

// program for entering age of any 10 students and displaying them using an array.

```
#include<stdio.h>
int main()
{
   int age[10],i;
   for(i=0;i<10;i++)
   {
      printf("Enter age of %d people",i+1);
      scanf("%d",&age[i]);
   }
   for(i=0;i<10;i++)
   {
      printf("%d\t",age[i]);
   }
}
```



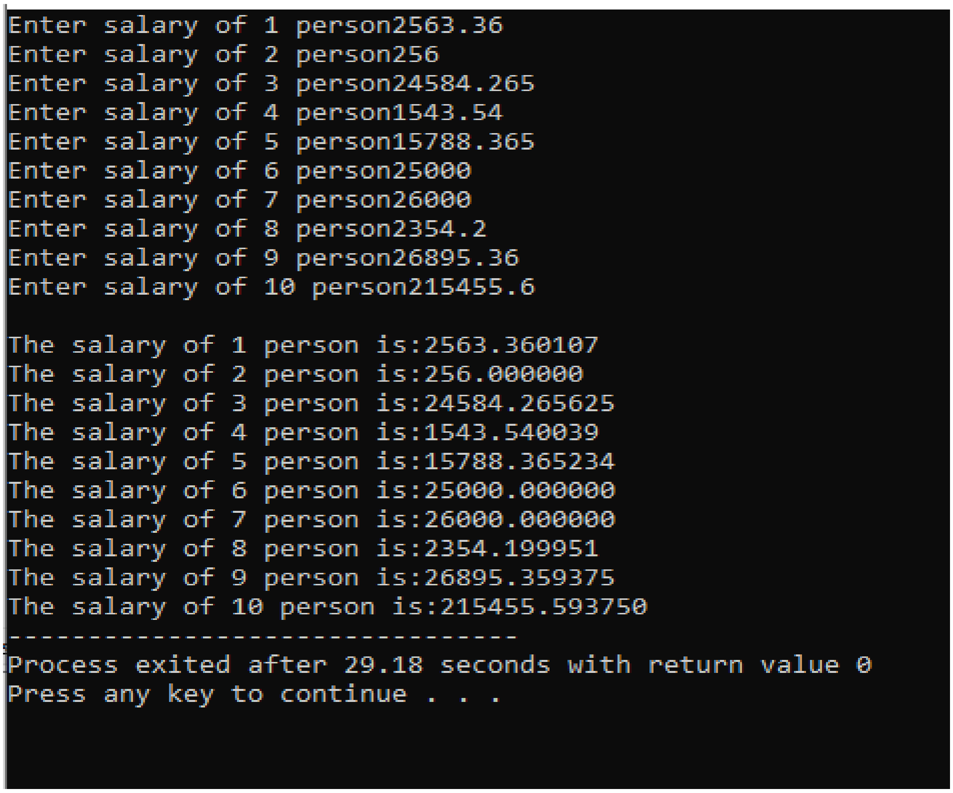
```
Enter age of 1 people55
Enter age of 2 people66
Enter age of 3 people3
Enter age of 4 people2
Enter age of 5 people44
Enter age of 6 people15
Enter age of 7 people2
Enter age of 8 people3
Enter age of 9 people6
Enter age of 10 people9
55      66      3       2       44      15      2       3       6       9
--------------------------------
Process exited after 6.686 seconds with return value 0
Press any key to continue . . .
```

Example 34

Write a c program to enter salary of 10 employees and display them.

```
#include<stdio.h>
int main()
{
      float salary[10];
      int i;
      for(i=0;i<10;i++)
      {
      printf("Enter salary of %d person",i+1);
      scanf("%f",&salary[i]);
      }
      for(i=0;i<10;i++)
      {
        printf("\nThe salary of %d person is:%f",i+1,salary[i]);
      }
}
```

Output



```
Enter salary of 1 person2563.36
Enter salary of 2 person256
Enter salary of 3 person24584.265
Enter salary of 4 person1543.54
Enter salary of 5 person15788.365
Enter salary of 6 person25000
Enter salary of 7 person26000
Enter salary of 8 person2354.2
Enter salary of 9 person26895.36
Enter salary of 10 person215455.6

The salary of 1 person is:2563.360107
The salary of 2 person is:256.000000
The salary of 3 person is:24584.265625
The salary of 4 person is:1543.540039
The salary of 5 person is:15788.365234
The salary of 6 person is:25000.000000
The salary of 7 person is:26000.000000
The salary of 8 person is:2354.199951
The salary of 9 person is:26895.359375
The salary of 10 person is:215455.593750
--------------------------------
Process exited after 29.18 seconds with return value 0
Press any key to continue . . .
```

5.4. Declaration of String

The way a group of integers can be stored in an integer array, similarly a group of characters can be stored in a character array. It is also called strings. String is nothing but a collection of characters in a linear sequence. 'C' always treats a string a single data even though it contains whitespaces. A single character is defined using single quote (' ') representation. A string is represented using double quote marks (" "). For example:

```
char alphabet='A';
char text[]= "Welcome to the world of programming!";
```

The string can be defined as the one-dimensional array of characters terminated by a null ('\0'). The character array or the string is used to manipulate text such as word or sentences. Each character in the array occupies one byte of memory, and the last character must always be 0. The termination character ('\0') is important in a string since it is the only way to identify where the string ends. When we define a string as char name[6], the character name[6] is implicitly initialized with the null in the memory. For example,

```
char name[6]={'G','o','p','a','l','\0'};
```

Each character in the array occupies one byte of memory and the last character is always '\0'. What character is this? It looks like two characters, but it is actually only one character, with the \ indicating that what follows it is something special. '\0' is called null character. Note that '\0' and '0' are not same. ASCII value of '\0' is 0, whereas ASCII value of '0' is 48. Note that the elements of the character array are stored in contiguous memory locations.

5.4.1 Declaring and Initializing a String Variables

A string is a simple array with char as a data type. 'C' language does not directly support string as a data type. Hence, to display a string in 'C', you need to make use of a character array.

The general syntax for declaring a variable as a string is as follows,

```
char string_variable_name [array_size];
```

The classic string declaration can be done as follow:

```
char string_name[string_length] = "string";
```

The size of an array must be defined while declaring a string variable because it used to calculate how many characters are going to be stored inside the string variable. Some valid examples of string declaration are as follows,

```
char name [10];    //declaration of a string variable
```

The above example represents string variables with an array size of 10. This means that the given character array is capable of holding 10 characters at most. The indexing of array begins from 0 hence it will store characters from a 0-9 position. The C compiler automatically adds a NULL character '\0' to the character array created.

Let's study the initialization of a string variable. Following example demonstrates the initialization of a string variable,

char name[15] = "Nitika Devkota";

char name[15]={'N','i','t','i','k','a','','D','e','v','k','o',

't','a','\0'};

// NULL character '\0' is required at end in this declaration

The reason that name had to be 15 elements long is that the string Nitika Devkota contains 14 characters and one element space is provided for the null terminator. Note that when we initialize a character array by listing its elements, we must supply explicitly the null terminator.

C also permits us to initialize a character array without specifying the number of elements. In such cases, the size of the array will be determined automatically, based on the number on the number of elements initialized. For example, the statement

char str1[]={'h','e','l','l','o','\0'} ; /*Declaration as set of characters ,Size 6*/

char str2[]="World";   /* string size = 'w'+'o'+'r'+'l'+'d'+"NULL" = 6 */

We can also declare the size much larger than the string size in the initializer. That is, the statement.

char str[12] = {"String"} ; is permitted. In this case, the computer creates a character array of size 12 places the value "String" in it, terminates with the null character, and initializes all other elements to NULL. The storage will look like

S	t	r	i	n	g	\0	\0	\0	\0	\0	\0

However, the following declaration is illegal.

char str[5] = {"String"}; this will result in a compile time error.

5.4.2 Reading String from Terminal

When writing interactive programs which ask the user for input, C provides the scanf(), gets(), fgets() and getchar() functions to find a line of text entered from the user.

Using scanf()function

When we use scanf() to read, we use the "%s" format specifier without using the "&" to access the variable address because an array name acts as a pointer.

Example 35

```
#include <stdio.h>
int main() {
    char name[10];
    printf("Enter your Name\n");
    scanf("%s", name);
    printf("Name: %s",name);
    return 0;
}
```

Output

```
Enter your  name
Bhanu
Name:Bhanu
```

The problem with the scanf function is that it never reads an entire string. It will halt the reading process as soon as whitespace, form feed, vertical tab, newline or a carriage return occurs. Suppose we give input as "Kathmandu Nepal" then the scanf function will never read an entire string as a whitespace character occurs between the two names. The scanf function will only read "Kathmandu".

Using gets() or fgets() function

In order to read a string contains spaces, we use the gets() function. Gets ignores the whitespaces. It stops reading when a newline is reached (the Enter key is pressed). Unlike scanf(), it does not skip whitespaces.

Example 36

```
#include <stdio.h>

int main() {

char name[50];

printf("Enter your full name: ");
gets(name);
printf("My full name is %s ", name);
return 0;
}
```

Output

```
Enter your full name: Arogya Khatiwada
My full name is Arogya Khatiwada
```

Another safer alternative to gets() is fgets() function which reads a specified number of characters. For example:

```
#include <stdio.h>
int main() {
    char name[20];
    printf("Enter your full  name: ");
    fgets(name, size of (name), stdin);
    printf("My name is %s ",name);
    return 0;
}
```

Output

```
Enter your full name: Arohi Shrestha
My name is Arohi Shrestha
```

The fgets() arguments are :

First argument: the string name,
Second argument: the number of characters to read, and
Third argument: **stdin** means to read from the standard input which is the keyboard.

Using getchar() function

getchar is a function in C programming language that reads a single character from the standard input stream stdin, regardless of what it is, and returns it to the program.

Example 37

```
#include <stdio.h>
int main() {
    char ans;
    printf("Would you like to Know my name?\n ");
    printf("Type Y for yes and N for no:");
    ans=getchar();
    if(ans=='Y'|| ans=='y')
    printf("\n My Name is Kaushal");
  else
    printf("\n You are doing well");
  return 0;
}
```

Output

```
Would you like to know my name?
Type Y for yes and N for no: Y
My name is Kaushal
```

5.4.3 Writing String to Screen

Using printf() function

We have used extensively the prinf function with %s format to print strings to the screen. The format %s can be used to display an array of characters that is terminated by null character. For example, the statement

```
printf("%s", name);
```

can be used to display the entire contents of the array name. We can also specify the precision with which the array is displayed. For instance, the specification **%12.5** indicates that the first five characters are to be printed in a field with of 10 columns.

The printf on UNIX supports another nice feature that allows for variable field width or precision. For instance

```
printf("%*.*s\n", w, d, string);
```

prints the first **d** characters of the string in the field width of **w**.

The following features of the "%*.*s specifications.

- When the field width is less than the length of the string, the entire string is printed.
- The integer value on the right side of the decimal point specifies the number of characters to be printed.
- When the number of characters to be printed is specified as zero, nothing is printed.
- The minus sign in the specification causes the string to be printed left-justified.
- The specification %.ns prints the first n characters of the string.

Example 38

```
#include <stdio.h>

#include <stdlib.h>

int main()

{

    char name[15]={"Kathmandu Nepal"};

    printf("%s\n",name);

    printf("%5s\n", name);

    printf("%15.9s\n",name);

    printf("%-15.9s\n",name);

    printf("%15.0s\n",name);

    printf("%.9s\n",name);

    printf("%*.*s\n",15,9, name);

    printf("%*.*s\n",-15,9, name);

    return 0;

}
```

Output

```
Kathmandu Nepal
Kathmandu Nepal
              Kathmandu
Kathmandu


Kathmandu
              Kathmandu
Kathmandu
```

Using puts or fputs() function

The puts function prints the string on an output device and moves the cursor back to the first position.

Example 39

```
#include <stdio.h>
int main() {
   char name[15];
   printf("Enter your name: ");
   gets(name);     //reads a string
   printf("My name is ")
   puts(name);     //displays a string
  return 0;
}
```

Output

```
Enter your name: Kumar Mahato
My name is Kumar Mahato
```

fputs() function

The fputs() needs the name of the string and a pointer to where you want to display the text. We use stdout which refers to the standard output in order to print to the screen.

Example 40

```
#include <stdio.h>
int main()
{
    char town[40];
    printf("Enter your town: ");
    gets(town);
    fputs(town, stdout);
    return 0;
}
```

Output

```
Enter your town: New York
New York
```

Using putchar() function

Like getchat(), C supports another character handling function putchar() to output the values of character variables. It takes the following form:

```
char ch='A';
putchar(ch);
```

The function putchar requires one parameter. This statement is equivalent to

```
printf("%c", ch);
```

Example 41

```
#include <stdio.h>
int main()
{
    int i;
    char name[6]={"Kapil"};
    for(i=0;i<5; i++)
    {
            putchar(name[i]);
        }
        putchar('\n');
        printf("%s\n", name);
      return 0;
    }
```

Output

```
Kapil
Kapil
```

5.5 String Functions

(strlen(), strcpy(), strcat(), strcmp(), strrev(),strlwr(), strupr())

The C library supports a large numbers of string handling functions that can be used to carry out many of the sting manipulation. The header file **#include<string.h>** is used for string manipulation functions. Some of the common string manipulation functions are below:

Function	Description
strlen(string_name)	Returns the length of string name
strrev(string_name)	Returns reverse string.
strlwr(string_name)	Returns string characters lowercase.
strupr(string_name)	Returns string characters uppercase.
strcpy(destination, source)	Copies the contents of source string to destination string.
strcat(frist_string, second_string)	Concatenates or joins first string with second string. The result of the string is stored in first string.
strcmp(first_string , second_string)	Compares the first string with second string. If both string are same, it returns 0.

We shall discuss briefly each of these functions can be used in the processing of strings.

strlen() function

The **strlen()** function return the length of a string which takes the string name as an argument.

Syntax: l=strlen(str);

Where **str** is the name of string and **l** in the length of the string, returned by strlen function.

Example 42

// simple example of strlen() function,

```
#include<stdio.h>
#include <string.h>
int main(){
    char str[10];
    int l;
    printf("Enter a string: ");
    gets(str);
    l=strlen(str);
    printf("The length of string is: %d", l);
  return 0;
}
```

Output

```
Enter a string: Hello word
The length of string is 10;
```

strrev() function

The strrev(string) function returns reverse of the given string.

Syntax

```
strrev(string);
```

Example 43

// a simple example of strrev() function.

```
#include<stdio.h>
#include <string.h>
```

```
int main(){
    char str[10];
    printf("Enter a string: ");
    gets(str);
    strrev(str);
    printf("Reverse String is %s", str);
  return 0;
}
```

Output

```
Enter a string: String
Reverse String is gnirtS
```

strlwr() function

The strlwr() function is used to convert upper case characters of string into lower case character

Syntax

```
strlwr(string);
```

Example 44

// a simple example of strlwr() function.

```
#include<stdio.h>
#include <string.h>
int main(){
    char str[10];
    printf("Enter a string: ");
    gets(str);
    strlwr(str);
    printf("Lower String is %s", str);
  return 0;
}
```

Output

```
Enter a string: LOWER CASE
Lower String is lower case
```

strupr() function

The strupr() function is used to convert lower case characters of string into upper case character

Syntax:

```
strupr(string);
```

Example 45

// a simple example of strupr() function.

```
#include<stdio.h>
#include <string.h>
int main(){
    char str[10];
    printf("Enter a string: ");
    gets(str);
    strupr(str);
    printf("Upper String is %s", str);
  return 0;
}
```

Output

```
Enter a string: upper
Upper string is UPPER
```

strcpy()Function

The strcpy() function is used to copy one string into another string

Syntax

```
strcpy(destination_ sting , source_string);
```

Example 46

// a simple example of strcpy() function.

```
#include<stdio.h>
#include <string.h>
int main(){
    char str1[10], str2[10];
    printf("Enter a first string: ");
    gets(str1);
    strcpy(str2, str1);
    printf("Second String is %s", str2);
  return 0;
}
```

Output

```
Enter a first String: Nepal
Second String is Nepal
```

strcat() function

The **strcat()** function is used to join or concatenate one string into other string.

Syntax: strcat(first_string, second_string)

Example 47

// a simple example of strcpy() function.

```
#include<stdio.h>
#include <string.h>
int main(){
    char str1[10], str2[10];
    printf("Enter a first string: ");
    gets(str1);
    printf("Enter a second string: ");
    gets(str2);
    strcat(str1, str2);
```

```
        printf("Concatenate String is %s", str1);
      return 0;
    }
```

Output

```
    Enter a first string: Sita
    Enter a second string: ram
    Concatenate String is Sitaram
```

strcmp() function

The strcmp() function is used to compare two string, character by character and stops comparison when there is a difference in the ASCII value or the end or any one sting and returns ASCII difference of the characters that is integer.

Syntax: v= strcmp(first_string, second_string)

Example 48

// a simple example of strcmp() function.

```
    #include<stdio.h>
    #include <string.h>
    int main(){
        char str1[10], Str2[10];
        int v;
        printf("Enter a first string : ");
        gets(str1);
        printf("Enter a second string : ");
        gets("str2);
        v=strcmp(str1, str2);
        if(v==0)
        {
        printf("Both string are same");
        }
        else if(v>0)
        {
        printf("First string comes after string second");
        }
        else
        {
        printf("Second string comes after string first ");
        }
      return 0;
    }
```

Output

```
Enter a first string: Sunita
Enter a second string: Samita
First string comes after string Second
```

Exercise

Choose the correct answer from the given alternatives.

1. Which of the following is correct syntax for declaring an array?

   a. Datatype array name[size]   b. Datatype [size]arrayname

   c. Arrayname[size]datatype   d. All of the above

2. The index of array element begins with

   a. 0   b. 1   c. 2   d. 3

3. The function of strcpy() function is .

   a. To find length of stirng   b. To copy the string

   c.To reverse string d. To delete string

4. The use of strlen() function is ……….

   a. To find length of string   b. To display all characters.

   c. To get string as input   d. None of the above

5. The string is terminated with …….character.

   a. /0   b. /1   c. /2   d. /3

Write short answers to the following questions.

1. Define an array. List the merits of using an array.
2. What is a string? Why we need string.
3. Initialize an array named students with size 25.
4. Declare an array named salary with 5 elements.
5. What is one dimensional array? Explain with examples.
6. Declare a string named name.
7. What are string functions? In which header file does they resides.
8. Explain various string handling functions with example.
9. Discuss different string functions in c programming language.

Lab Work

Write c program to:

1. Enter salary of 10 employees and display them.
2. Read the age of all students from your class and display them.
3. Enter fee of 25 students and display them.
4. Enter two string and display their length.
5. Enter one string display it in reverse order.
6. Enter a string and display it in lower case.
7. Enter a string and display in upper case.
8. Enter a word and copy it in another string.
9. Enter two words and join them.
10. Enter two words and join them.
11. Enter a word and display it in reverse order.

Structure and Union

Unit 6

6.1 Introduction to Structure

We can use an array to store elements having similar data type. If we need to store different data type elements under same name then we use structure. Structure is the collection of different data types.

In another words, Structure in C is a user-defined data type that enables us to store the collection of different data types. It means a single structure is a collection of heterogeneous data types. The different types of data type that is hold by structure are called member of structure. The keyword ***struct*** is used to declare structure variable and type of structure variable is defined by the tag name of the structure.

Features of structure

» Structure can have different data type elements.

» We can create structure variables.

» Structure elements are accessed by . operator

» We can create array of an structure.

To define a structure, struct statement is used. The struct statement defines a new data type, with more than one member. The syntax of the struct statement is as follows:

6.2 Declaration of Structure and Structure Variable

Declaration of structure refers to giving name of the structure, data type of variables and name of variables which data is to be stored in structure.

In C, declaring a structure is like building a box to hold related data together. This box has compartments for different types of data, making your code cleaner and easier to use multiple times.

Syntax for declaring structure and structure variable:

```
struct structure_name
{
    data_type member 1;
    data_type member 2;
    data_type member 3;
}
struct structure_name variable 1, variable 2…………….. Variable;
```

Example

```
struct employee
    {
    int id;
    char name [25];
    float salary;
    }
    struct employee s1,s2,s3;
```

Here, structure name of structure is employee and it has three member id, name and salary. And also s1, s2 and s3 are structure variables and it is initialized later.

6.3 Accessing Member of Structure Variable

Access member operator is used to access elements of structure. The member access operator is a period (.). The period is placed between the structure variable name and the structure member that we want to access.

Syntax

```
variable.member
```

Example

```
s1.id, s2.name, s3.salary
```

Example 49

Write a program to store id, name and salary of employee and display them.

```
# include <stdio.h>
# include <string.h>
struct employee {
     int  id;
    char name [20];
     float salary;
     };
     void main ( ) {
     struct employee s1;
     printf ("\n Enter employee id");
     scanf ("%d", & s1.id);
     printf ("\n Enter employee name");
     scanf ("%s",&s1.name);
     printf ("\n Enter salary");
     scanf ("%f",& s1.salary);
printf ("Staff id %d \n employee name %s \n salary is %f",s1.id, s1.name,s1.salary);
}
```

Output

```
Enter employee id 901
Enter employee name Ram
Enter employee name 20500
Staff id 901
Staff name Ram
Salary is 20500.000000
```

6.4. Introduction to Union

A union is similar to a structure, but it's different. In a structure, each part has its own memory space, while in a union, all parts share the same memory space. A union is a data type that can store various data types in one memory location. We use the keyword "union" to define it. Although we can have many parts in a union, only one part can hold a value at a time. When a new value is stored, it replaces the previous one. Unions are useful for efficiently using the same memory space for different purposes. A union is a data type that allows to stores different data types in the same memory location. The keyword union is used to define the union. We can define many members in a union, but only one member can contain a value at any given time. It overwrites the data previously stored in the union. Unions provide an efficient way of using the same memory location for multiple-purpose.

6.5. Declaration of Union and Union Variable

Declaration of union and union variable

To define a union, union statement is used. The union statement defines a new data type with more than one member. The syntax of the union statement is as follows:

Syntax

```
union union_name
    {
    data_type member 1;
    data_type member 2;
    data_type member 3;
    }
union union_name variable 1, variable 2……………variable;
```

Any meaningful name tag name is given to union. The member of union is defined as like a variable proceeding with their data type such as inti; or float f; or any other valid variable definition. One or more one union variables are defined at the end of the structure's definition and ended with semicolon

Example

```
union employee
    {
    int id;
    char name [20];
    float salary;
    }
    Union employee s;
```

Here, tag name of union is employee and it variable name is s. It consists three members int id, char name [20] and float salary. The largest size of member of this union is char name [20] which occupies 20 bytes memory of system. The memory size of union is 20 for this example. So, all the rest two members int id and float salary share the same memory.

Accessing Union Members

To access any member of a union, we use the member access operator. The member access operator is a period (.). The period is placed between the union variable name and the union member that we want to access. But only one variable with member can be used at a time.

Syntax

```
variable.member
s.id, s.name, s.salary
```

Example 50

Write a program to store and display the data such as id, name and salary of a employee.

```
# include<stdio.h>
# include<string.h>
union employee {
    int  id;
    char name [20];
    float salary;
    };
void main( ) {
    union employee s;
    printf ("\n Enter id");
    scanf ("%d",& s.id);
    printf ("\n Enter name");
    scanf ("%s", & s.name);
    printf ("\n Enter salary");
    scanf ("%f", & s.salary);
printf ("\n Staff id %d \n employee name %s \n salary is %f", s.id, s.name, s.salary);
}
```

Output

```
Enter id 123
Enter name hari
Enter salary 20000
Staff id 1184645120
employee name
salary is 20000.000000
```

Here, we can see that the values of id and name members of union got corrupted because the final value assigned to the variables. Salary has occupied the memory location. So, that the value of salary member is getting printed very well.

Difference between Structure and Union

Structure	**Union**
Structure is the container to store data variables of different type and also supports for the user defined variables storage.	Union is containers which can also holds the different type of variables along with the user defined variables is same memory.
Structure in C is internally implemented as that there is separate memory location is allotted to each input member	Union memory is allocated only to one member having largest size among all other input variables and the same location is being get shared among all of these.

Syntax:
structtag_name
{
data_type member1;
data_type member2;
data_type member3;
}var;
	Syntax:
	union tag_name
	{
	data_type member 1;
	data_type member 2;
	data_type member 3;
	} var;

Memory size structure is equal or greater than the sum of size of all the data members.	Memory size union is equal to the size of largest member among all data members.
There is specific memory location for each input data member and hence it can store multiple values of the different members.	There is only one shared memory allocation for all input data members so it stores a single value at a time for all members.
Multiple members can be initializing at same time.	Union only the first member can get initialize at a time.

Exercise

Choose the correct answer from the given alternatives.

1. Which of the following keyword is used to declare structure?

   a. structure b. struct c. Both i and ii d. None of above

2. Structure is the collection of ………..data types.

   a. Heterogenous b. Homogenous c. Similar d. None of above

3. Which of the following is correct while declaring structure?

   a. structure employee{int x,y}

   b. struct employee;

   c. struct employee{int x,y;};

   d. struct employee{int x,y};

4. Data elements of structure are accessed using………operator.

   a. .

   b. *

   c. /

   d. +

5. What is the main difference between structure and union?

   a. Structure stores different data type but union stores similar data type.

   b. Structure uses same memory location but union uses different memory for variables.

   c. Structure uses different memory location but union shares same memory

   d. All of the above

Write short answers to the following questions.

1. Define a structure. Write syntax for declaring structure and structure variable.
2. How the member variable of structure is accessed? Describe with example.
3. Define union.
4. Explain syntax of union and union variable with example.

5. Describe with example how members of union are accessed.
6. Differentiate between structure and unio.

Project Work

1. Store name, age and fee of 10 students and display them using structure.
2. Store employee id, name, salary, post of 15 employee and display them using structure.
3. Enter name, age and fee of 10 students and display them using union.
4. Enter employee id, name, salary, post of 15 employee and display tem using unio.

Pointer

Unit 7

7.1 Introduction to Pointer

The pointer in C language is a variable which stores the address of another variable. It directs address of the memory location of another variable. This variable may be of type int, char, array, function, or any other pointer. Simply, pointers are symbolic representation of addresses.

In C programming, pointer is essential because some of specific tasks are performed more easily with pointers such as dynamic memory allocation, file handling etc. It cannot be performed without using pointers. Pointers in C language are widely used in arrays, functions, and structures. It reduces the code and improves the performance. It is important to learn pointers because it is the most powerful and useful tools of C programming. The concepts of pointer makes C language distinct form other programming language. As we know, every variable is a memory location and every memory location has its address defined which can be pointed and accessed by using the help of pointer.

Advantages of Pointer

- » Pointer reduces the code and improves the performance.
- » Pointer saves memory space.
- » Pointer increases execution time because of manipulation of data is done inside the memory.
- » It is used to retrieving strings, trees, etc. and used with arrays, structures, and functions.
- » We can return multiple values from a function using the pointer.
- » It makes us able to access any memory location in the computer's memory.
- » It is efficient for solving hardware interfacing problem.

7.2 Declaration of Pointer and Pointer Variable

As like a variable, pointer must be defined before using it. The syntax of the pointer is as like follows:

Syntax

```
data_type *variable_name
```

data_type is type of valid data type of C language such as int, float, char etc. var_name is the name of the pointer variable. The asterisk*used to declare a pointer.

Example

```
int n = 100;
int *ptr;
ptr = &n;
```

In above example, when a variable n is declared, it allocates the memory of computer and store the 100. The address of the variable n is 2042 which is shown in figure. When the pointer variable *ptr is declared, it also allocates the memory of the computer which memory address is 2602 as like shown in figure. It is also used to store value but address of variable n that is 2042. It means the pointer is used to point the address of another variable.

Two types of operator are used in pointer. They are (&) and (*)

Unary operator (&) is used to assign the address of a variable to a pointer. So, it returns the address of that variable to the pointer. In the statement ptr = &n, the pointer ptr is assigning address of the variable n in the above example.

7.3 Referencing and Differencing

Referencing: Referencing is storing the address of a variable into a pointer variable. For referencing address of (&) operator is used.

Dereferencing: Dereferencing operator is used to access the value stored in a memory. We use asterisk (*) for dereferencing.

Unary operator (*) is used to accessing the value stored in the address. So, it returns the value of the variable located at the address specified by its operand. In the statement int *ptr, the integer variable ptr is declared as pointer, which is able to access address of any variable assign with it.

Example 51

// a program to show pointer arithmetic.

```
# include<stdio.h>
void main ( )
{
    int n = 100;
    int* ptr;
    ptr = &n;
      printf ("\n The value of variable n is: %d", n);
      printf ("\n The address of variable n is: %d", ptr);
      printf ("\n The address of variable n is: %d", & n);
      printf ("\n The address of pointer ptr is: %d", & ptr);
      printf ("\n The value of variable n is: %d",*ptr);
    printf ("\n The value of variable n is: %d",*(& n));
}
```

Output

```
The value of variable  n is: 100
The address of variable  n is: 2424396
The address of variable  n is: 2424396
The address of pointer ptris: 2424384
The value of variable  n is: 100
The value of variable  n is: 100
```

7.4 Advantages of Pointer

- Pointer provide direct access to the memory.
- Pointer provide a way to returns more than one value to the functions.
- Pointers provides an alternate way to access array elements.
- Pointers reduces the storage space and complexity of the program.
- Pointer reduces the execution of the program.

7.5 Types of Function Call

There are two methods of passing arguments while calling a function in C. They are:

a) Call by Value

b) Call by Reference

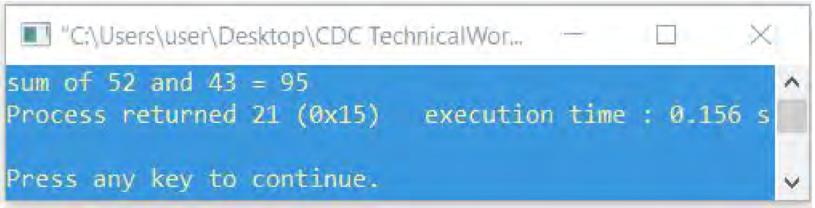
7.5.1 Call by Value

It is the default method of passing arguments into a function. In the call by value method the actual values of arguments are passed/copied to the parameters of the called function. Since the values of arguments are copied to the parameters, arguments and parameters have different memory addresses. Any changes made inside the called function does not affect the arguments.

Example 52

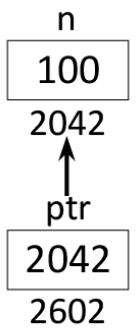
```
#include<stdio.h>
  int sum (int, int);           // Declaration of function i.e. function prototype
void main(){
    int x, y, total;
    x=52;
    y=43;
total = sum (x, y);             // Calling a function by value method
printf ("sum of %d and %d = %d", x, y, total);
}
    int sum (int a, int b) {
    int t;
    t = a + b;
    return t;
}
```

Output



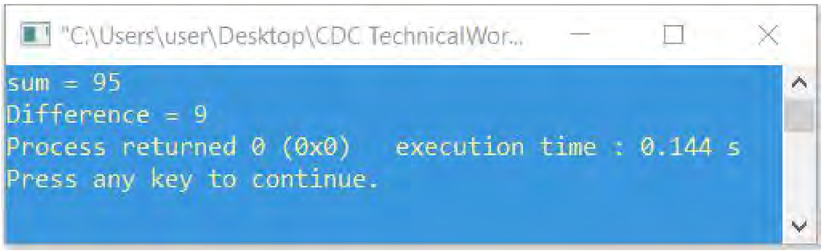
```
"C:\Users\user\Desktop\CDC TechnicalWor...
sum of 52 and 43 = 95
Process returned 21 (0x15)   execution time : 0.156 s

Press any key to continue.
```

Example 53



```
        #include<stdio.h>
        void sum (int a, int b);
// Declaration of function i.e. function prototype
        void diff (int a, int b);
        int main() {
        int x, y, total;
        x=52;
        y=43;
// sum function by the value method
        sum (x, y);
// calling diff funciton by a value method
        diff (x, y);
        return 0;
        }
        void sum (int a, int b) {
        printf ("sum = %d \n",a + b);
        }
        void diff (int a, int b) {
        printf ("Difference = %d", a - b);
        }
```

Output



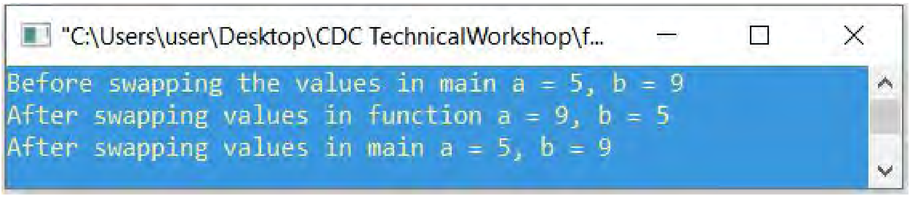
```
sum = 95
Difference = 9
Process returned 0 (0x0)   execution time : 0.144 s
Press any key to continue.
```

Example 54

```
        #include <stdio.h>
        void swap (int , int);                    //prototype of the function
```

```
int main() {
int a = 5, int b = 9;
printf ("Before swapping the values in main a = %d, b = %d \n", a, b);
swap (a, b);                        // calling a function by value method
printf ("After swapping values in main a = %d, b = %d\n", a, b);
}
void swap (int a, int b) {
int temp;
temp = a;
a = b;
b = temp;
printf ("After swapping values in function a = %d, b = %d\n", a, b);
}
```

Output



"C:\Users\user\Desktop\CDC TechnicalWorkshop\f...

```
Before swapping the values in main a = 5, b = 9
After swapping values in function a = 9, b = 5
After swapping values in main a = 5, b = 9
```

7.5.2 Call by Reference

In the call by reference method, the memory addresses of arguments are passed/ copied into the parameters (i.e., formal parameters). In this method, both the actual and the formal parameters use the same memory allocation. Any changes made inside the called function also changes the arguments in the calling function.

Example 55

// finding sum of any two integers by using call by reference method

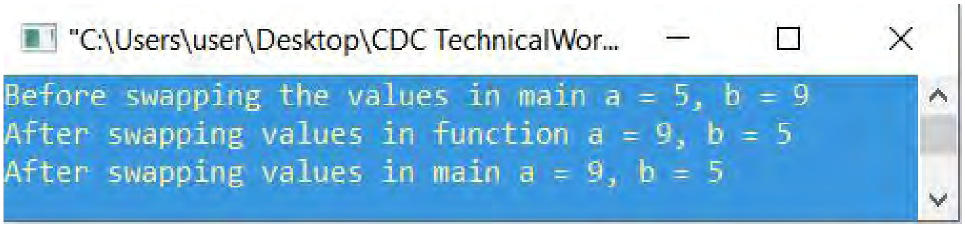
```
#include <stdio.h>
int sum(int *, int *);
int main() {
   int x, y, t;
   printf(" Enter the first integer : ");
```

```
    scanf("%d", &x);
    printf(" Enter the second integer : ");
    scanf("%d", &y);
    t = sum(&x, &y); // Calling the sum function using call by reference method
    printf (" The sum of %d and %d = %d \n", x, y, t);
    return 0;
}
int sum (int *a, int *b) {
    int total;
    total = *a + *b;
    return total;
}
```

Example 56

```
#include <stdio.h>
void swap(int *, int *);                //prototype of the function
int main() {
int a = 5;
int b = 9;
printf ("Before swapping the values in main a = %d, b = %d \n", a, b);
swap (&a, &b);                          // calling a function by reference method
printf ("After swapping values in main a = %d, b = %d \n", a, b);
}
void swap (int *a, int *b)  {
int temp;
temp = *a;
*a = *b;
*b = temp;
printf ("After swapping values in function a = %d, b = %d \n", *a, *b);
}
```

Output:



```
"C:\Users\user\Desktop\CDC TechnicalWor...
Before swapping the values in main a = 5, b = 9
After swapping values in function a = 9, b = 5
After swapping values in main a = 9, b = 5
```

Difference between call by value and call by reference methods

Call by value	Call by reference
• While calling a function, the values of variables or constants are passed to the parameters of the called function.	• While calling a function, the memory allocations of arguments are passed to the parameters of the called function.
• An arguments (i.e. actual parameter) and a parameter (i.e. formal parameter) use the different memory locations.	• Both an actual and a formal parameters use the same memory location.
• Any changes made on a formal parameter inside a function does change the value of actual parameter.	• Any changes made on a formal parameter inside a function changes the value of actual parameter.
• Pointer variables are not used.	• Pointer variables are used.

Exercise

Choose the correct answer from the given alternatives.

1. The special variable which holds the address of another variable is.......

   a. String Variable b. Pointer Variable

   c. Numeric variable d. Address variable

2. Which of the following symbol is used to declare a pointer variable?

   a. & b. * c. ! d. %

3. Which of the following is advantage of pointer?

   a. Pointer saves memory space.

   b. Pointer helps to access any memory location

   c. It increases execution time.

   d. All of the above

4. Which of the following is the correct syntax for declaring variable?

   a. Data_type variable name*

   b. Data_type &variablename

   c. Data_type *variablename

   d. Data_type variablename

5. Pointer is a used to store ....of variable.

   a. Address b. Data c. Memory content d. None

6. The dereferencing operator is ....

   a. * b. & c. % d. #

7. if int x=10 and int *y=&x; we have statement printf("%d",*y+5); what will be its output.

   a. 5 b.10 c. 15 d. 20

Write short answers to the following questions.

1. What is a pointer? Why it is used.
2. Describe the method of declaring the pointer and pointer variable with an example.
3. Define referencing and dereferencing with examples.
4. List the advantages of pointer

Write long answers to the following questions.

1. Describe the advantages of the pointer.
2. How is the pointer declared? Explain with an example.
3. What is the use of * and & operators in pointer.
4. Why pointers are necessary in c programming?

Project Work

Write C program to

1. WAP to enter a number and display its address using a pointer.
2. WAP to enter a number and display value using a dereferencing operator.

Reference

"Computer Fundamentals and Programming in C" by Reema Thareja B.S. Gottfried, Schaum's Outline Series for Programming with C, Second Edition

Kernighan and Ritchie "The C Programming Language", 2nd ed, 1988, (The "K&R". The Bible for the C language).

Plauger, "The Standard C Library", 1992.

Book References

"The C Programming Language" by Kernighan & Ritchie

"Programming in ANSI C" by E. Balagurusamy

"Introduction to Algorithms and Problem Solving" by Goyal & Aggarwal (Indian edition)

"C Programming: A Modern Approach" by K. N. King

"Computer Fundamentals and Programming in C" by Reema Thareja

B.S. Gottfried, Schaum's Outline Series for Programming with C, Second Edition

"Modern Operating System", 4th Edition by Tanenbaum

Web References

https://www.programiz.com/c-programming

https://www.w3schools.com/c/c_intro.php

https://www.programiz.com/c-programming/online-compiler/

https://en.wikipedia.org/wiki/C_(programming_language)

https://www.geeksforgeeks.org/c/c-programming-language/

https://www.tutorialspoint.com/cprogramming/index.htm

https://www.learn-c.org/