

**Department of Computer Science**

**Lab Manual**

**CMC 111**

# **Programming Fundamentals Lab**

Instructor’s Name: Anisa Fatima

Student’s Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Roll No.: \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Batch: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Semester: \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Year: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Department: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Department of Computer Science**

**Lab Manual**

**CMC 111**

**Programming Fundamentals Lab**

Prepared By:

Reviewed / Approved By:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Faculty of Engineering Sciences & Technology**

**Iqra University**

CONTENTS

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Lab. No.** | **Date** | **List of Experiment** | **Total Marks** | **Signature** | **Page #** |
| 1. |  | [**Introduction to Scratch Programming**](#_heading=h.k9dfalantvwv)  **Objective:** Introduce students to foundational programming concepts in Scratch, focusing on logic and structured thinking. |  |  |  |
| 2. |  | [**Scratch Programming Blocks & Functions**](#_heading=h.uhgw4yaq2dej)  [**Objective:**](#_heading=h.uhgw4yaq2dej) Introduce students to foundational programming concepts in Scratch, focusing on logic and structured thinking. |  |  |  |
| 3. |  | [**Introduction to C**](#_heading=h.7ulrikx9foyo)  Objective: To Learn about compiler installation, basic C program, basic data types, basic input or output and the use of variables. |  |  |  |
| 4. |  | [**Conditional Statements**](#_heading=h.o6x0l023e32q)  Objective: Arithmetic operators, conditional operators, assignment operators, Increment/decrement operators, Boolean Operators.  **Switch Case in C**  Objective:   * Understand the concept of the switch statement. * Learn how to use switch statements for decision-making in C.   Handle common beginner errors and understand syntax. |  |  |  |
| 5. |  | [**Loops**](#_heading=h.mwv8apqfii5c)  Objective: To understand and implement different loop constructs in C (while, do while, and for loops) and to identify suitable problems for each loop type. |  |  |  |
| 6. |  | [**Nested Loops**](#_heading=h.6qumdd4wnq1q)  Objective: To learn how to use nested loop constructs including:   * Nested while loop * Nested do-while loop * Nested for loop |  |  |  |
| 7. |  | *Open-ended Lab* |  |  |  |
|  |  | **Mid Term Examination** | 25 |  |  |
| 8 |  | [**Arrays in C**](#_heading=h.oc7tlxongj7y)  Objective: To understand and implement the concept of arrays in the C programming language. This lab will cover the declaration, initialization, and manipulation of arrays in C. And the implementation of algorithms using 1D and 2D Arrays. |  |  |  |
| 9 |  | [**Functions in C**](#_Functions)  To learn about functions in C language   * Why we use functions * How to define and use functions |  |  |  |
| 10 |  | [**Recursion**](#_heading=h.8ihcxz9cqxy9)**:**  **Objective:** The objective of this lab is to introduce the concept of recursion, specifically focusing on how the function call stack operates during recursive calls. Through this lab, students will gain an understanding of stack operations like rolling (push) and unrolling (pop), and explore potential issues like stack overflow in recursive functions. |  |  |  |
| 11 |  | [**Understanding Pointers in C Programming**](#_heading=h.izr39dtmn1hd)  **Objective:**   * Understand the concept of pointers and their use in C programming. * Learn how to declare, initialize, and use pointers to access and manipulate memory. * Explore how pointers enable "pass by reference" to allow functions to modify actual variables. * Study advanced applications of pointers, including pointer arithmetic and dynamic memory allocation. |  |  |  |
| 12 |  | [**File Handling in C Programming**](#_heading=h.30j0zll)  **Objective:**  To understand file handling concepts in C programming, including file creation, reading, writing, and appending using standard library functions. |  |  |  |
| 13 |  | [**Strings:**](#_heading=h.ehnsuj6qzi4e)   * Understand how a string constant is stored in a character array. * Learn the use of character arrays (strings) in C programming. * Understand and use the %s placeholder in printf() and scanf() functions for strings. * Apply loops for manipulating strings stored in character arrays. |  |  |  |
| 15 |  | Revision and Q/A Session |  |  |  |
| 16 |  | [**Open Ended Lab Assessment**](#_heading=h.da6py42aydcm) | 20 |  |  |
| **17** |  | **Final Examination** | **40** |  |  |

**Psychomotor Rubrics for Software based Lab**

***Course Name (Course Code): \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***Semester: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Criteria** | **Exceeds Expectations (>=90%)** | **Meets Expectations (70%-89%)** | **Developing (50%-69%)** | **Unsatisfactory (<50%)** |
| **Software Skills** | Ability to use software with its standard and  advanced features without assistance | Ability to use software with its standard and  advanced features with minimal assistance | Ability to use software with its  standard features with assistance | Unable to use the software |
| **Programming/ Simulation** | Ability to program/ simulate the lab tasks  with simplification | Ability to program/ simulate the lab tasks  without errors | Ability to program/ simulate lab tasks  with errors | Unable to program/simulate |
| **Results** | Ability to achieve all the desired results with alternate ways | Ability to achieve all the desired results | Ability to achieve most of the desired results with errors | Unable to achieve the desired results |
| **Laboratory Manual** | All sections of the report are very well written and technically accurate. | All sections of the report are technically accurate. | Few sections of the report contain technical errors. | All sections of the report contain multiple technical errors. |

**Psychomotor Rubrics Assessment Software based Lab**

**Course Name (Course Code): \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Semester: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Lab # | Score Allocation | | | | | |
| **Software Skills**  **Marks (3)** | **Programming/ Simulation**  **Marks (3)** | **Experimental Results**  **Marks (2)** | | **Laboratory Manual**  **Marks (2)** | **Total Marks (10)** |
| 1 |  |  |  | |  |  |
| 2 |  |  |  | |  |  |
| 3 |  |  |  | |  |  |
| 4 |  |  |  | |  |  |
| 5 |  |  |  | |  |  |
| 6 |  |  |  | |  |  |
| 7 |  |  |  | |  |  |
| 8 |  |  |  | |  |  |
| 9 |  |  |  | |  |  |
| 10 |  |  |  | |  |  |
| 11 |  |  |  | |  |  |
| 12 |  |  |  | |  |  |
| 13 |  |  |  | |  |  |
| 14 |  |  |  | |  |  |
| Total Marks | | 140 | | Total Obtained marks | |  |

***Overall Score: \_\_\_\_\_\_\_\_\_\_\_out of 15 Examined by****: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

***(Obtained Score / 140) x 15 (Name and Signature of lab instructor)***

**Affective Domain Rubrics Assessment**

**Course Name (Course Code): \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Semester: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CATEGORY** | **Excellent (100% - 85%)** | **Good (84% - 75%)** | **Fair (74% - 60%)** | **Poor (Less than 60%)** |
| **Speaks Clearly** | Speaks clearly and distinctly all the time, and confidently. | Speaks clearly and distinctly most of the time, but is confused for a brief period of time, however, recovers. | Speaks clearly and distinctly most of the time, but seems not confident about what has been delivered. Shows lack of confidence. | Often mumbles or cannot be understood and clearly lacks confidence in delivering the content |
| **Points:** |  |  |  |  |
| **Preparedness** | Student is completely prepared and has obviously rehearsed. | Student seems pretty prepared but might have needed a couple more rehearsals. | The student is somewhat prepared, but it is clear that rehearsal was lacking. | Student does not seem at all prepared to present. |
| **Points** |  |  |  |  |
| **Answer back** | Student calmly listens to the questions and responds to the question confidently and correctly | Student calmly listens to the questions, responds confidently but some of the responses are incorrect. | Student shows anxiety while listening to the questions, and gives some correct responses, but some of the responses are incorrect. | Student shows anxiety while listening to the questions, and most of the responses are incorrect. |
| **Points:** |  |  |  |  |
| **Posture, Eye Contact & Speaking Volume** | Stands up straight, looks relaxed and confident. Establishes eye contact with everyone in the room during the presentation. Volume is loud enough to be heard by all members in the audience throughout the presentation. | Stands up straight and establishes eye contact with everyone in the room during the presentation. Volume is loud enough to be heard by the audience, but is sometimes not audible. | Sometimes stands up straight and establishes eye contact. Volume is loud enough to be heard by the audience, but many sentences spoken are not clear. | Lazy and informal posture. Does not look at people during the presentation. Volume is also too soft to be heard by the audience. |
| **Points:** |  |  |  |  |

***Overall Score: \_\_\_\_\_\_\_\_\_\_\_out of 15 Examined by****: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

***(Name and Signature of lab instructor)***

***Open Ended Lab Assessment Rubrics***

***Course Name (Course Code): \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***Semester: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

|  |  |  |  |
| --- | --- | --- | --- |
| **Criteria and Scales** | | | |
| **Excellent**  **(20-18)** | **Good**  **(14-17)** | **Average**  **(10-13)** | **Poor**  **(1-8)** |
| **Criterion 1:** **Understanding the Problem:** How well the problem statement is understood by the student | | | |
| Understands the problem clearly and clearly identifies the underlying issues. | Adequately understands the problem and identifies the underlying issues. | Inadequately defines the problem and identifies the underlying issues. | Fails to define the problem adequately and does not identify the underlying issues. |
| **Criterion 2: Research:** The amount of research that is used in solving the problem | | | |
| Contains all the information needed for solving the problem | Good research, leading to a successful solution | Mediocre research which may or may not lead to an adequate solution | No apparent research |
| **Criterion 3: Class Diagram:** The completeness of the class diagram | | | |
| Class diagram with complete notations | Class diagram with incomplete notations | Class diagram with improper naming convention and notations | No Class diagram |
| **Criterion 4: Code:** How complete and accurate the code is along with the assumptions | | | |
| Complete Code according to the class diagram of the given case with clear assumptions | Incomplete Code according to the class diagram of the given case with clear assumptions | Incomplete Code according to the class diagram of the given case with unclear assumptions | Wrong code and naming conventions |
| **Criterion 5: Report:** How thorough and well organized is the solution | | | |
| All the necessary information clearly organized for easy use in solving the problem | Good information organized well that could lead to a good solution | Mediocre information which may or may not lead to a solution | No report provided |

***Open Ended Lab Assessment Rubrics***

***Course Name (Course Code): \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***Semester: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Criteria and Scales** | | | | |
| **Excellent**  **(20-18)** | **Good**  **(14-17)** | **Average**  **(10-13)** | **Poor**  **(1-8)** | **Total Marks**  **20** |
| **Criterion 1:**  **Understanding the Problem:** How well the problem statement is understood by the student | | | | |
| **(20-18)** | **(14-17)** | **(10-13)** | **(1-8)** |  |
| **Criterion 2:**  **Research**: The amount of research that is used in solving the problem | | | | |
| **(20-18)** | **(14-17)** | **(10-13)** | **(1-8)** |  |
| **Criterion 3:**  **Class Diagram:** The completeness of the class diagram | | | | |
| **(20-18)** | **(14-17)** | **(10-13)** | **(1-8)** |  |
| **Criterion 4:**  **Code:** How complete and accurate the code is along with the assumptions | | | | |
| **(20-18)** | **(14-17)** | **(10-13)** | **(1-8)** |  |
| **Criterion 5:**  **Report**: How thorough and well organized is the solution | | | | |
| **(20-18)** | **(14-17)** | **(10-13)** | **(1-8)** |  |
| Total | | | | **(\_\_\_\_/20)** |

Total marks obtained: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Name and Signature of lab instructor: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Rubrics for Lab Project / CCA**

***Course Name (Course Code): \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***Semester****: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Criteria** | **Exceeds Expectations**  **(>=90%)** | **Meets Expectations (70%-89%)** | **Developing (50%-69%)** | **Unsatisfactory (<50%)** |
| **Project Presentation + Project Demonstration** | Ability to demonstrate the project with achievement of required objectives having clear understanding of project limitations and future enhancements. Hardware and/or Software modules are fully functional, if applicable. | Ability to demonstrate the project with achievement of required objectives but understanding of project limitations and future enhancements is insufficient.  Hardware and/or Software modules are functional, if applicable. | Ability to demonstrate the project with achievement of a\*t least 50% required objectives and insufficient understanding of project limitations and future enhancements.  Hardware and/or Software modules are partially functional, if applicable. | Ability to demonstrate the project with achievement of less than 50% required objectives and lacks in understanding of project limitations and future enhancements.  Hardware and/or Software modules are not functional, if applicable. |
| **Project Report** | All sections of the Project report are very well- written and technically accurate. | All sections of the Project report are technically accurate. | Few sections of the Project report contain technical errors. | Project report has several grammatical/ spelling errors and sentence construction is poor. |
| **Viva** | Able to answer the questions easily and  correctly across the project. | Able to answer the questions related to the project | Able to answer the questions but with mistakes | Unable to answer the questions |

Total marks: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Name and Signature of lab instructor: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Project Rubric based Assessment**

**Course Name (Course Code): \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Semester: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Project # | Score Allocation | | | |
| **Project Presentation + Project Demonstration**  **Marks (10)** | **Project Report**  **Marks (5)** | **Viva**  **Marks (5)** | **Total**  **Marks (20)** |
| 1 |  |  |  |  |
| 2 |  |  |  |  |
| Total Obtained Score | | | |  |

Total marks obtained: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Name and Signature of lab instructor: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Final Lab Assessment**

|  |  |  |  |
| --- | --- | --- | --- |
| **Assessment Tool** | **CLO-1** | **CLO-2** | **CLO-3** |
| **Lab Manual** |  |  |  |
| **Subject Project / Viva** |  |  |  |
| **Lab Exam / Viva** |  |  |  |
| **Score Obtained** |  |  |  |
| **Total Score: \_\_\_\_\_\_\_\_ out of 40** | | | |

***Examined by****:*

***(Name and Signature of concerned lab instructor)***

# General Laboratory Procedure

While there is no specific document to be submitted at the beginning of the Lab –unless your instructor advises you otherwise-, you are expected to read the experiment fully before you come to the laboratory? Interestingly, you can even try parts of the experiment at home. Here is a list of programs that will equip you with a virtual lab at your home:

## Troubleshooting

Things will not always go as expected; this is the nature of the learning process. While conducting the Experiment **think before you do anything.** If you do so you will avoid wasting time going down dead-end streets. Be logical and systematic. First, look for obvious errors that are easy to fix. Is your measuring device correctly set and connected? Are you looking at the proper scale? Is the power supply set for the correct voltage? Is the signal generator correctly set and connected? How are the variables in the code set? Is there a syntax error? And so on. Next, check for obvious misconnections or broken connections, at least in simple circuits.

As you work through your circuit, use your Lab Manual record tests and changes that you make as you go along; don't rely on your memory for what you have tried. Identify some test points in the system at which you know what the signal should be and work your way backwards from the output through the test points until you find a good signal.

1. **Neatness**

When you have finished for the day, return all modules to their proper storage bins, return all test leads and probes to their storage racks, return all equipment to its correct location, and clean up the lab station. If appropriate switch off the unneeded equipment. Save your files in the computer and on any USB device for your records because you might not get the same PC System again for the next experiment. Also email your file contents to your email address as a backup.

Laboratory Safety



Always pay attention to what you are doing and you’re surrounding during the experiments, notify the Instructor for any unlikely event or mishap, and leave the Laboratory with the permission of Instructor immediately. All students must read and understand the information in this document with regard to laboratory safety and emergency procedures prior to the first laboratory session. Your personal laboratory safety depends mostly on YOU. Efforts have been made to address situations that may pose a hazard in the lab but the information and instructions provided cannot be considered all inclusive. Students must adhere to written and verbal safety instructions throughout the academic term. Since additional instructions may be given at the beginning of laboratory sessions, it is important that all students arrive at each session on time. With good judgment, the chance of an accident in this course is very small. Nevertheless, research and teaching workplaces (labs, shops, etc.) are full of potential hazards that can cause serious injury and or damage to the equipment. Working alone and unsupervised in laboratories is forbidden if you are working with hazardous substances or equipment. With prior approval, at least two people should be present so that one can shut down equipment and call for help in the event of an emergency. Safety training and/or information should be provided by a faculty member, teaching assistant, lab safety contact, or staff member at the beginning of a new assignment or when a new hazard is introduced into the workplace.

1. **Emergency Response**

* It is your responsibility to read safety and fire alarm posters and follow the instructions during an emergency
* Know the location of the fire extinguisher, eye wash, and safety shower in your lab and know how to use them.
* Notify your instructor immediately after any injury, fire or explosion, or spill.
* Know the building evacuation procedures.

1. **Common Sense**

Common Sense Good common sense is needed for safety in a laboratory. It is expected that each student will work in a responsible manner and exercise good judgment and common sense. If at any time you are not sure how to handle a particular situation, ask your Teaching Assistant or Instructor for advice DO NOT TOUCH ANYTHING WITH WHICH YOU ARE NOT COMPLETELY FAMILIAR!!! It is always better to ask questions than to risk harm to yourself or damage to the equipment.

1. **Personal and General laboratory safety** A no food sign with a bottle and a burger

   Description automatically generated

* Never eat, drink, or smoke while working in the laboratory.
* Read labels carefully.
* Do not use any equipment unless you are trained and approved
* as a user by your supervisor.
* Wear safety glasses or face shields when working with hazardous materials and/or equipment.
* Wear gloves when using any hazardous or toxic agent.
* Clothing: When handling dangerous substances, wear gloves, laboratory coats, and safety shield or glasses. Shorts and sandals should not be worn in the lab at any time. Shoes are required when workingin the machine shops.
* If you have long hair or loose clothes, make sure it is tied back or confined.
* Keep the work area clear of all materials except those needed for your work. Coats should be hung in the hall or placed in a locker. Extra books, purses, etc. should be kept away from equipment that requires air flow or ventilation to prevent overheating.
* Disposal - Students are responsible for the proper disposal of used material if any in appropriate containers.
* Equipment Failure - If a piece of equipment fails while being used, report it immediately to your lab assistant or tutor. Never try to fix the problem yourself because you could harm yourself and others.
* If leaving a lab unattended, turn off all ignition sources and lock the doors.
* Never pipette anything by mouth.
* Clean up your work area before leaving.
* Wash hands before leaving the lab and before eating.
* Unauthorized person(s) shall not be allowed in a laboratory for any reason

1. **Electrical safety**

* Obtain permission before operating any high voltage equipment.
* Maintain an unobstructed access to all electrical panels.
* Wiring or other electrical modifications must be referred to the Electronics Shop or the Building
* Coordinator.
* Avoid using extension cords whenever possible. If you must use one, obtain a heavy- duty one that is electrically grounded, with its own fuse, and install it safely. Extension cords should not go under
* doors, across aisles, be hung from the ceiling, or plugged into other extension cords.
* Never, ever modify, attach or otherwise change any high voltage equipment.
* Always make sure all capacitors are discharged (using a grounded cable with an insulating handle) before touching high voltage leads or the "inside" of any equipment even after it has been turned off. Capacitors can hold charge for many hours after the equipment has been turned off.
* When you are adjusting any high voltage equipment or a laser which is powered with a high voltage supply, USE ONLY ONE HAND. Your other hand is best placed in a pocket or behind your back. This procedure eliminates the possibility of an accident where high voltage current flows up one arm, through your chest, and down the other arm.
* Discard damaged cords, cords that become hot, or cords with exposed wiring.
* Before equipment is energized ensure, (1) circuit connections and layout have been checked by a Teaching Assistant (TA) and (2) all colleagues in your group give their assent.
* Know the correct handling, storage and disposal procedures for batteries, cells, capacitors, inductors and other high energy-storage devices.
* Experiments left unattended should be isolated from the power supplies. If for a special reason, it must be left on, a barrier and a warning notice are required.
* Equipment found to be faulty in any way should be reported to the Lab Engineer immediately and taken out of service until inspected and declared safe.
* Voltages above 50 V rms AC and 120 V DC are always dangerous. Extra precautions should be considered as voltage levels are increased.
* Never make any changes to circuits or mechanical layout without first isolating the circuit by switching off and removing connections to power supplies.
* Know what you must do in an emergency.
* Emergency Power Off: Every lab is equipped with an Emergency Power Off System.
* Only authorized personnel are permitted to reset power once the Emergency Power Off system has been engaged.

1. **Electrical Emergency Response**

The following instructions provide guidelines for handling two types of electrical emergencies:

* When someone suffers serious electrical shock, he or she may be knocked unconscious. If the victim is still in contact with the electrical current, immediately turn off the electrical power source. If you cannot disconnect the power source, depress the Emergency Power Off switch.
* Do not touch a victim that is still in contact with a live power source; you could be electrocuted.
* Have someone call for emergency medical assistance immediately. Administer first-aid, as appropriate.
* If an electrical fire occurs, try to disconnect the electrical power source, if possible. If the fire is small and you are not in immediate danger; and you have been properly trained in fighting fires, use the correct type of fire extinguisher to extinguish the fire. When in doubt, push in the Emergency Power Off button.
* NEVER use water to extinguish an electrical fire.

1. **Mechanical safety**

* When using compressed air, use only approved nozzles and never direct the air towards any person.
* Guards on machinery must be in place during operation.
* Exercise care when working with or near hydraulically- or pneumatically driven equipment. Sudden or unexpected motion can inflict serious injury.

1. **Additional Safety Guidelines**

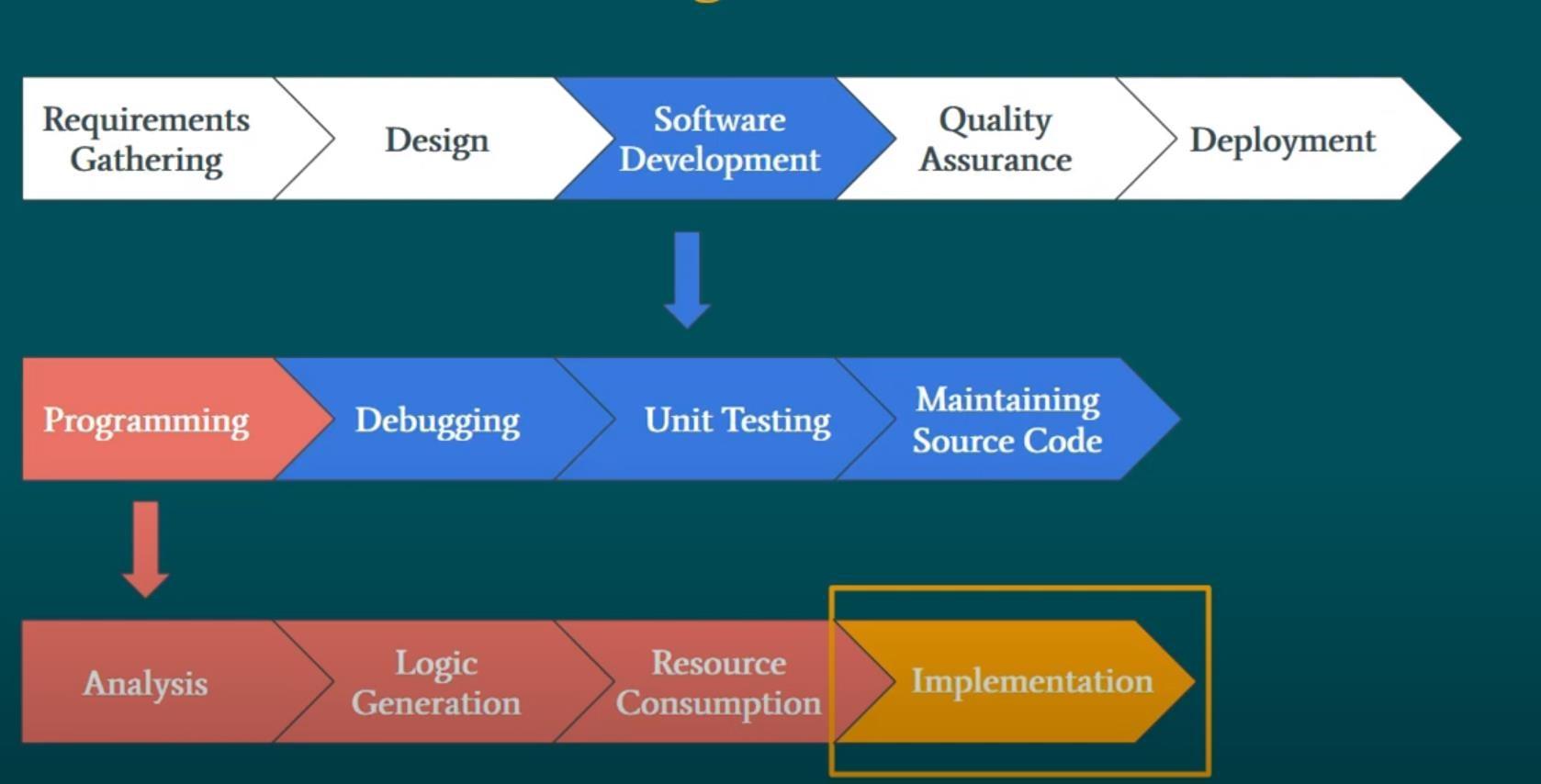
* Never work alone in a laboratory.
* Keep your lab space clean and organized.
* Do not leave an on-going experiment unattended.
* Always inform your instructor if you break a thermometer. Do not clean mercury yourself!!
* Never taste anything. Never pipette by mouth; use a bulb.
* Never use open flames in the laboratory unless instructed by TA.
* check your glassware for cracks and chips each time you use it. Cracks could cause the glassware to fail during use and cause serious injury to you or lab mates.
* Maintain unobstructed access to all exits, fire extinguishers, electrical panels, emergency showers, and eye washes.
* Do not use corridors for storage or work areas.
* Do not store heavy items above table height. Any overhead storage of supplies on top of cabinets should be limited to lightweight items only. Also, remember that a 36" diameter area around all fire sprinkler heads must be kept clear at all times.
* Areas containing lasers, biohazards, radioisotopes, and carcinogens should be posted accordingly.
* However, do not post areas unnecessarily and be sure that the labels are removed when the hazards are no longer present.
* Be careful when lifting heavy objects. Only shop staff may operate forklifts or cranes.
* Clean your lab bench and equipment and lock the door before you leave the laboratory.

1. **Clothing**

* Dress properly during a laboratory activity.
* Long hair, dangling jewelry, and loose or baggy clothing are a hazard in the laboratory.
* Long hair must be tied back, and dangling jewelry and baggy clothing must be secured.
* Shoes must completely cover the foot.
* No sandals allowed on lab days.
* A lab coat or smock should be worn during laboratory experiments. Accidents and Injuries 1. Do not panic.
* Report any accident (spill, breakage, etc.) or injury (cut, burn, etc.) to the teacher immediately, no matter how trivial it seems.
* If you or your lab partner is hurt, immediately (and loudly) yell out the teacher's name to get the teacher's attention.

### **General Warning Signs**





**Lab 01**

# **Introduction to Scratch Programming**

## Objective:

Introduce students to foundational programming concepts in Scratch, focusing on logic and structured thinking.

## Required Equipment / tools:

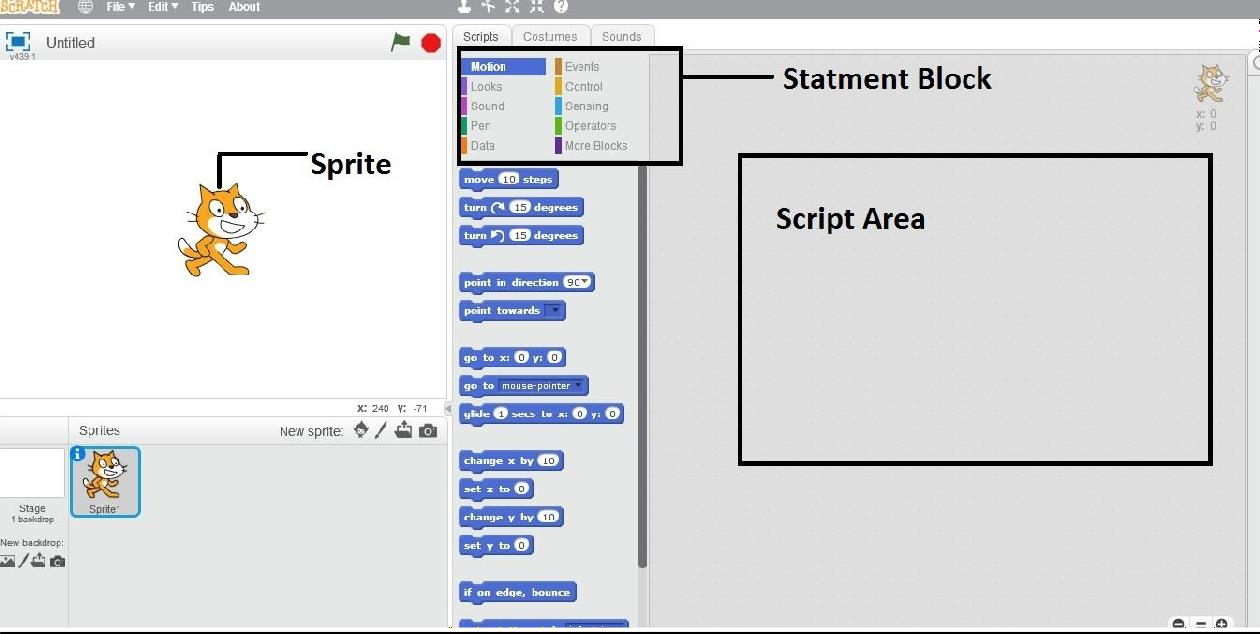
* Scratch Account

## Introduction:

**Scratch Programming**

Scratch is a block-like programming language to create visual programs and also write code for other innovative problems easily using visual blocks of codes. With Scratch, you can program your own interactive stories, games, and animations. Using Scratch, users can create online projects and develop them into almost anything by using a simple block-like interface.

**Scratch Environment**



**Sprite**

Is an actor in scratch programming which shows the result/output for the written block of code in the script area.

**Script Area**

The area in which the statement block dragged from script block and dropped to the script area.

## Procedure

### Step 1: Open Scratch

1. **Go to the Website**:  
   Open a browser and visit <https://scratch.mit.edu/>.  
   Alternatively, download and install the Scratch offline editor from the website.
2. **Log In (Optional)**:
   * Create a free Scratch account or log in to an existing one.
   * Logging in allows you to save and share your projects.
3. **Start a New Project**:
   * Click on the **“Create”** button at the top of the page to launch the Scratch editor.

### Step 2: Get Familiar with the Interface

1. **The Editor Layout**:
   * **Stage Area**: Displays your project and sprites in action.
   * **Sprite List**: Shows all the characters or objects in your project.
   * **Blocks Palette**: Contains pre-built code blocks grouped by categories (e.g., Motion, Looks, Sound).
   * **Script Area**: The workspace where you drag and drop code blocks.
2. **Explore Tabs**:
   * **Code Tab**: Use blocks to program your sprites.
   * **Costumes Tab**: Design or edit the appearance of your sprites.
   * **Sounds Tab**: Add or record sound effects for your project.

### **Step 3: Add a Sprite**

1. **Default Sprite**: Scratch starts with a cat sprite. You can keep it or delete it by clicking the trash icon.
2. **Add New Sprites**:
   * Click the **“Choose a Sprite”** button (at the bottom of the Sprite List).
   * Options include:
     + Selecting from the library.

### **Step 4: Program Your Sprite**

1. **Select Your Sprite**:
   * Click on the sprite in the Sprite List.
2. **Drag and Drop Code Blocks**:
   * From the Blocks Palette, drag blocks to the Script Area to create a script.
   * Connect blocks like puzzle pieces to build the desired behavior.

Statements Block or Scripts

## Observation

|  |
| --- |
|  |

**Examples**

**Example 01**



To move a sprite 10 step in forward direction

**Example 02**



To move a sprite 10 step in forward direction with **when click condition When Clicked** is an event to execute the script by clicking on it.

**Example 03**



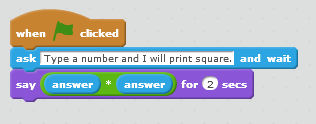
Join the puzzle pieces

**Example 04**



**Ask user name from user**

**Example 05**

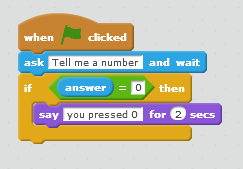


**Usage of Operators**

**Decision making Block**

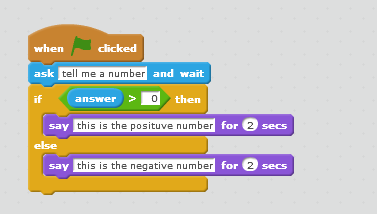
Decides whether certain criteria (condition) is true or not. If true it executes if block otherwise it executes else block.

**Example 06**



**Decision making Statement-checks whether you typed zero-(without else block)**

**Example 07**

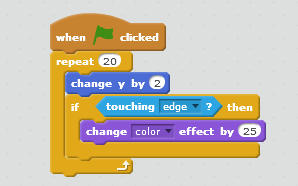


**Decision making Statement -checks whether you typed a positive number or a negative number (with else block)**

**Repeat Block**

Repeats itself until the number of times given.

**Example 08**



Repeat Block

## Lab Tasks

**Task01 (hobby.sb)**

Create a script to ask the user for his hobby and spirit will show it for 5 seconds.

**Task02 (cube.sb)**

Create a script to take a number from the user and print its cube.

**Task03 (SpellCheck.sb)**

Create script to ask name from user and check if its Pakistan then sprite said Pakistan for 5 sec using if block.

## Home Tasks

**Problem01 (signup.sb)**

Take a username, password and age as input and create an account of the user if he/she is above 18 years.

**Problem02 (login.sb)**

Take a username and password as input and validate if the password is equal to 12345 then login to the system.

**Problem03 (change.sb)**

Take a number as input if the number is less than 0 then change the color of the sprite else change the position of the sprite by 2 points by y.

**Lab Task Submission Instructions**

1. Screenshot your code and output and paste it into a Word file.
2. Copy paste your code into that word file according to the Question number.
3. Number your Word document as question number e.g.Q1.sb, Q2.sb, etc. (Q is in upper case)
4. Create a new folder with your Name and ID e.g. Naveed Ahmed (61452)\_Lab 01 Task.
5. Copy all the Word document and source files into this folder.
6. Right-click on the folder you created and create a zip file by selecting the option
   * “Send to” and selecting “Compressed (zipped) folder” [for windows].
   * “Create Archive” and change the option to “.zip” instead of “ .tar.gz” and click on “Create”. [for linux]
7. Now make sure a zip file is created with your name and id e.g. Naveed Ahmed

(61452).zip.

1. Submit Lab Task on LMS.

*Note:*

*Plagiarism is strictly prohibited. Any instance of plagiarism will result in a deduction of 100 points. Ensure that all work submitted is original.*

## Discussion and Analysis of Results

|  |
| --- |
|  |

## Conclusion

|  |
| --- |
|  |

**Lab 02**

# **Scratch Programming Blocks & Functions**

## Objective:

Introduce students to foundational programming concepts in Scratch, focusing on logic and structured thinking.

## Required Equipment / tools:

* Scratch Account

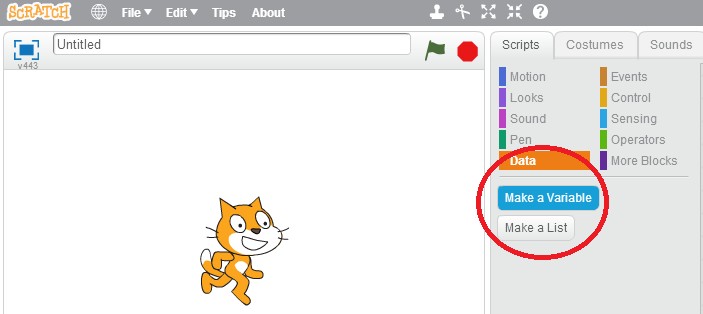
## Introduction

**Scratch Programming**

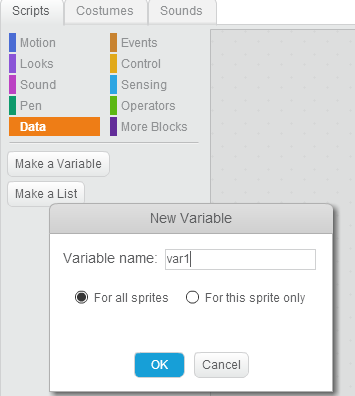
Scratch is a block-like programming language to create visual programs and also write code for other innovative problems easily using visual blocks of codes. With Scratch, you can program your own interactive stories, games, and animations

**Variables in Scratch:** Variable is like a container in a memory that holds data temporary for the computer program.

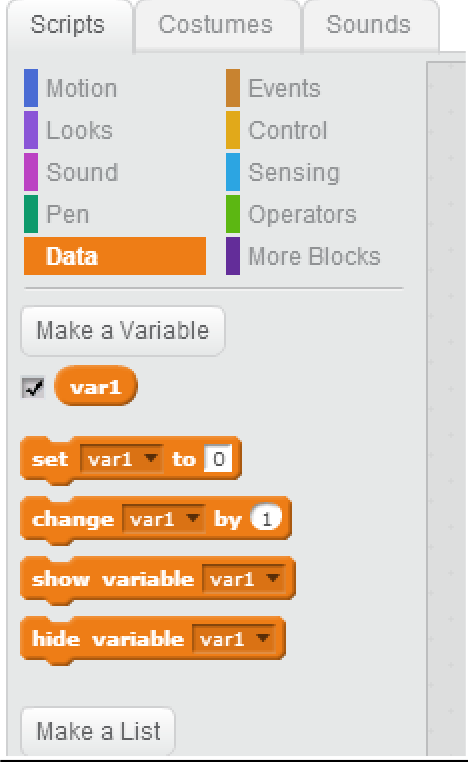
## Procedure



**Making a variable**



**Naming the variable**



**Variable ‘var1’ made**

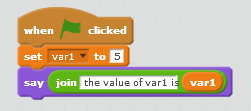
## Observation

|  |
| --- |
|  |

**Examples**

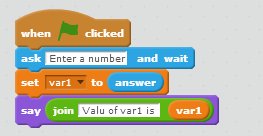
**Example 01**

**Task01 (DisplayName.sb):**



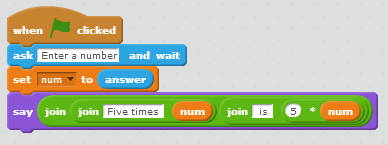
**Setting value of a variable**

**Example 02**



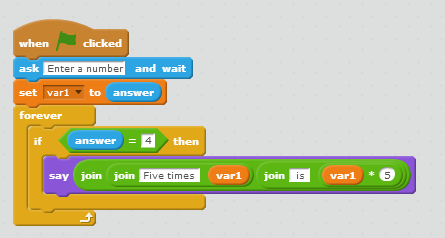
**Setting “user entered value” to a variable**

**Example 03**



**Performing operations on user entered variable value**

**Example 04**



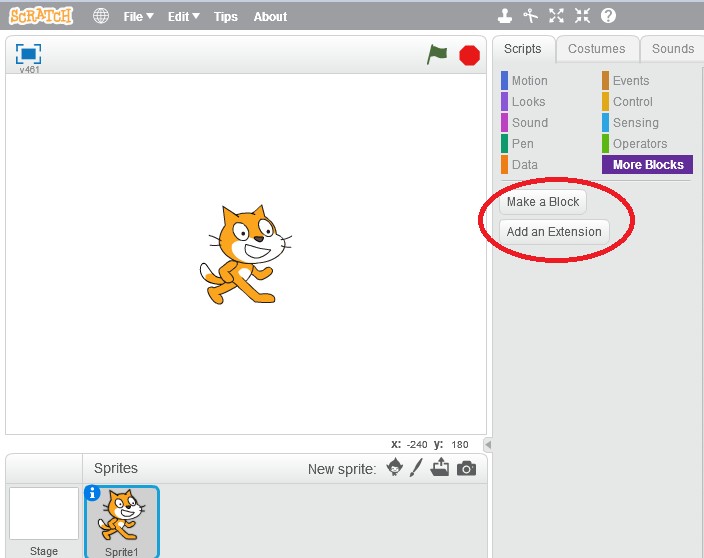
**Forever loop with if condition**

**Example 05**

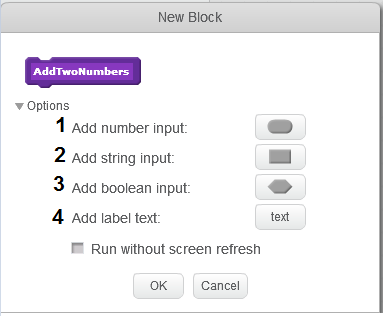


**Picking random Number**

**Functions/Blocks in Scratch:** Block/Function in scratch is a group of puzzles to perform an activity or task.



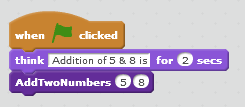
**Creating functions/block in scratch**



**Write name of the function in purple block**

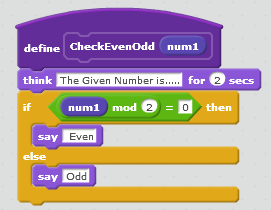
**Select 1: to pass number as an argument to function Select 2: to pass string as an argument to function Select 3: to pass Boolean as an argument to function Select 4: to add label to the function**

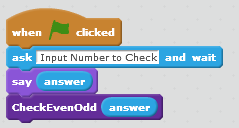
**Example 06**



**Function (AddTwoNumbers) Definition**

**Function Calling**

**Example 07**



**Check Given Number is even or using functions Function Definition**

**Function Calling**

## Lab Tasks:

**Task04 (maximum.sb):**

Input two numbers from the user and find the larger of the two. (Ignore equal case i.e. assume numbers are distinct)

**Task 05 (Guessgame.sb):**

Create a script to take a number entered from the user and match with a random generated number (1-5), if your guess number is matched you will win, otherwise you lose the game.

**Task 06 (product.sb):**

Create a script to multiple two numbers passed by the script to the function and function will return the result where function is invoked.

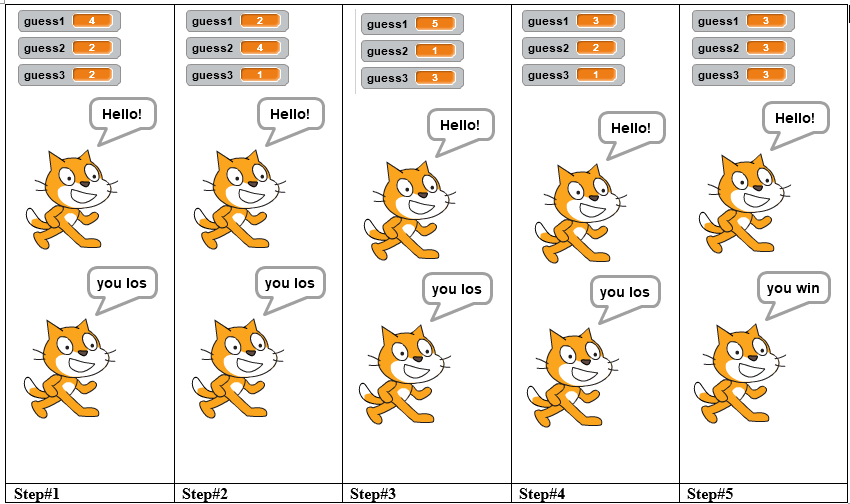
## Home Tasks

**Problem 01 (Guessing Game.sb)**

Create script to generate three numbers randomly and set to the variables (guess1, guess2

guess3) respectively after that play a game if the all three generated number are equal then you win

the game otherwise loses.



**Problem02 (circle.sb)**

**Create a script to draw circle shape with pen size 5 points**



**Problem03 (voteCasting.sb)**

Create script to take age of the person as input and determine whether it is eligible for casting his/her own vote.

**Note: (the person must be 18 years or above to be eligible f**

**Lab Task Submission Instructions**

1. Screenshot your code and output and paste it into a Word file.
2. Copy paste your code into that word file according to the Question number.
3. Number your Word document as question number e.g.Q1.sb, Q2.sb, etc. (Q is in upper case)
4. Create a new folder with your Name and ID e.g. Naveed Ahmed (61452)\_Lab 02 Task.
5. Copy all the Word document and source files into this folder.
6. Right-click on the folder you created and create a zip file by selecting the option
   * “Send to” and selecting “Compressed (zipped) folder” [for windows].
   * “Create Archive” and change the option to “.zip” instead of “ .tar.gz” and click on “Create”. [for linux]
7. Now make sure a zip file is created with your name and id e.g. Naveed Ahmed

(61452).zip.

1. Submit Lab Task on LMS.

*Note:*

*Plagiarism is strictly prohibited. Any instance of plagiarism will result in a deduction of 100 points. Ensure that all work submitted is original.*

## Discussion and Analysis of Results

|  |
| --- |
|  |

## Conclusion

|  |
| --- |
|  |

**Lab 03**

# **Introduction to C**

## Objective:

To Learn about compiler installation, basic C program, basic data types, basic input or output and the use of variables.

## Required Equipment / tools:

1. Dev C++

## Introduction:

Programming

A programming language is a formal language designed to communicate instructions to a machine, particularly a computer. Programming languages can be used to create programs that control the behavior of a machine and/or to express algorithms precisely.

1.1 Compiler

A compiler is a computer program (or set of programs) that transforms source code written in a programming language (the source language) into another computer language (the target language, often having a binary form known as object code). The most common reason for wanting to transform source code is to create an executable program.

1.2 IDE

An Integrated Development Environment (IDE) is a software that provides a comprehensive consolidated environment to computer programmers to develop software applications. An IDE consists of various components such as a code editor, build automation tools, debuggers, and sometimes visual designers. We will use the Dev-C++ IDE to develop C programs. Dev-C++ is a free, open-source, lightweight IDE for C and C++ programming that uses the MinGW (Minimalist GNU for Windows) port of the GCC compiler, supporting efficient program development on the Windows platform.

1.2.1 Basic Components of an IDE

* **Code Editor (CE):**

A code editor is a specialized text editor designed for writing and editing the source code of computer programs. It is an essential part of an IDE, providing features like syntax highlighting and code formatting to make programming easier and more efficient. In Dev-C++, the built-in code editor supports both C and C++ languages, allowing you to write, edit, and organize your code within a single interface.

* **Build Automation Tools (BAT):**

Build automation tools streamline the process of creating software builds. This involves compiling source code into object code, linking libraries, and generating the final executable file. In Dev-C++, the build automation process is handled using the MinGW compiler, which compiles and links your code into a runnable program. These tools simplify the process, allowing for easy building and testing of software without manual intervention.

* **Debugger:**

The debugger is a critical component of an IDE that assists in testing and identifying bugs in software. Dev-C++ integrates with the GDB (GNU Debugger) to help developers test their code, set breakpoints, inspect variables, and track down issues in complex applications. This makes debugging an efficient process, allowing you to find and fix errors directly within the IDE.

**1.2.2 Few Features of Dev-C++**

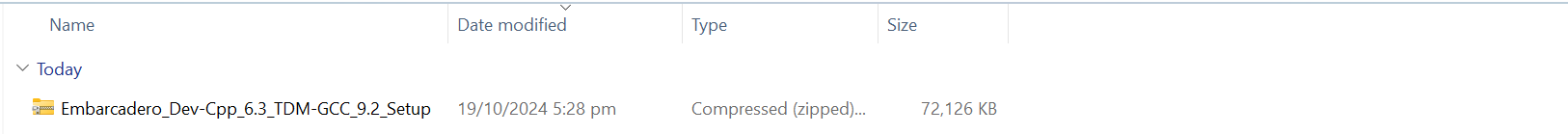
* Syntax highlighting
* Code completion
* Debugging
* Code searching
* Project templates
* Single-platform compilation

## Procedure

1.3 Install Compiler

Step#1: <https://www.embarcadero.com/free-tools/dev-cpp/free-download>

Step#2: After downloading Extract the folder



*Figure 1: Zipped Setup of Dev C++*

Step#3: Now run the Embarcadero\_Dev-Cpp\_6.3\_TDM-GCC\_9.2\_Setup

Step#4: After running the setup, follow the on-screen instructions:

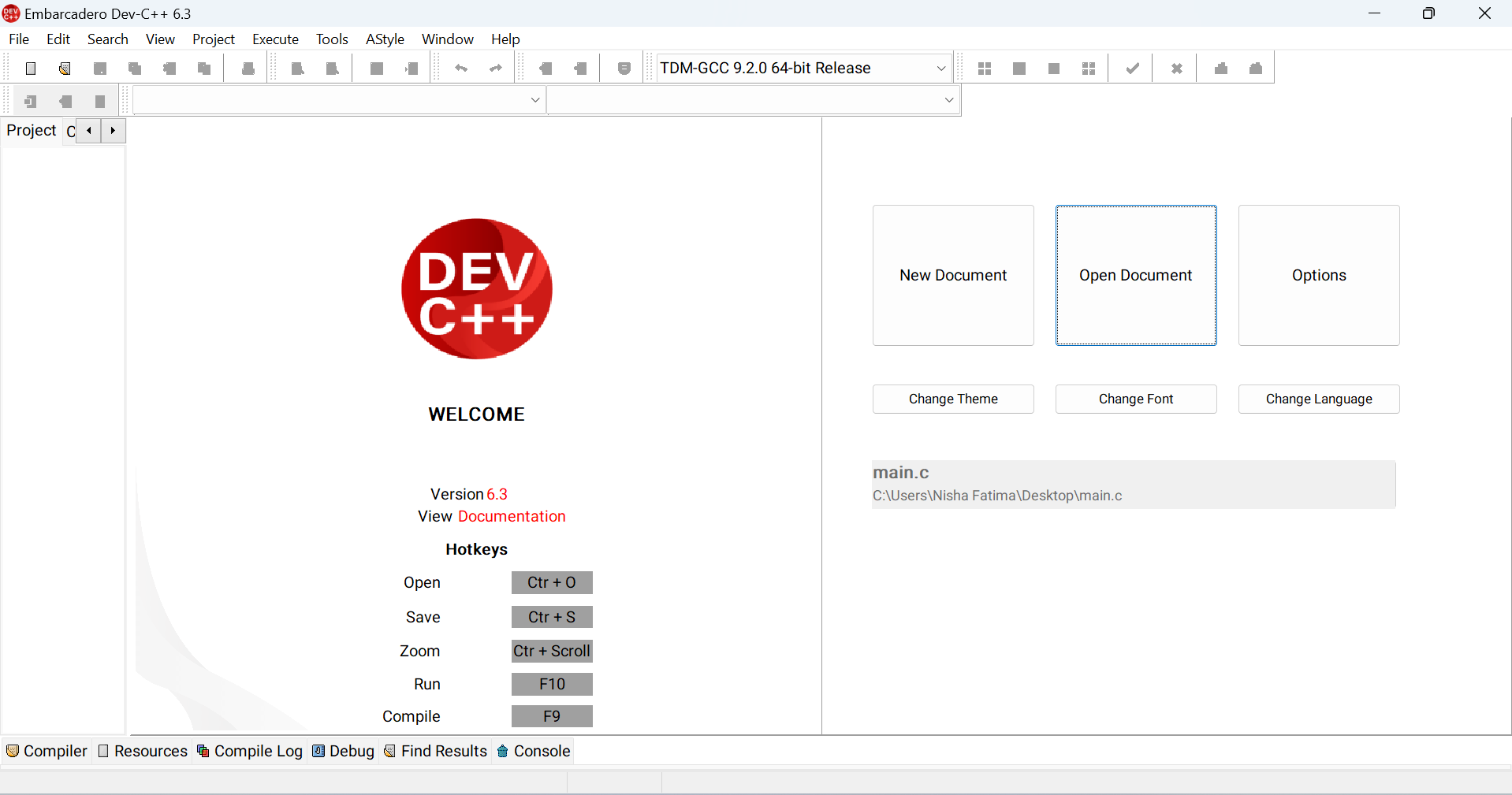
* Choose the language (English or any other preferred language).
* Click Next and accept the License Agreement.
* Choose the installation path or leave it as default.
* Click Install and wait for the installation to complete.

Step#5: Creating a New File:

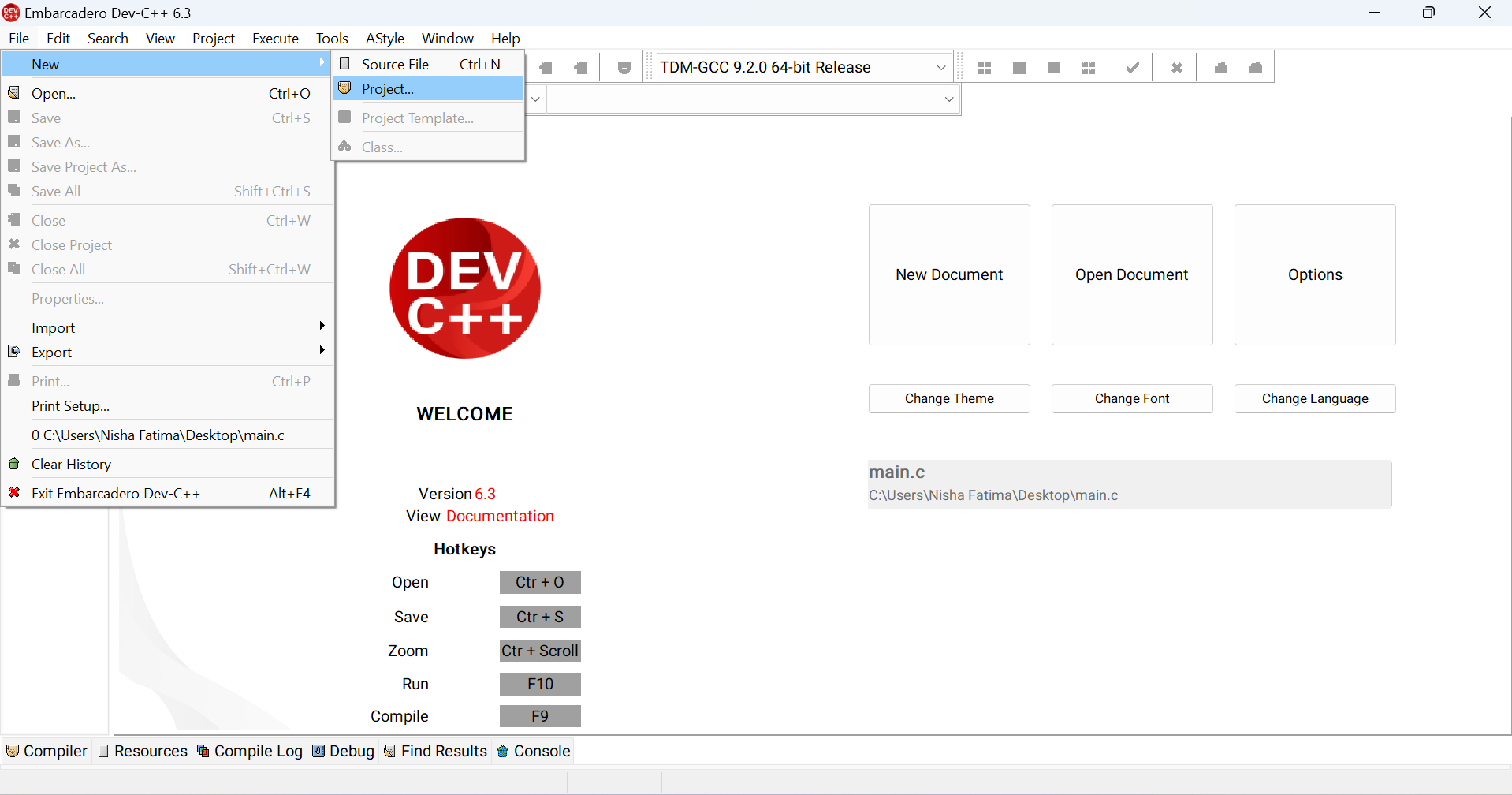
After installation, open Dev C++ from your desktop or start menu.

To start a new project:

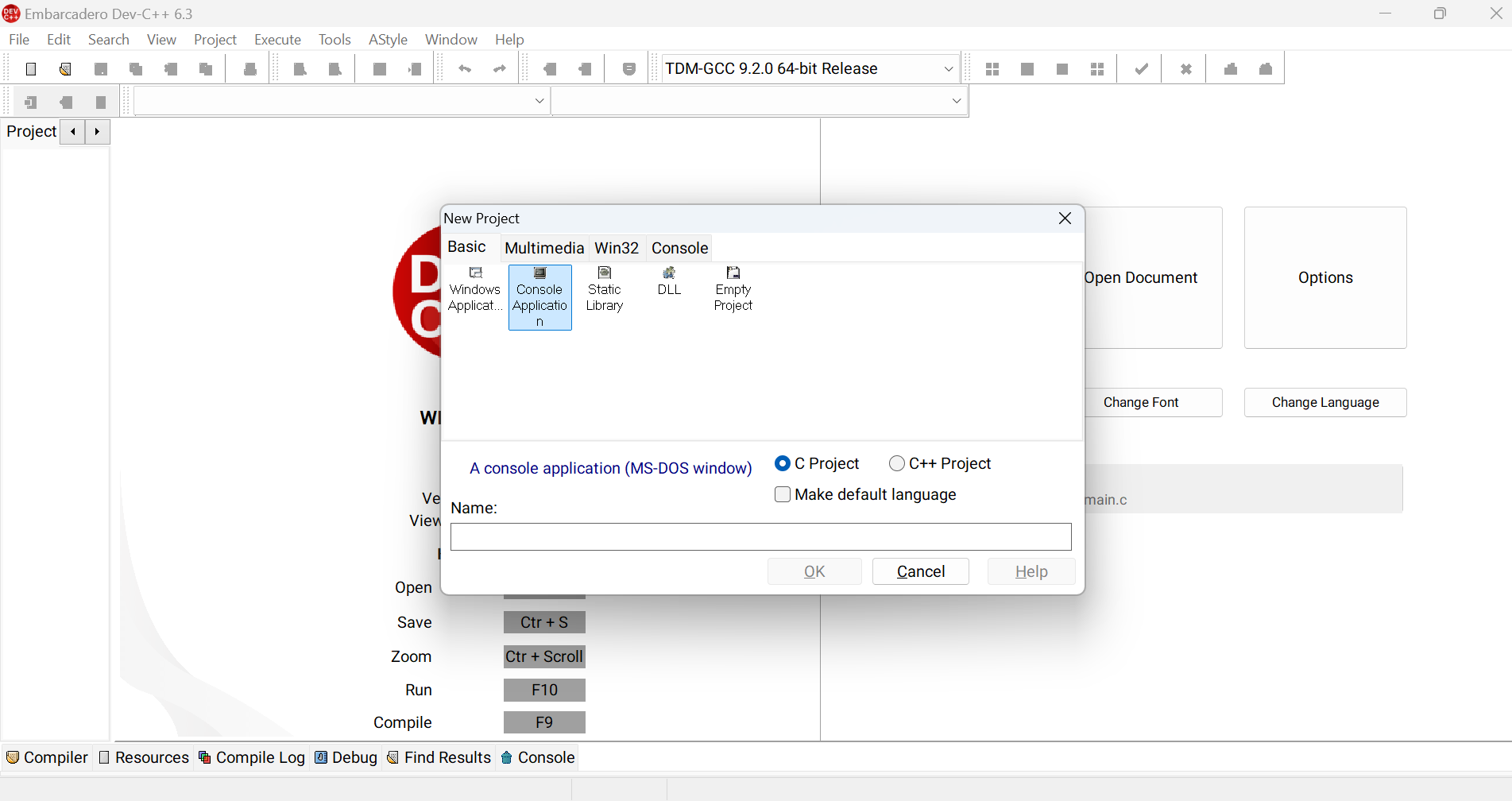
* Click File > New > Source File.
* Write your C code in the editor.
* To save your file, go to File > Save As, and save it with a .c extension (e.g., main.c).



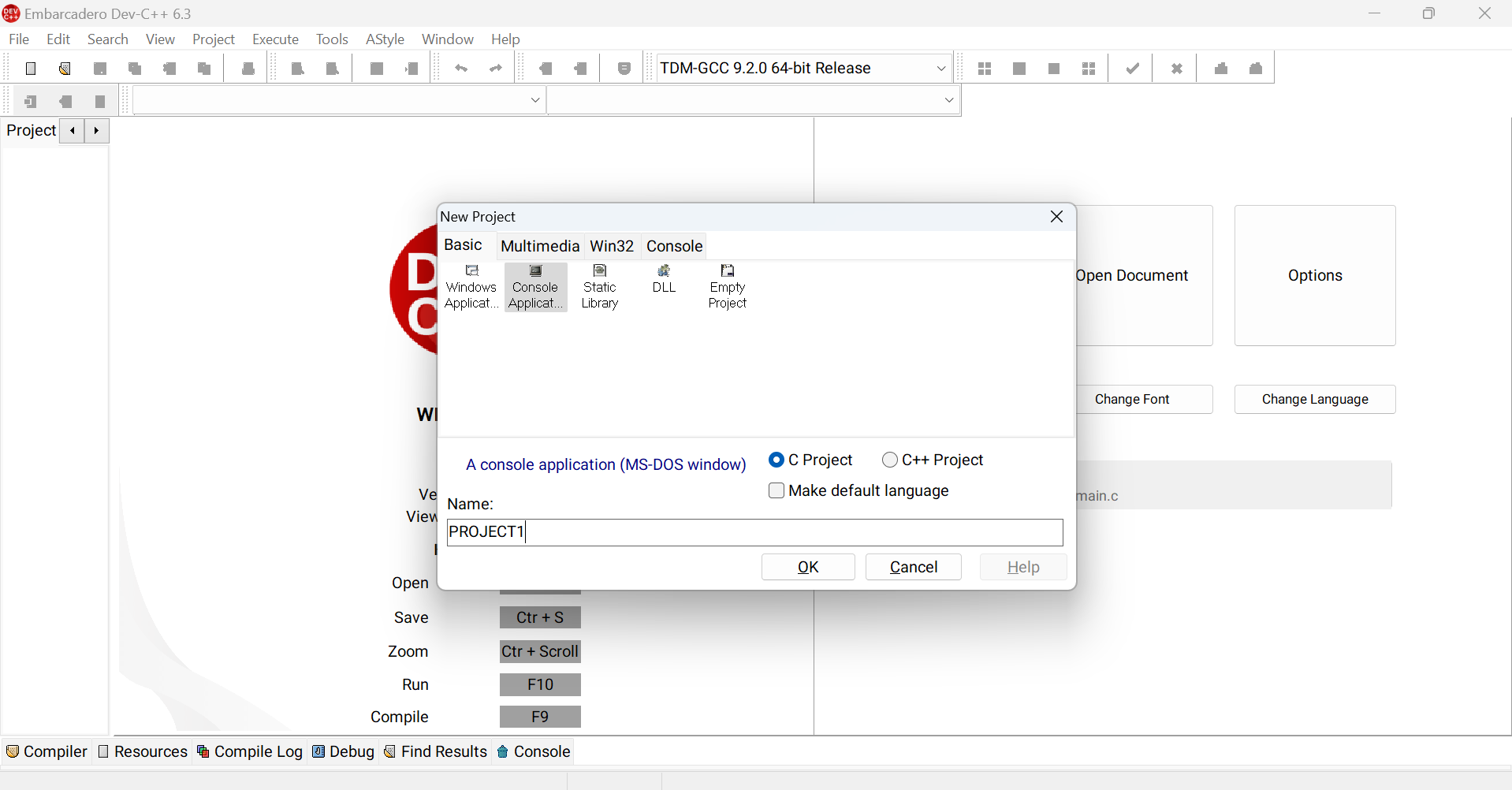
*Figure 2: Front View of Dev C++*



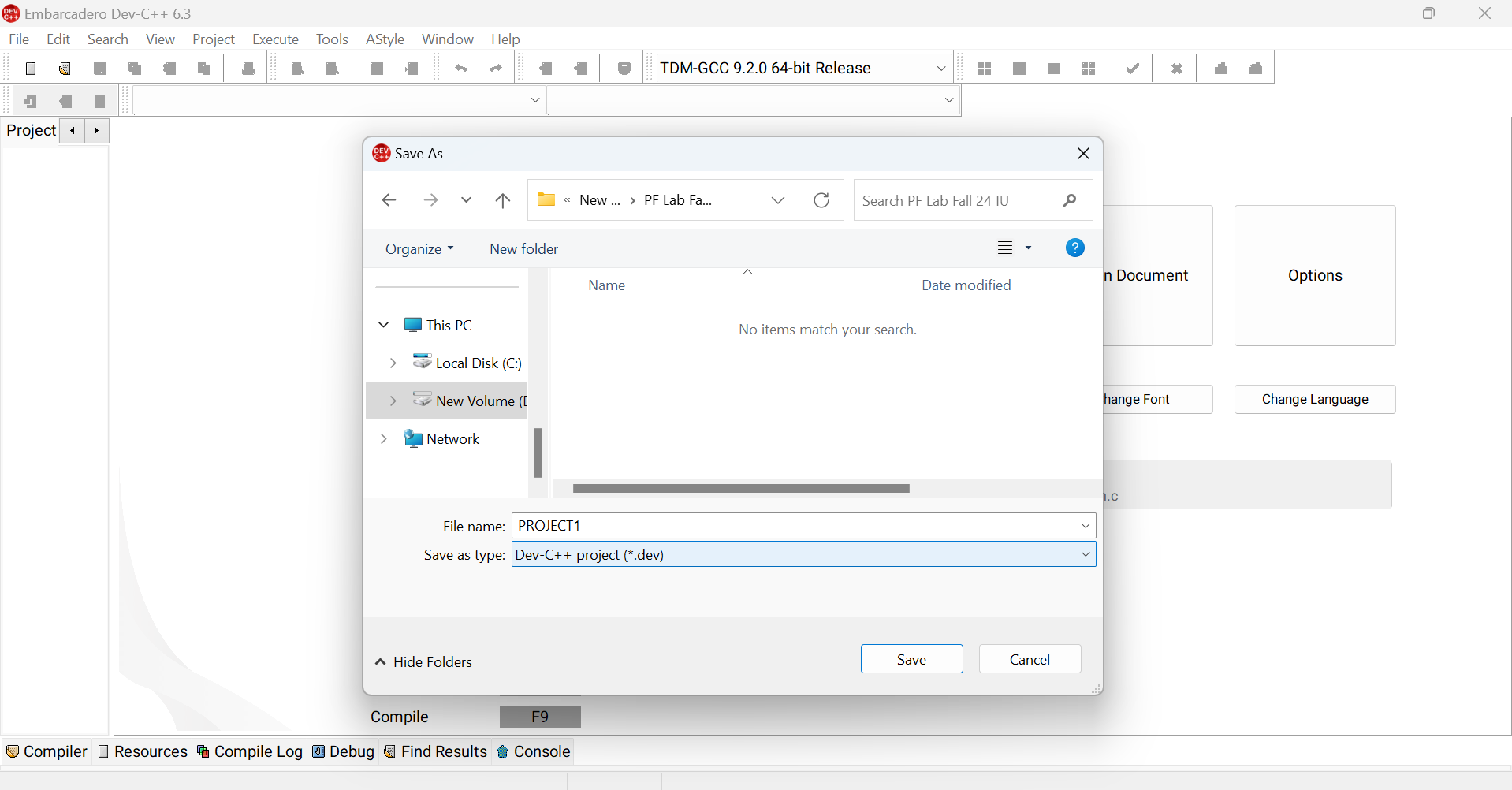
*Figure 3:Select a new project*



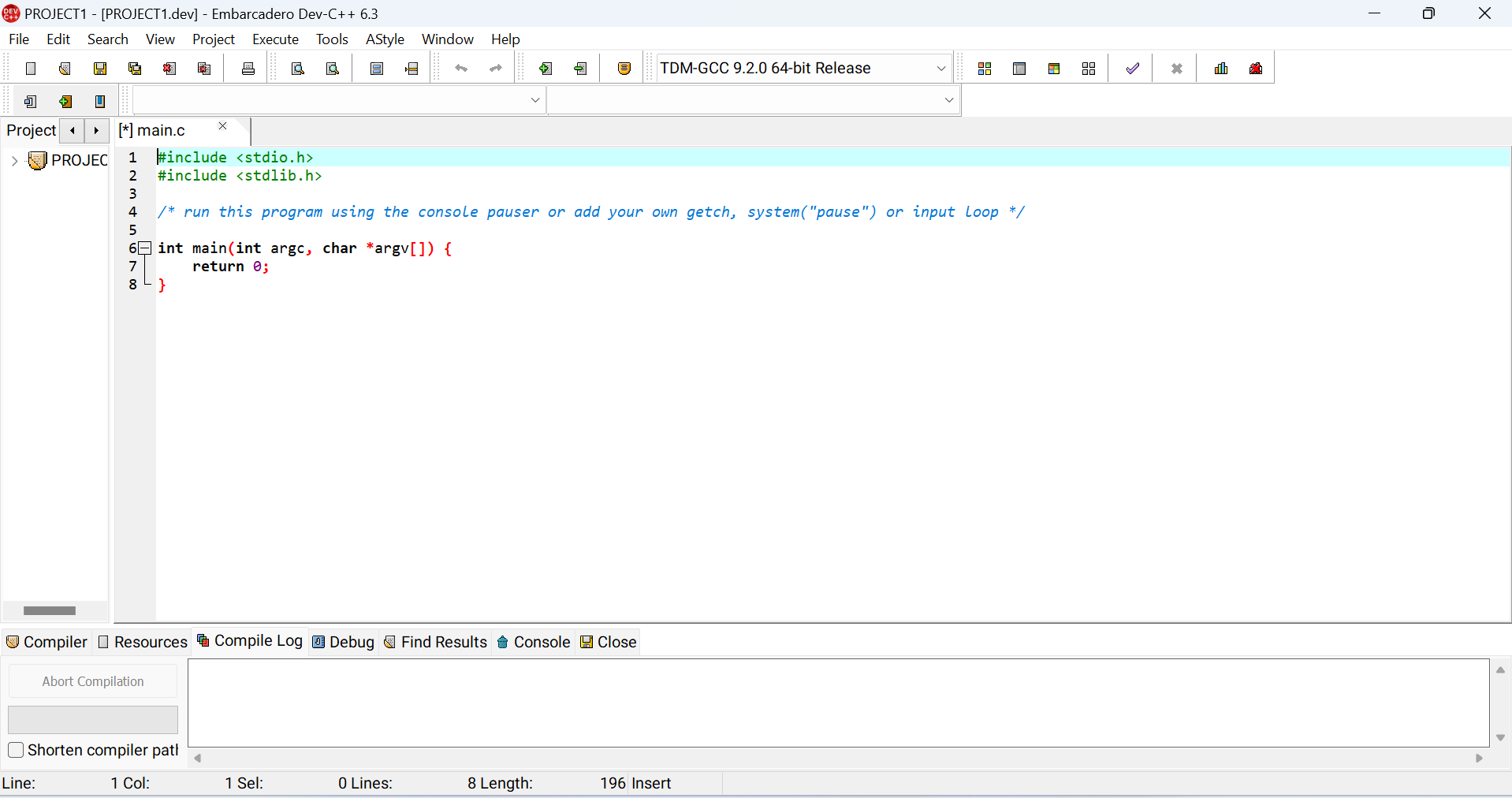
*Figure 4: Creating a new project by selecting console application option*



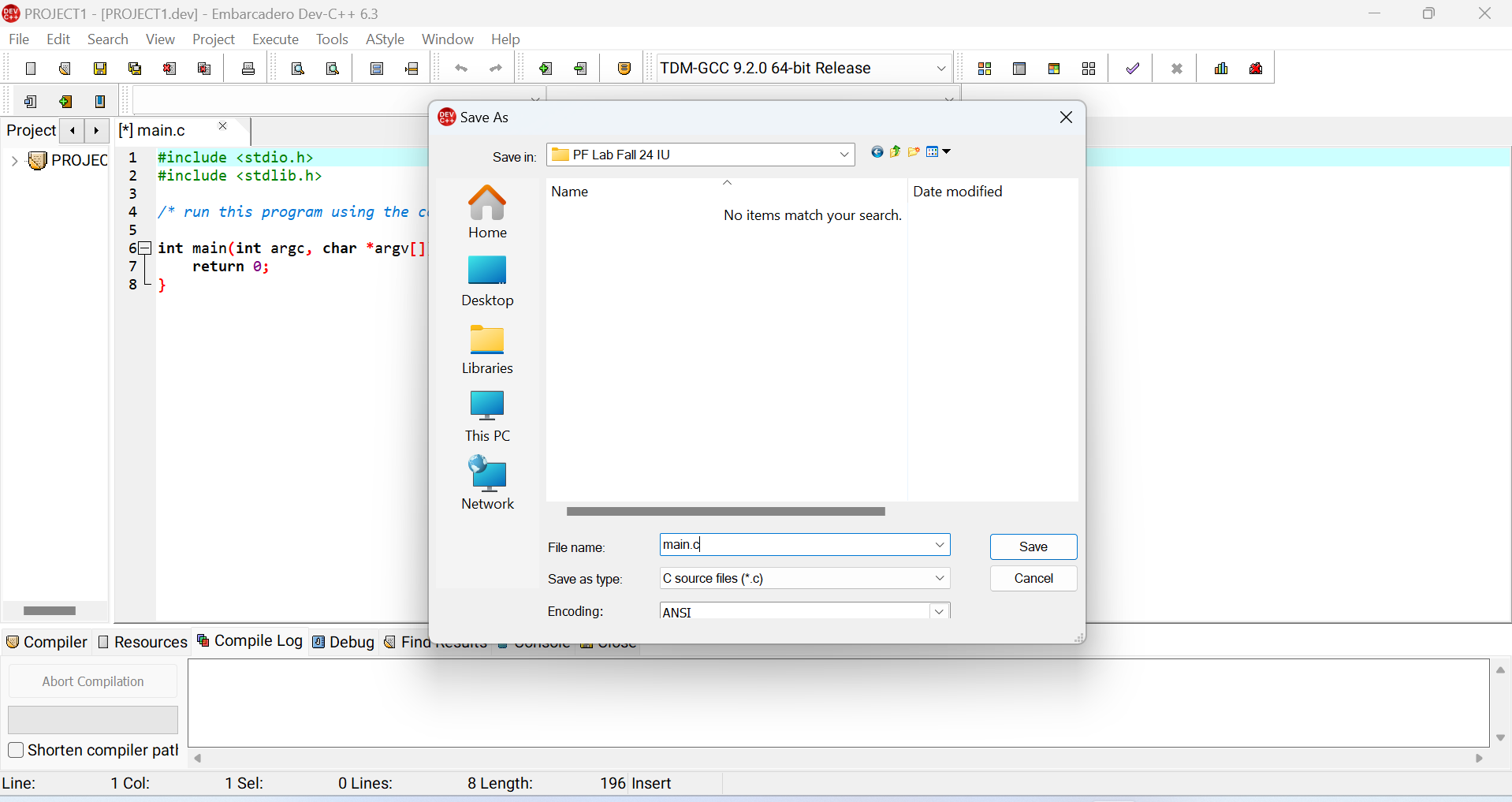
*Figure 5: Write the project name and click “OK”*



*Figure 6: Save your project in non- C drive and create a folder and create your all labs in that folder*



*Figure 7: After saving, your file main.c is created*

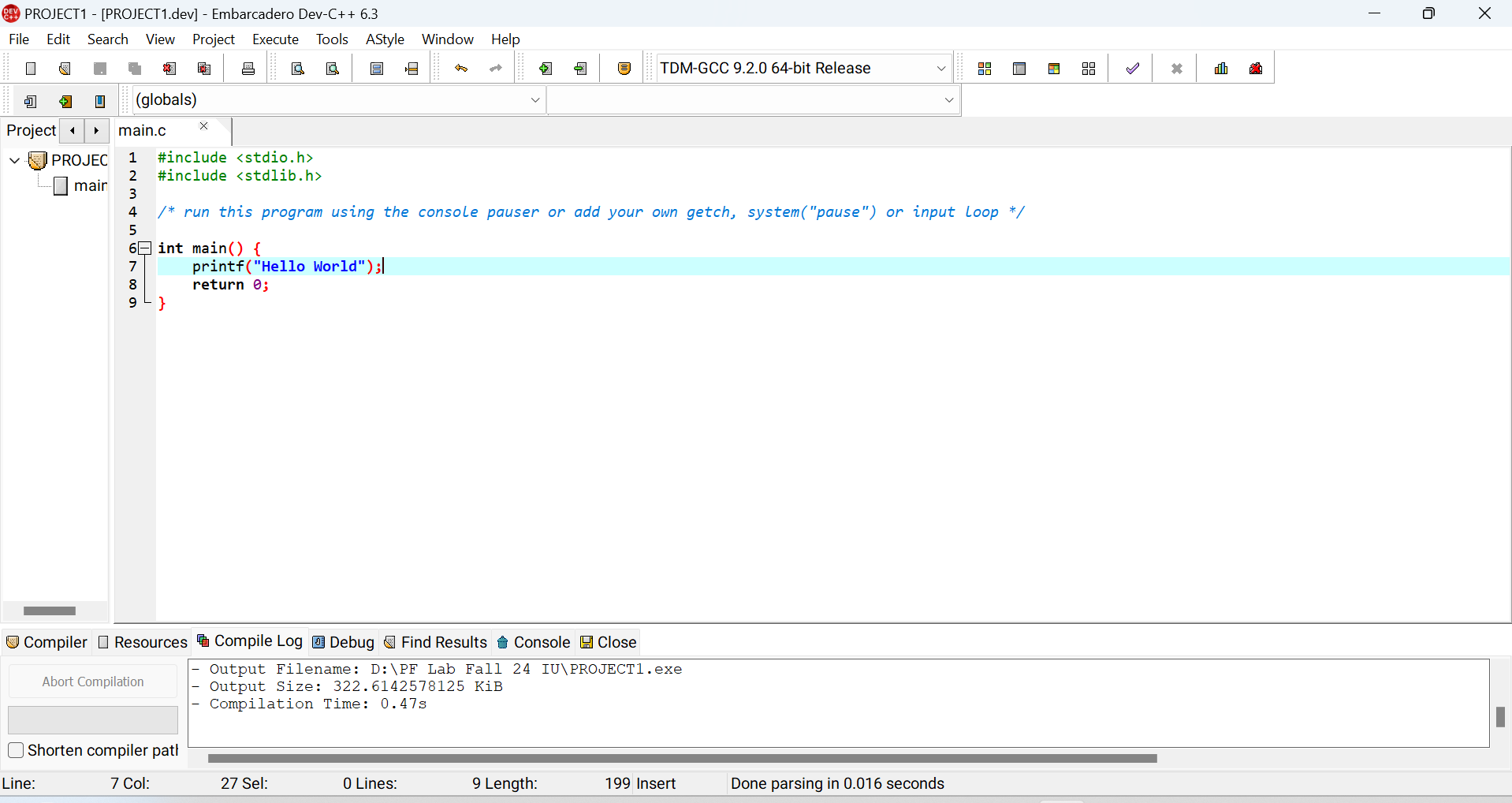


*Figure 8: Save your file with .c extension to run the program*

Before writing a program the following points should be kept in mind.

1. Comment your code properly by // signs.
2. Each statement should be terminated by a semi-colon (;).
3. Indent your code properly.

**Step#6:** **A Simple Program**



*Figure 9: Program on console*

**How does this program work?**

* All valid C programs must contain one main() function. The code execution begins from the start

of main() function.

* Compiler reads source code from top to bottom and left to right.
* printf() is a library function, which is used to print to standard output streams.
* The printf() function is declared in "stdio.h" header file.
* stdio.h is a header file for input and output operations.
* # include is a preprocessor directive whose main job is to include the code from another file, in this case stdio.h.
* Preprocessor directives are instructions to the compiler unlike program statements which are instructions to the CPU.
* In most cases, compiler generates error messages when it encounters printf() or any other library function

and it does not find its respective header file included in the source code

* The return 0; statement is the "exit status or code" of the program. Usually 0 is considered as successful

clean exit.

* String enclosed /\* ... \*/ is a comment. Compiler ignores comment. There are two types of comments

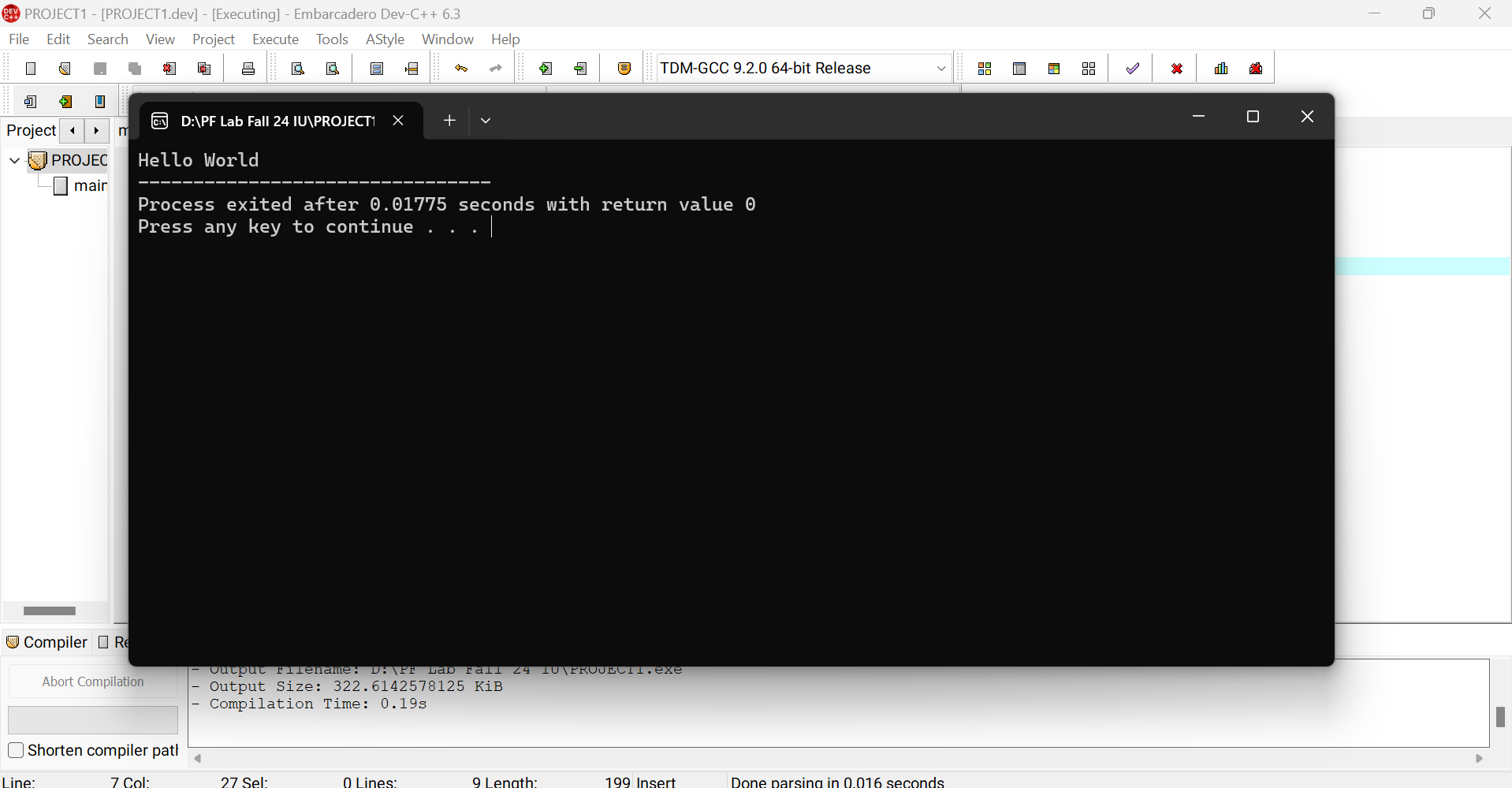
in C language i.e. single line and multiline comment.

* Every program statement must be terminated by ’;’..

**Step#7:** **Seeing the Output:**

Compile and run the program by clicking on the small square option of the Dev C++ or press F11a from

keyboard to see the output. A black screen will appear showing your output



*Figure 10: Output of Program*

1.4 Input/Output (I/O) Operations In C

**1.4.1 Input/Output (I/O) Operations in C**

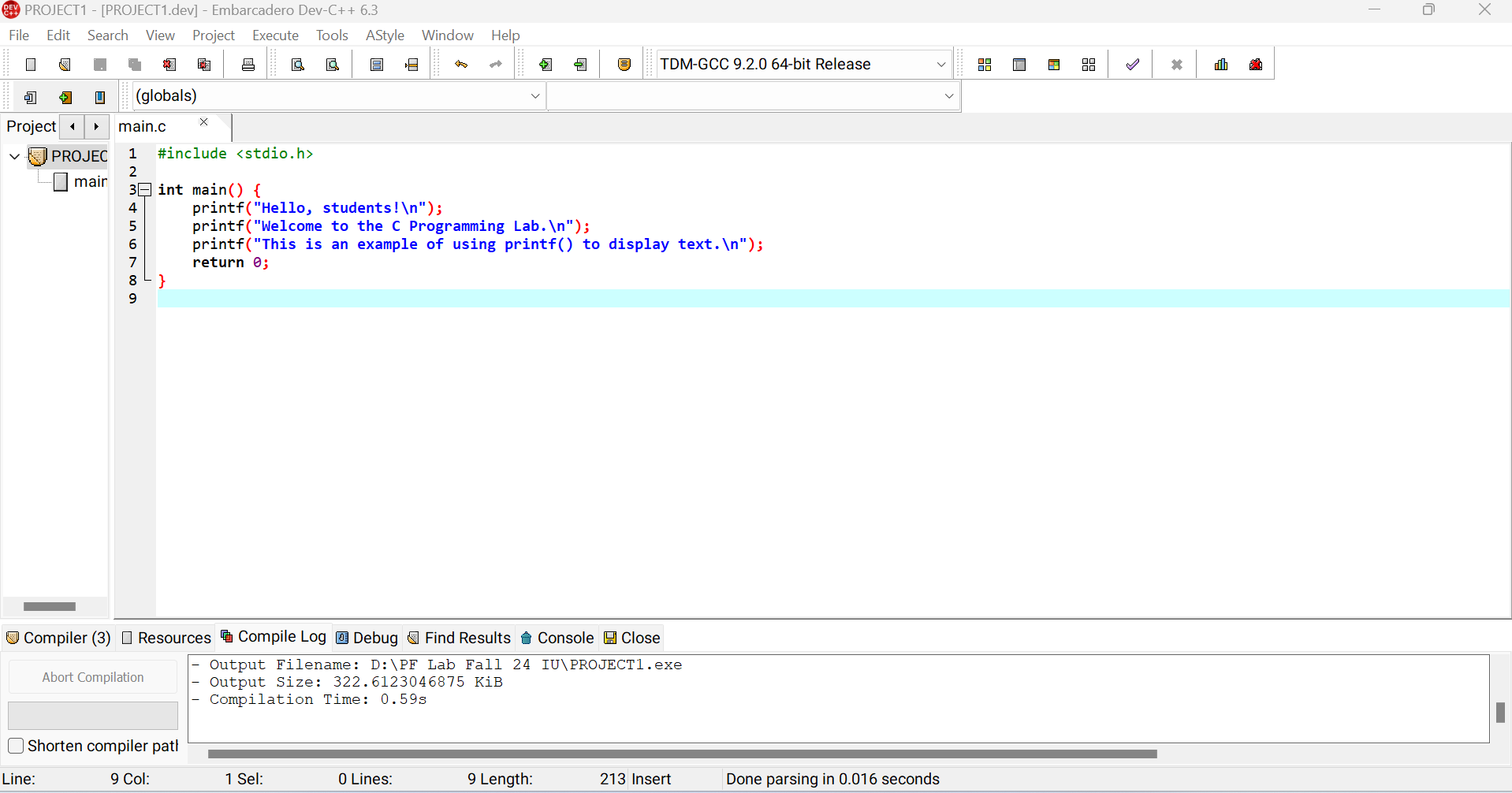
C programming has several built-in library functions to perform Input/Output tasks. Two

commonly used functions for Input/Output are printf()and scanf().

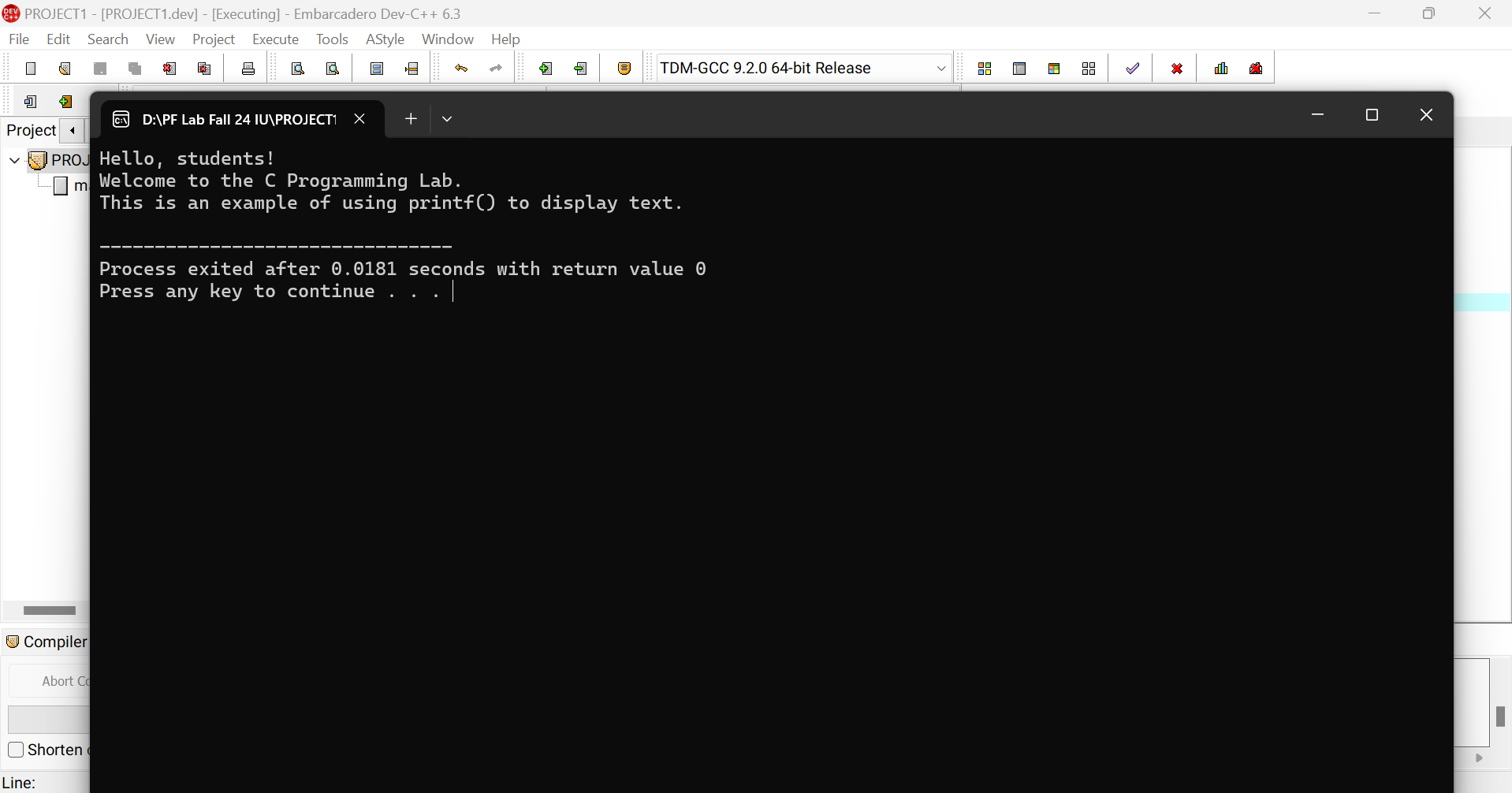
The scanf() function reads formatted input from input entered from keyboard by the user

whereas the printf() function sends output to the screen.

**Program # 2 displaying output**



*Figure 11: Program*



*Figure 12: Output*

1.5 Escape Sequence in C Language

Escape sequence is a special character that causes "escape" from the normal way of characters

are being interpreted or managed. For example, if you want to have a newline then you will

need to use escape sequence \n. Here \ will invoke ’escape’ from the normal execution and alter

the output behavior. Escape sequences are special characters that cannot be typed but have

special meanings in programming. Table. 1 displays some commonly used escape sequence in

practice.

*Table 1: Commonly used Escape Sequences*

|  |  |
| --- | --- |
| **Escape Sequences** | **Character** |
| \b | Backspace |
| \n | Newline |
| \r | Return |
| \t | Horizontal tab |
| \v | Vertical tab |
| \\ | Backslash |
| \’ | Single quotation mark |
| \” | Double quotation mark |
| \? | Question mark |
| \0 | Null character |

1.6 Format Specifiers and Formatted I/O Operations

Format specifiers in C language are used to specify the format of data to be input via standard

input stream or the output to be displayed on standard output or error streams. They

are generally used with printf(), scanf(), etc. functions and their variants. These

functions are used to produce output in a controlled fashion. Here are some of the most commonly

used conversion specifiers:

* %d: Print an integer
* %o , %x: Print an integer in octal, hexadecimal
* %c: Print a character
* %s: Print a string
* %f: Print a floating int (single precision) number
* %e: Print a double precision number, in fixed format
* %g: Print a double in a general format

1.7 American Standard Code for Information Interchange (ASCII)

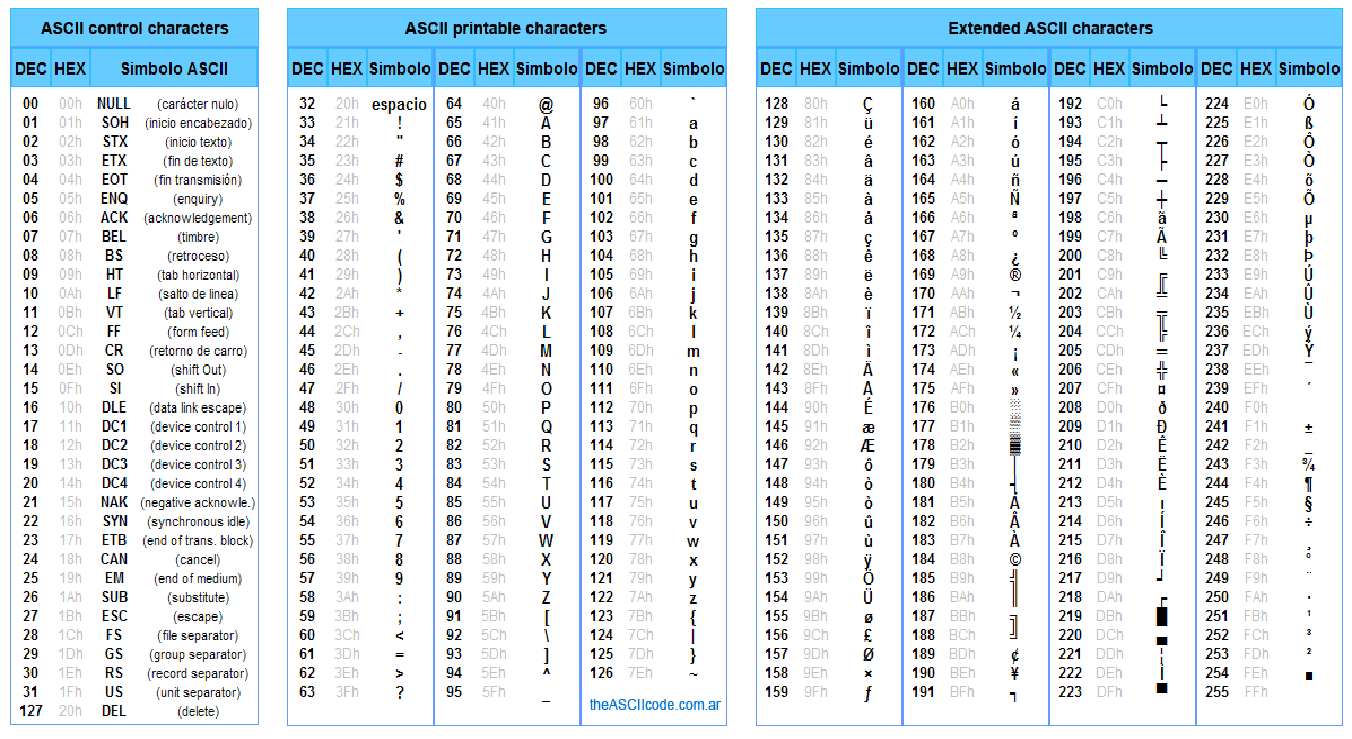
Computers can only understand numbers (binary). ASCII code is the numerical representation

of a character such as ’a’ or ’@’ or an action of some sort. American Standard Code for Information

Interchange (ASCII) consists of 1 byte or 8 bits. So there can be 256 possible characters

in the character set. The entire character set is divided into 2 parts i.e. standard ASCII and

extended ASCII. Figure. 13 shows standard and extended ASCII characters set.



*Figure 13: ASCII Chart*

1.8 Character Sets

A character is a basic element of a word, sentence or paragraph. Using characters, words, expressions

and statements can be formed. In order to interact with the computer, each character

is represented by a unique numeric value. List of character sets is shown in Table. 2. The table

covers only selected special characters and Whitespaces, for further reading please see the

course textbook.

*Table 2: Basic Character Sets in C*

|  |  |
| --- | --- |
| **Types** | **Character Sets** |
| Lower case | a-z |
| Upper case | A-Z |
| Digits | 0-9 |
| Special Character | # $ % ˆ & \_ { } ˜ |
| White space | Tab or new lines or |

1.9 Tokens

A token is the smallest unit of a statement in a program. C language contains different types of

tokens, which are as follows:

**1.9.1 Keywords**

Keywords are the reserved words by the compiler. Every keyword has assigned a fixed special

meaning and therefore, cannot be used as a variable name. There are 32 keywords available in

C which have been listed in Figure 14.



*Figure 14: List of Keywords in C*

**1.9.2 Identifiers**

Identifiers are the symbolic names of Arrays, functions, structures, variables etc. (all will be

covered in later sections). In simple words, identifiers are user defined names. There are

certain rules to be followed for using identifiers as the names of variables, arrays, functions etc.

* An identifier must start with an alphabet (a-z or A-Z).
* After the first alphabet, the remaining letters may be a sequence of alphabets and/or digits.
* digits (0 − 9), ’\_’ and $ are allowed in identifiers
* An identifier can not start with a digit.
* Constants only cannot be used as identifiers.
* identifiers, ABC, abc and AbC are all different.
* In the C compiler, the maximum length allowed for an identifier is 31 characters.
* An identifier should not be a C keyword.

`

**1.9.3 Data types**

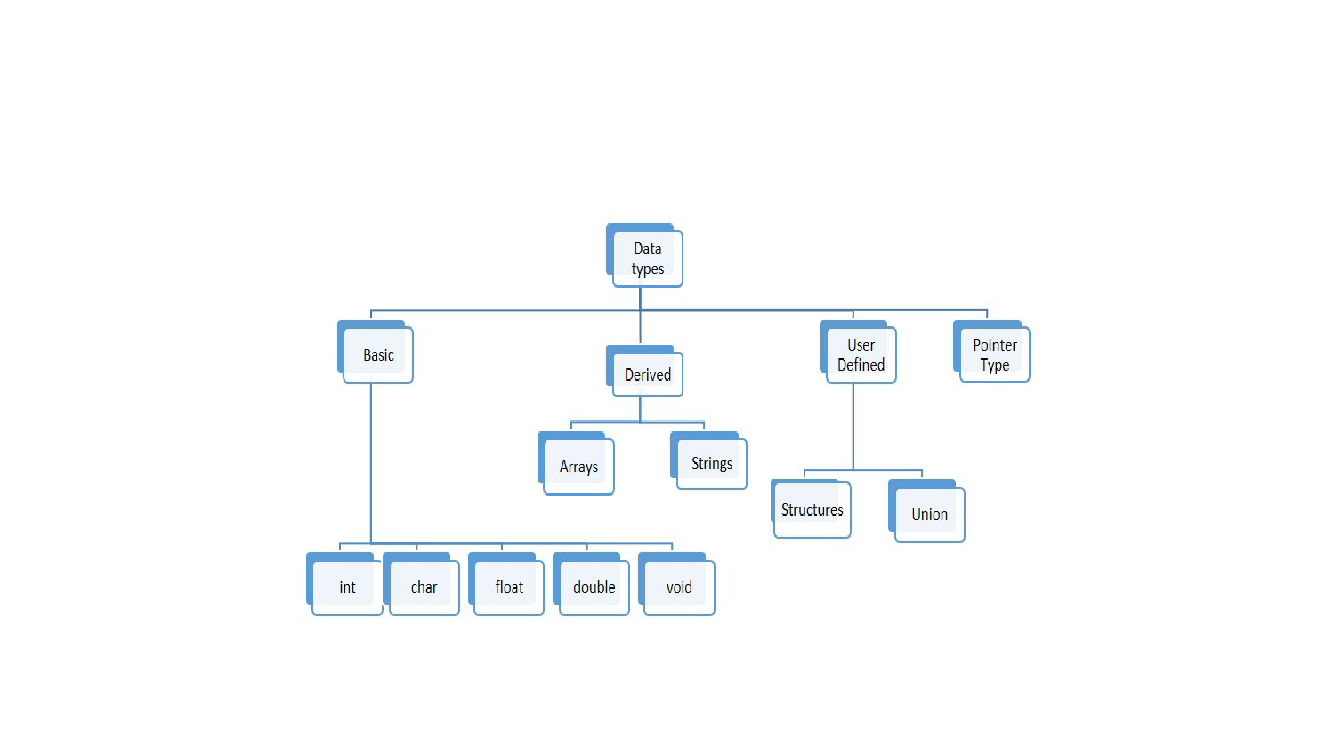
Generally, data is present in the form of numbers (whole/integer or decimal/floating) and characters.

Data types distinguish characters, integers or real numbers from each other. It enable

programmers to select appropriate data type of variables as per the requirements. Specifying

data type of a variable determines how much space it occupies in memory and how the bit

pattern is stored. Data types that are supported in C compilers are shown in Figure 15



*Figure 15: Data types in C*

Here we only cover basic/primitive data types. As shown in Figure. 15, the basic data types of

C language are Integer (int), character (char), floating point (float), and void. Brief explanation

of each data type is given below:

1. **Integer (int)**

C compilers offer short, long, signed and unsigned integer data types. Table. 3 and Table. 4

outline a brief description of these data types.

*Table 3: Short and Long Integer Data type*

|  |  |
| --- | --- |
| **Short Integer** | **Long Integer** |
| Occupies 2 bytes in  Memory | Occupies 4 bytes in  Memory |
| Range is -32768 to  32767 | Range is -2147483648  to 2147483647 |
| Format specifier is  %d | Format specifier is  %ld |

1. **Character (char)**

C compilers offer signed and unsigned character data types. Table. 5 outlines brief description

of character data type.

1. **Floating Point (float):**

In floating point C compilers offer floating and double floating data types. Table. 6 outlines

a brief description of floating point data types.

*Table 4: Signed and Unsigned Integer Data Type*

|  |  |
| --- | --- |
| **Signed Integer** | **Unsigned Integer** |
| Occupies 2 bytes in  Memory | Occupies 2 bytes in  Memory |
| Range is -32768 to  32767 | Range is Range is 0 to  65535 |
| Format specifier is  %d or %i | Format specifier is  %u |

*Table 5: Signed and Unsigned Character Data types*

|  |  |
| --- | --- |
| **Signed Character** | **Unsigned Character** |
| Occupies 1 byte in  memory | Occupies 1 byte in  Memory |
| Range is -128 to 127 | Range is 0 to 255 |
| Format specifier is  %c | Format specifier is  %c |

*Table 6: Floating and double floating data type*

|  |  |
| --- | --- |
| **Floating** | **Double Floating** |
| Occupies 4 bytes in  Memory | Occupies 8 bytes in  Memory |
| Range is 3.4e-38 to  3.4e38 | Range is 1.7e-308 to  1.7e-308 |
| Format specifier is %f | Format specifier is  %lf |

* + 1. **Constants**

Constant meaning in C is the value, which remains the same during the execution process of a program. C language offers several types of constants, which have been explained below:

1. **Numerical constants:**

Numerical constants are of two types:

* **Integer constants:**

Integer constants are the constants, which are in whole numbers form (without decimal)

and require 2 to 4 bytes to store in memory. These include decimal,

hexadecimal as well as octal numbers and can be found in both positive and negative

forms. Examples of integer constants are: 13 (decimal), 0171 (octal) and 0x 54

(Hexadecimal) etc.

* **Real constants:**

Real constants are also known as floating point constants. These are the numbers,

which contain decimals. Examples of real constants are: 2.1, 1.342, 30.03 or 3.7e-5 etc.

1. **Character constants:**

Character constants are also of two types:

* **Single character constants:**

Single character constants are based on single character only. If a single character,

single digit or a single special character is enclosed in pair of single quotation

marks(’ ’) , it represents a character constant. Examples of character constant are:

’A’, ’b’, ’4’, ’-’ etc.

* **String constants:**

When a sequence of characters is enclosed in double quotation marks (" "), it represents

a string. A string can be a combination of all possible types of symbols.

Examples of strings are: "Pakistan", "123a", "333" etc.

**Syntax of Constant**

**const** data\_type variable\_name ;

**Examples of Constant**

**const** double PI = 3.14;

**const** int a = 2;

**1.9.5 Variables**

A variable is a container (storage area) to hold data. To specify the storage region, a distinctive

name should be provided to each variable. Variable names are just the symbolic representation

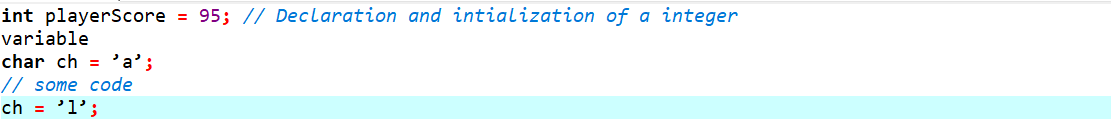
of a memory location. In the program, a variable’s value can be changed, hence the name of

the variable.

**Syntax of Variable**

data\_type variable\_name ;

**Example of Variable**



*Figure 16: Variable initialization*

**Note:** A variable is declared, anywhere in the program. If it is declared outside main function

It is called a global variable. Such variables can be used anywhere in the program. If it declared

inside the main function it is called local variable. Such variable cannot be used outside main

function.

When a variable is declared but not initialized with some value it stores a garbage value.

## Observation

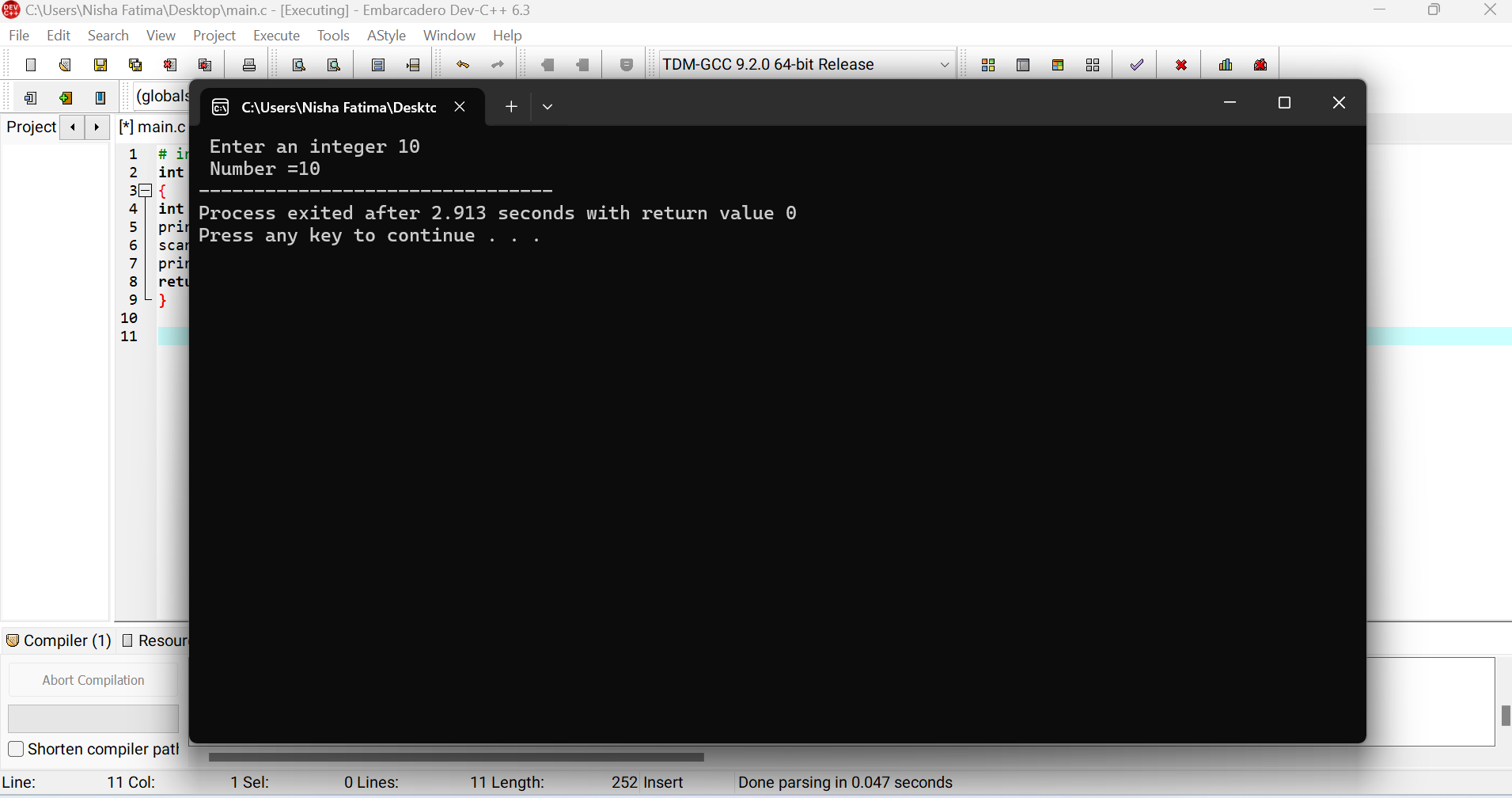
|  |
| --- |
|  |

**Programs Using Variables, User defined I/O operations, Format specifiers and Escape sequence**

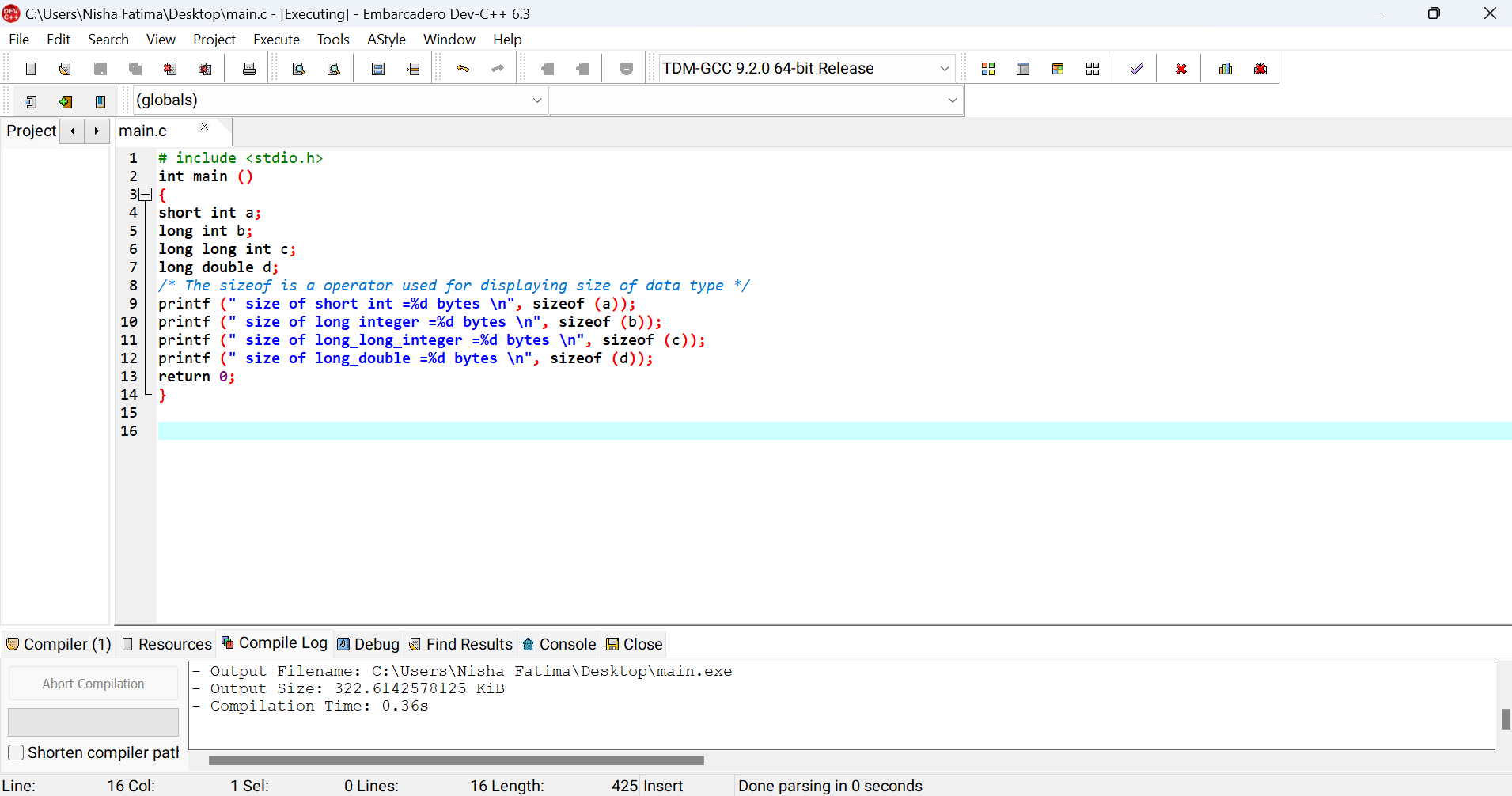
**Program # 3: Integer Variable Input/Output with User Defined Input**



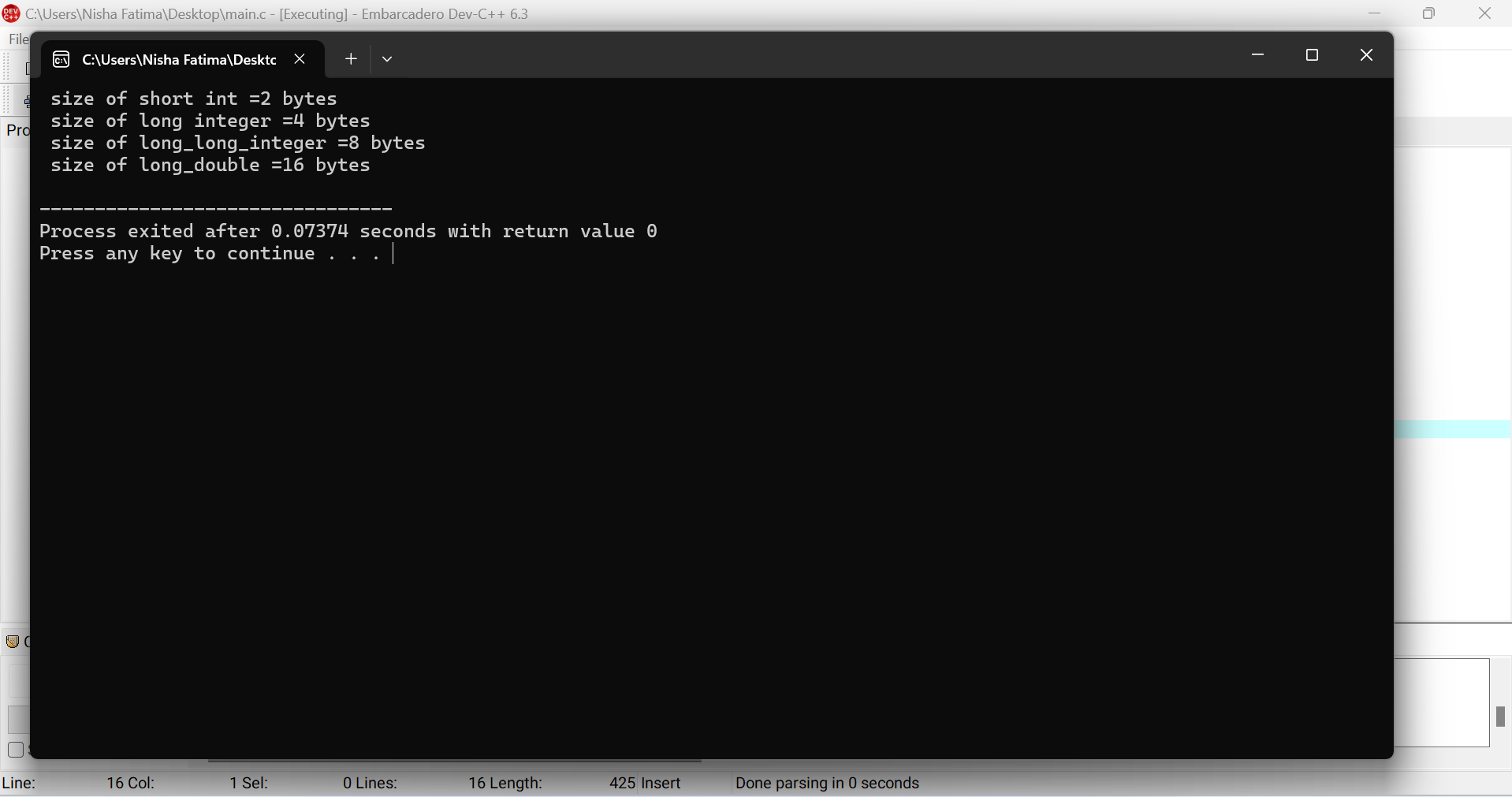
**Output #3:**

****

**Program # 4: Variable Declaration**

****

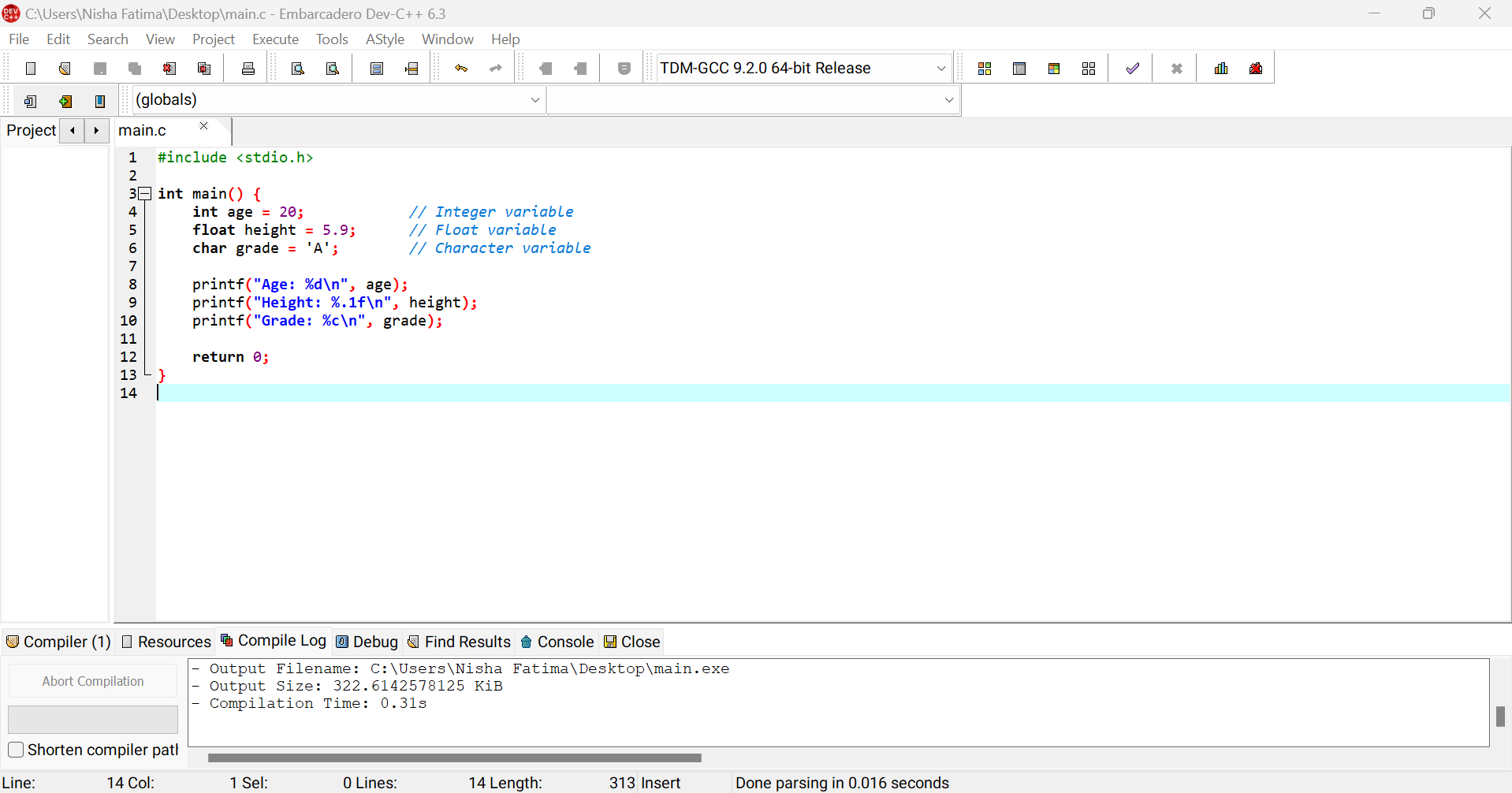
**Output #4:**



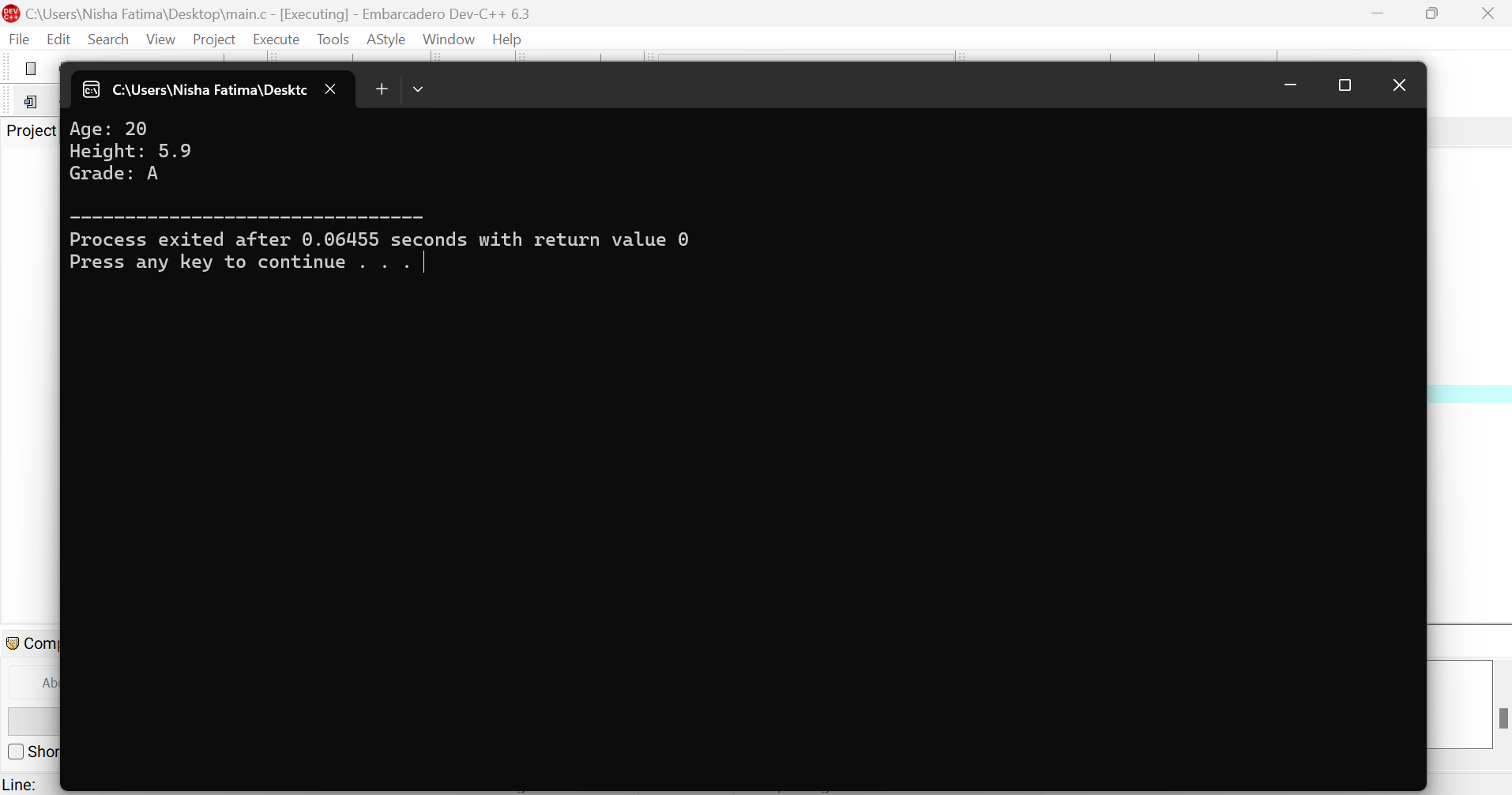
**Note: sizeof()** is an operator which tells about the size of data types depending upon on your

OS.

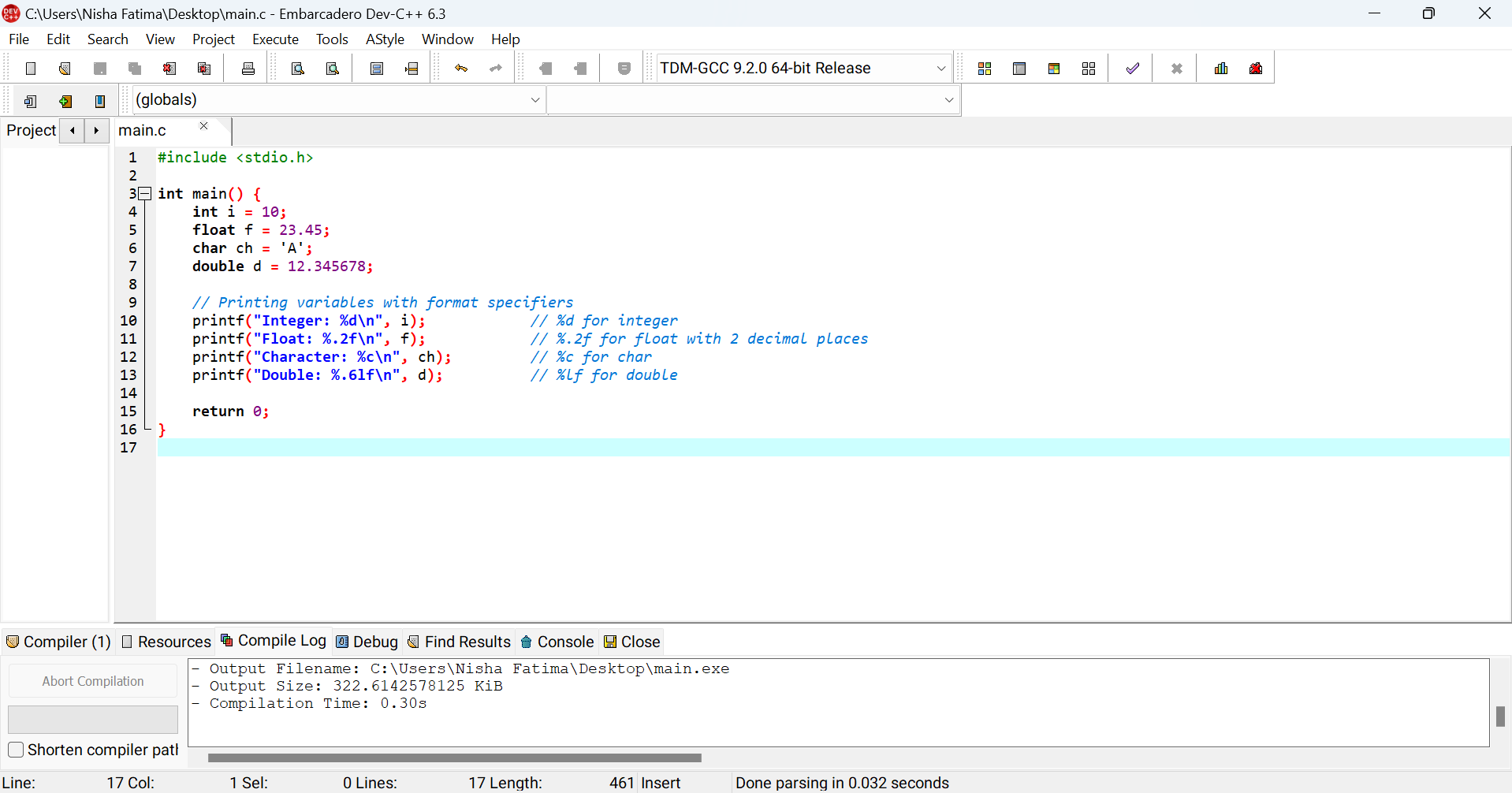
**Program # 5: Variable Declaration and Initialization**

****

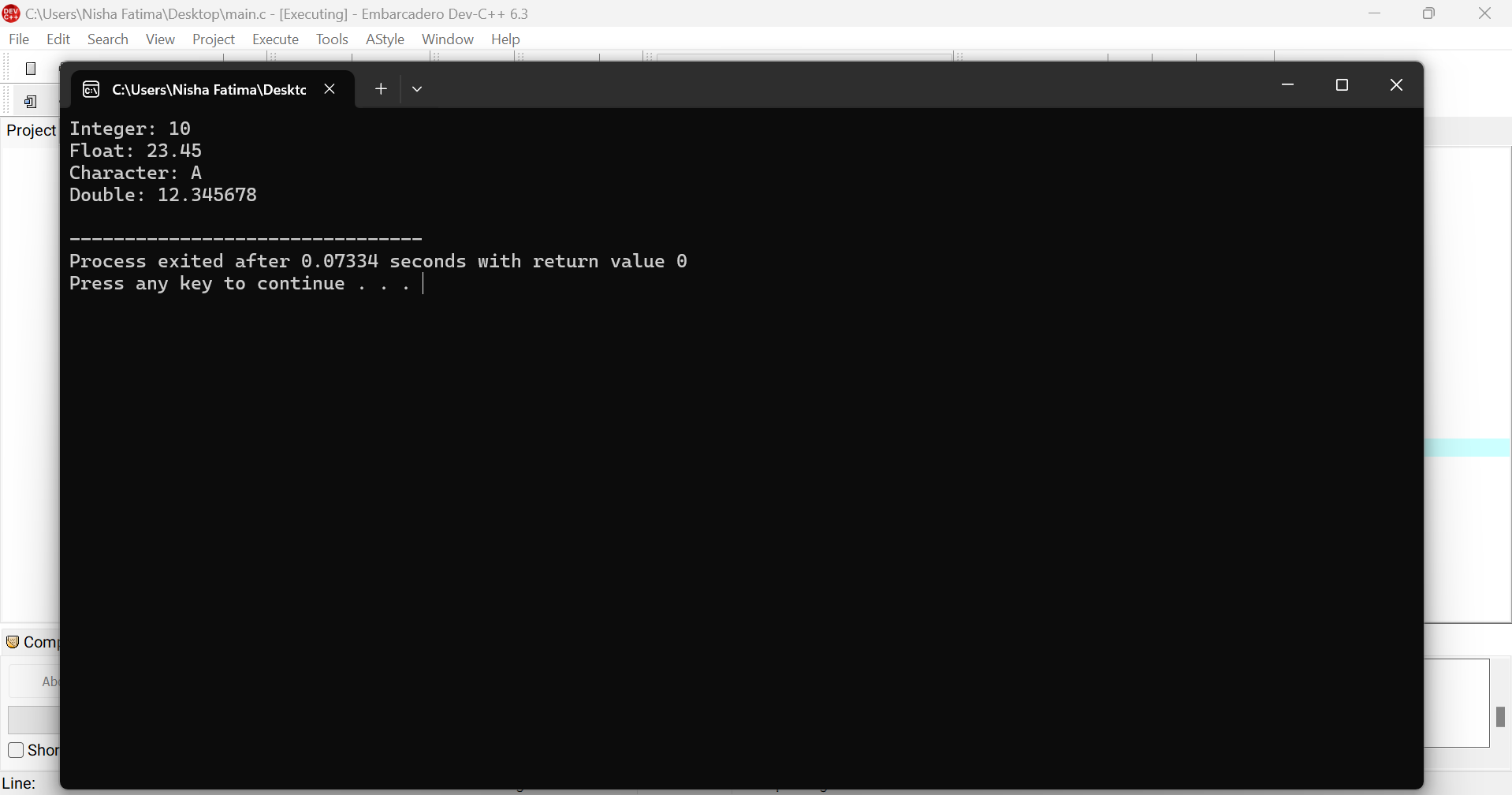
**Output #5**

.

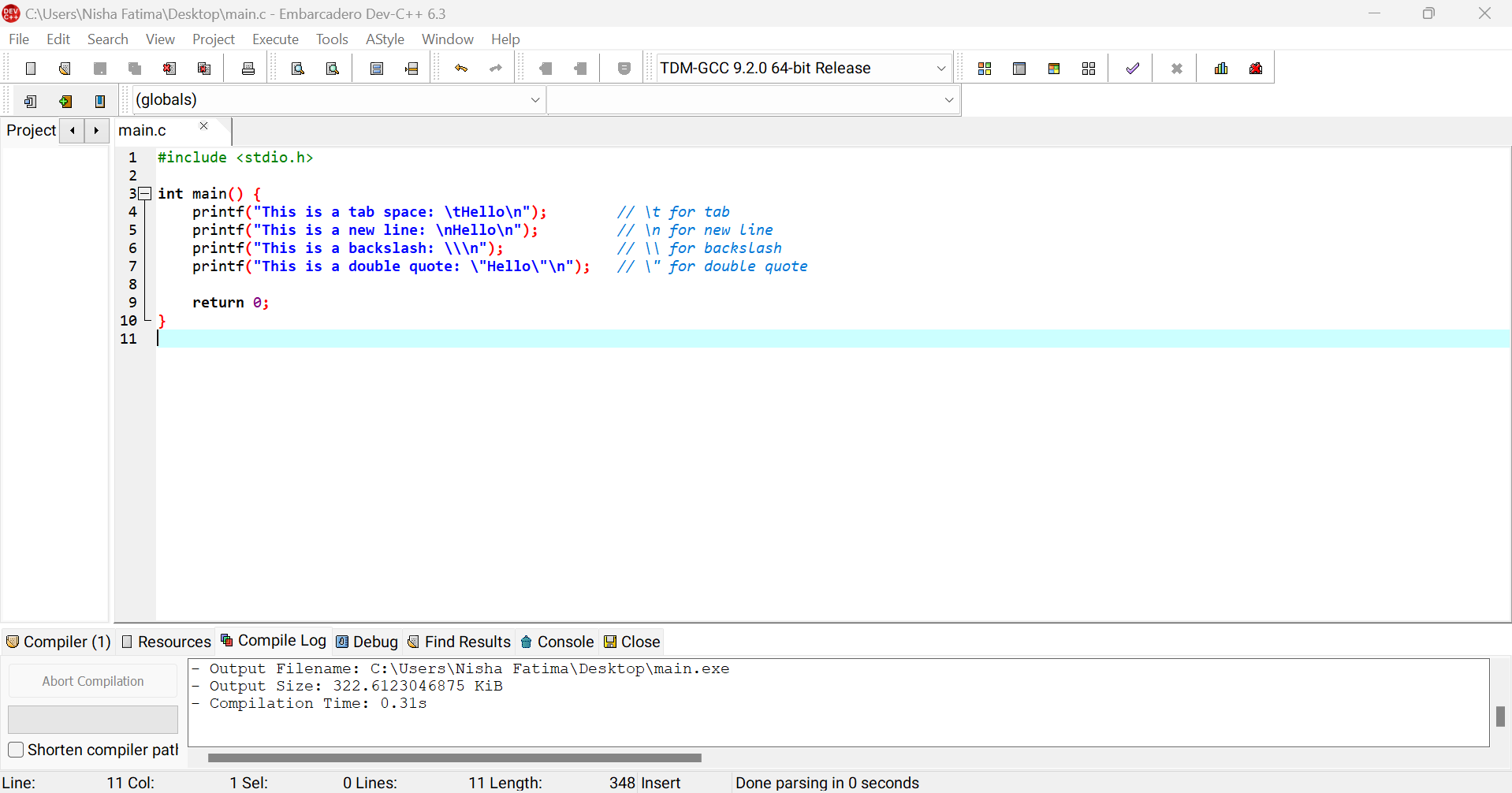
**Program # 6: Format Specifiers**

****

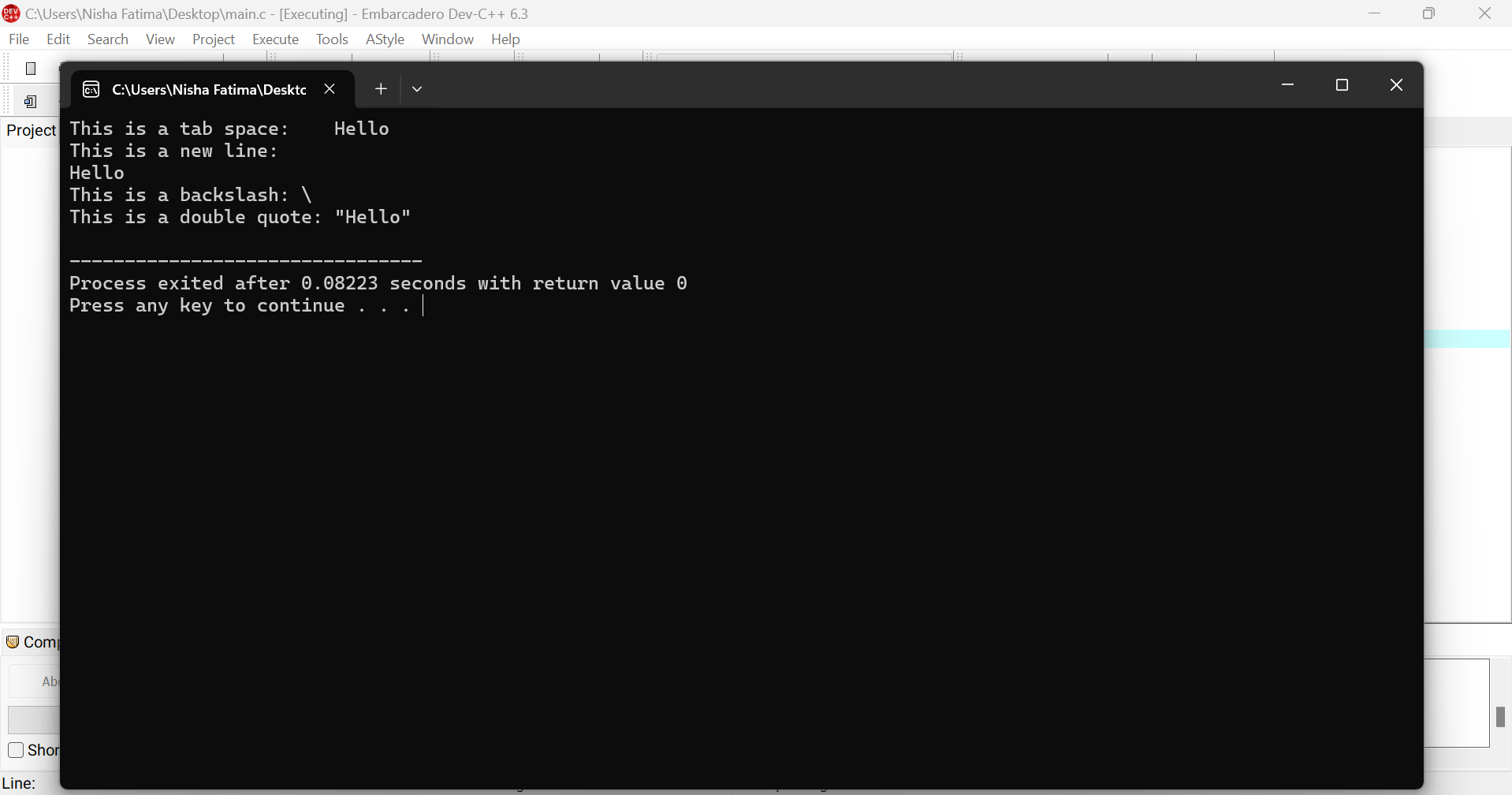
**Output #6:**



**Program # 7: Escape Sequence**

****

**Output #7:**

****

## Lab Tasks:

1. Write a C program to calculate the area of a rectangle
2. Write a C program to calculate the simple interest using the formula SI = (Principal \* Rate \* Time) / 100. (SI stand for Simple Interest)
3. Write a C program to swap two numbers using a third variable.

## Home Tasks

1. Write a C program to convert temperature from Celsius to Fahrenheit. Use the formula:

Fahrenheit = (Celsius \* 9/5) + 32

1. Write a C program to calculate the volume of a cube. The volume of a cube is given by:

Volume = side \* side \* side

1. Write a C program to take three numbers as input and calculate their sum and average.

**Lab Assignment Submission Instructions**

* 1. Screenshot your code and output and paste it into a Word file.
  2. Copy paste your code into that word file according to the Question number.
  3. Number your Word document as question number e.g.Q1.sb, Q2.sb, etc. (Q is in upper case)
  4. Create a new folder with your Name and ID e.g. Naveed Ahmed (61452)\_Lab 03 Assign.
  5. Copy all the Word document and source files into this folder.
  6. Right-click on the folder you created and create a zip file by selecting the option
  7. “Send to” and selecting “Compressed (zipped) folder” [for windows].
  8. “Create Archive” and change the option to “.zip” instead of “ .tar.gz” and click on “Create”. [for linux]
  9. Now make sure a zip file is created with your name and id e.g. Naveed Ahmed

(61452).zip.

* 1. Submit Assignment on LMS.

*Note:*

*Plagiarism is strictly prohibited. Any instance of plagiarism will result in a deduction of 100 points. Ensure that all work submitted is original.*

## Discussion and Analysis of Results

|  |
| --- |
|  |

## Conclusion

|  |
| --- |
|  |

**Lab 04**

# **Conditional Statements**

## Objective:

Using Selection Building Blocks in C (if, if-else, Nested if-else, and Cascaded if-else)

To understand and implement selection building blocks in C programming:

- if statement

- if-else statement

- nested if-else statement

- cascaded if-else statement

-switch case

## Required Equipment / tools:

Dev C++

## Introduction:

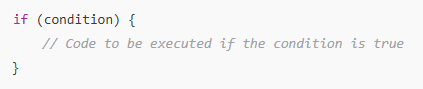
Selection building blocks are control flow statements that allow us to make decisions in our programs based on conditions. They are essential for writing programs that react differently depending on inputs or other conditions.

## Procedure

**4.1.1 if Statement:**

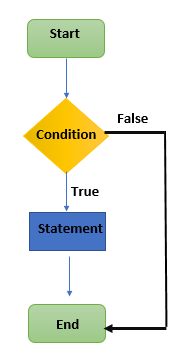
The simplest form of selection statement. It checks a condition, and if the condition is true, the code block associated with it gets executed.

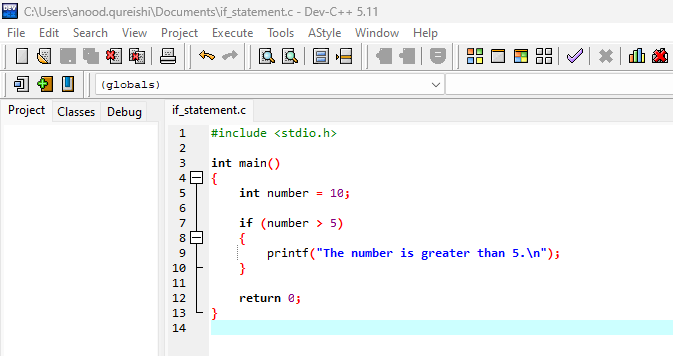
**Syntax:**



*fig 4.1.1.1*

**flowchart:**

**ample:**



*fig 4.1.1.2*

**Output:**



*fig 4.1.1.3*

**Explanation:**

If number > 5 evaluates to true, the message "The number is greater than 5." is printed.

**4.1.2 if-else Statement:**

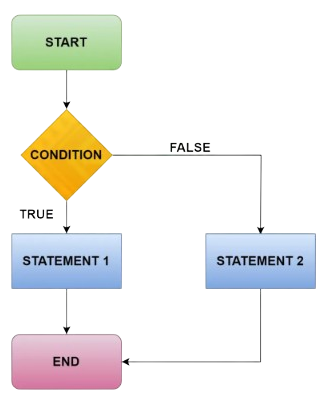
The if-else structure provides two pathways: one if the condition is true, and another if the condition is false.

**Syntax:**

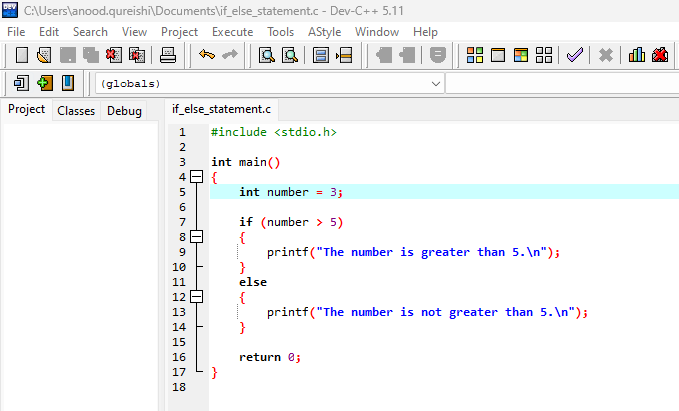


*fig 4.1.2.1*

**Flow Chart:**

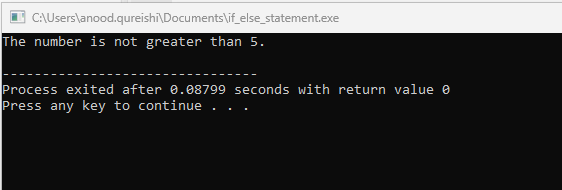


**Example:**



*fig 4.1.2.2*

**Output:**



*fig 4.1.2.3*

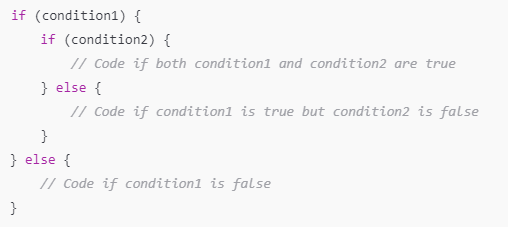
**Explanation:**

If the condition number > 5 is false, the else block is executed, printing "The number is not greater than 5."

**4.1.3 Nested if-else Statement:**

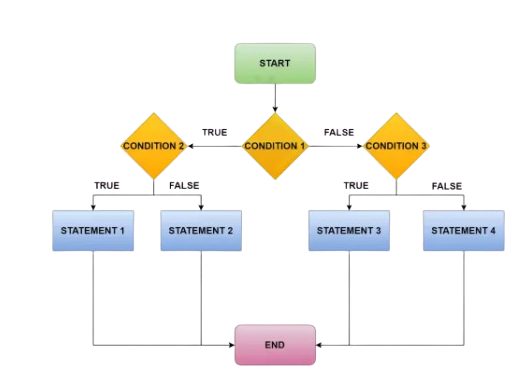
In this structure, you can place one if-else statement inside another if-else block. This allows for checking multiple conditions sequentially.

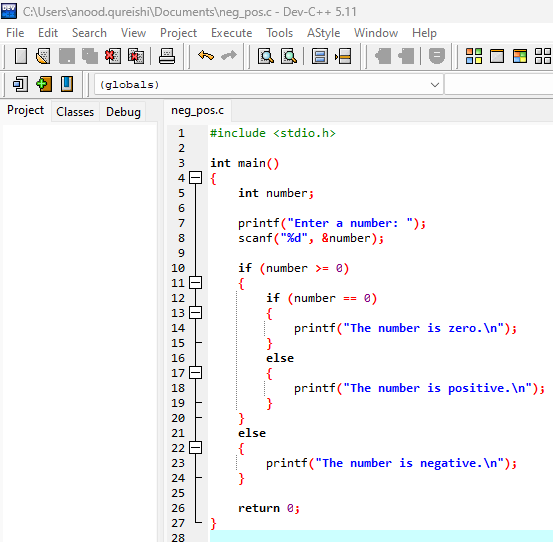
**Syntax:**



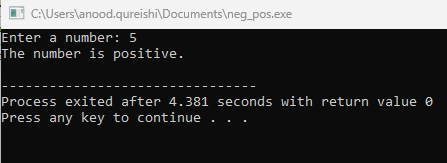
*fig 4.1.3.1*

**Flow Charts:**

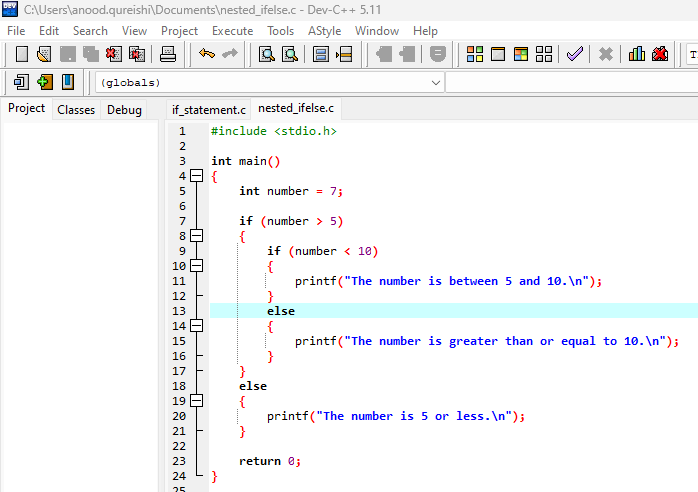




**Output:**

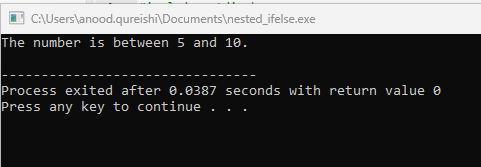


**Example:**



*fig 4.1.3.2*

**Output:**



*fig 4.1.3.3*

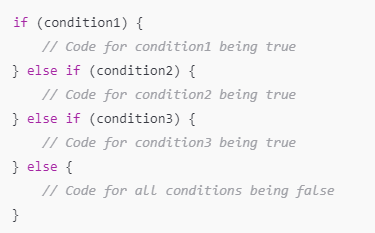
**Explanation:**

The program first checks if the number is greater than 5, and then if it is also less than 10.

**4.1.4 Cascaded if-else Statement:**

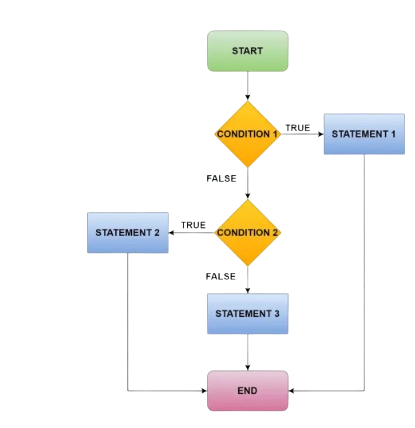
A cascaded if-else structure checks multiple conditions, each with its own branch, until one condition is true.

**Syntax:**

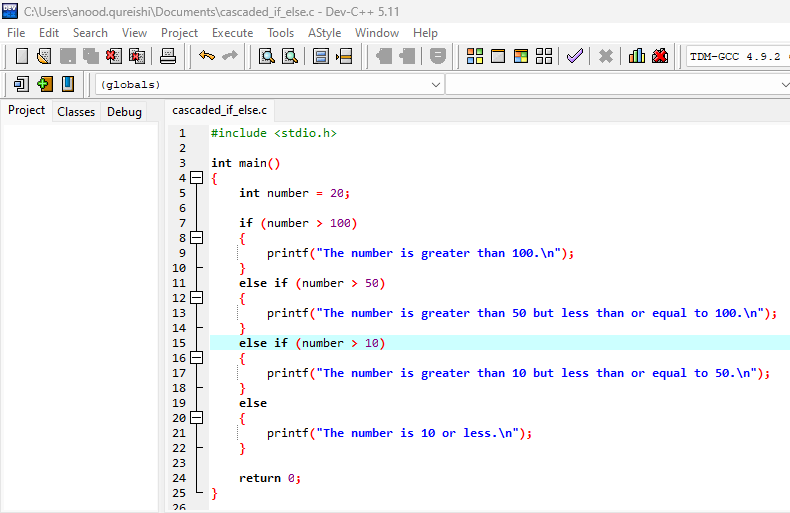


*fig 4.1.4.1*

**Flow Charts:**

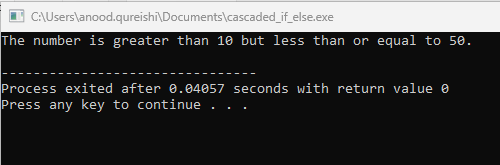


**Example:**



*fig 4.1.4.2*

**Output:**



*fig 4.1.4.3*

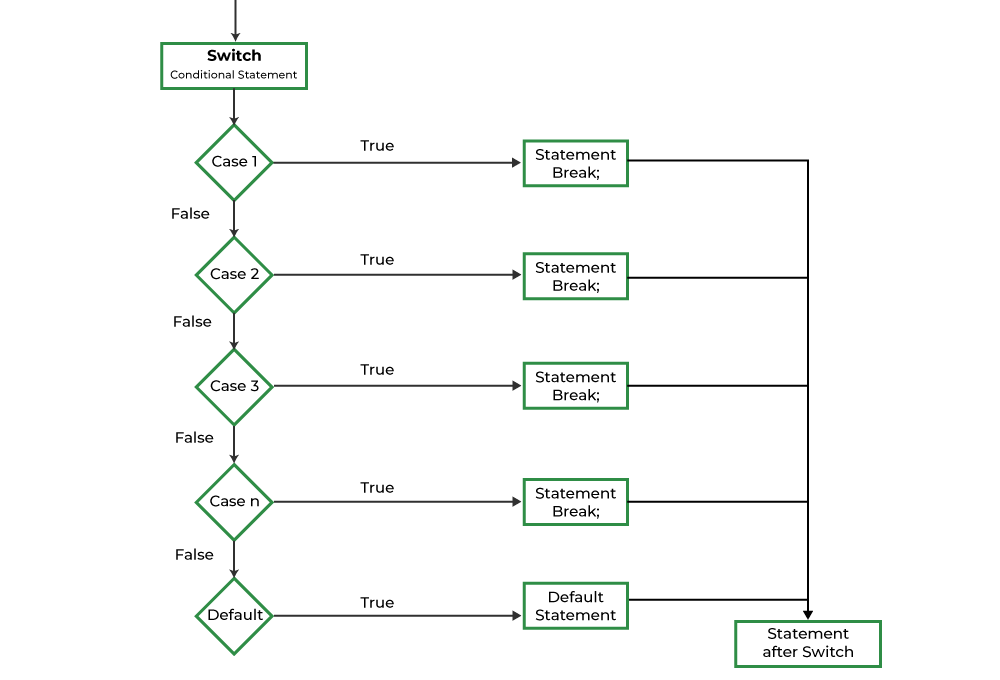
**Explanation:**

The program evaluates each condition in sequence. If number > 100 is false, it moves to the next condition, and so on, until it finds a true condition or reaches the final else block.

**4.2 Introduction to Switch Case**

The **switch** statement is a control structure that allows you to choose between multiple options based on the value of a variable or expression. It’s often used when you have a variable that can take multiple distinct values, and you want to execute different code blocks based on each value.

**Procedure**

**4.2.1 Flow chart for Switch case**

**4.2.2 Rules for Switch Case**

* 1. **Expression Type**
* The switch expression must result in an integer or character value.
* You cannot use floating-point numbers (float, double) or strings directly in a switch expression.
  1. **Case Labels**
* Each case label must be a constant integer or character value.
* You cannot use variables or expressions that calculate values at runtime as case labels.
* The case values must be unique within the same switch block.

1. **Break Statement**

* Each case should generally end with a break statement to prevent "fall-through" (execution continuing to the next case even if it doesn't match).
* If break is omitted, execution will continue to the next case, which is called "fall-through behavior."
* In some scenarios, fall-through can be intentional to execute multiple cases, but it should be done carefully.

1. **Default Case**

* The default case is optional but useful for handling any unmatched values.
* It acts like an "else" statement in an if-else structure.
* default can be placed anywhere within the switch, but it’s usually placed at the end for readability.

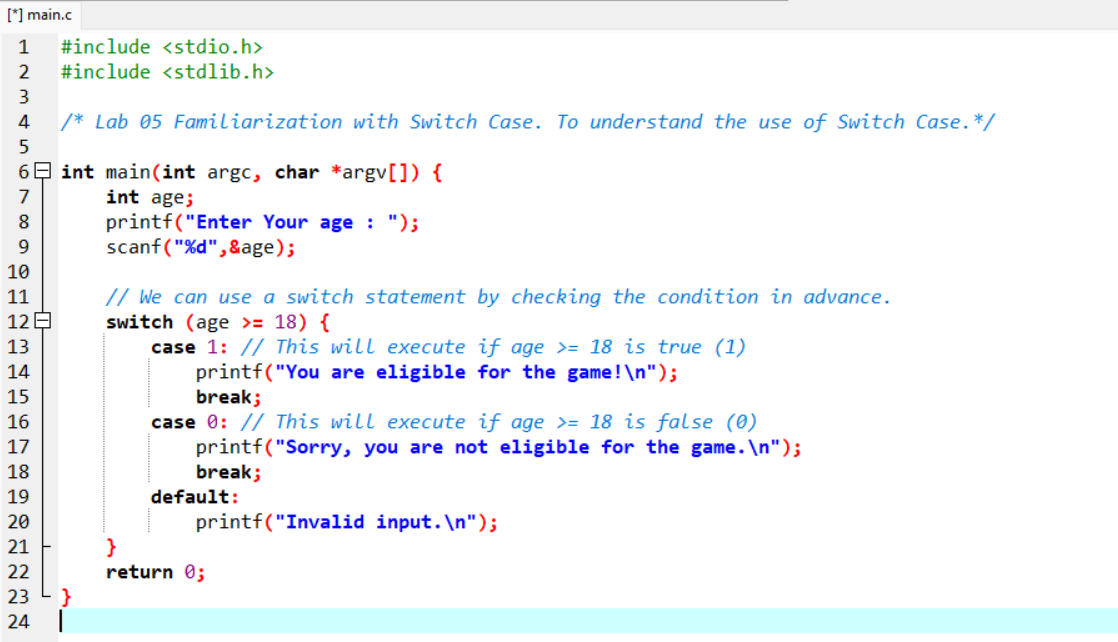
1. **Multiple Cases, Same Action**

* You can group multiple case labels together if they should execute the same code.

1. **Simple Code in Switch Case**

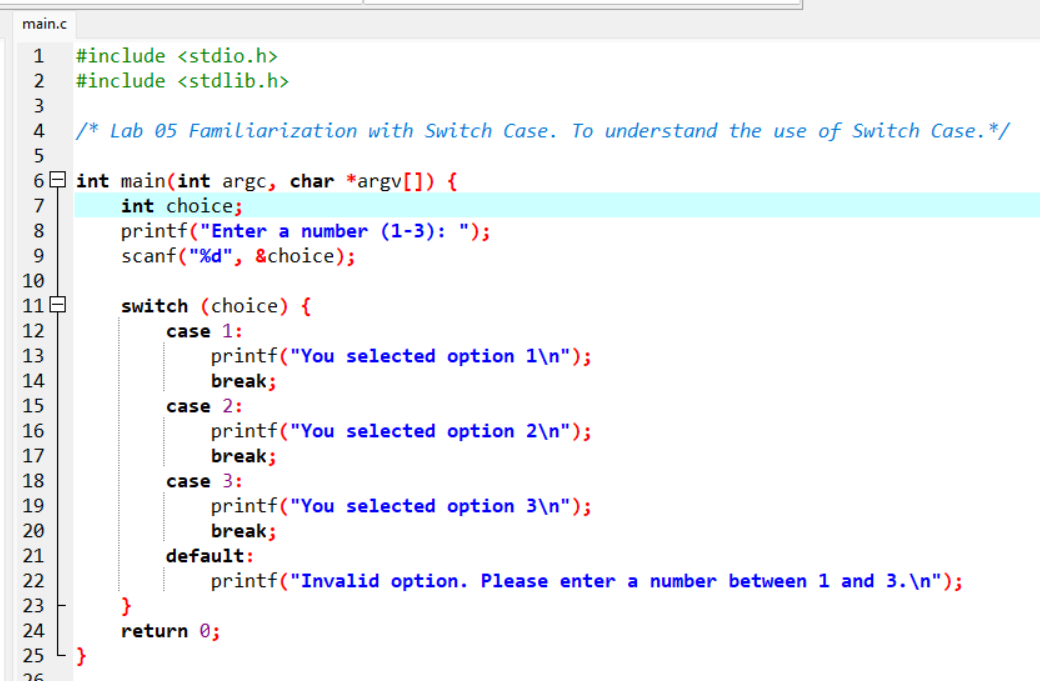
To implement a simple age-checking game using a switch statement, we can use a slightly different approach since switch doesn’t directly support comparisons like if statements do. We can categorize the age by checking if it’s 18 or greater or less than 18.

Here's how we can structure the code to determine eligibility based on user input:

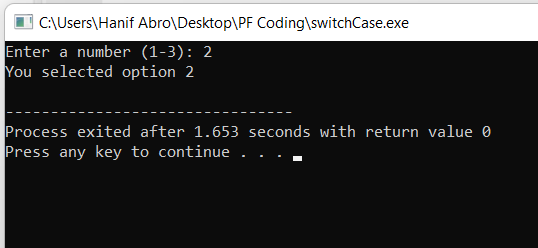


|  |  |
| --- | --- |
| Output less then age 18 | Output Greater then age 18 |

**Example**



**Output**



**Step-by-Step Explanation**

**Explanation of Code**

* **Step 1**: The user inputs a number, stored in choice.
* **Step 2**: The switch statement evaluates choice.
  + **Case 1, 2, or 3**: Executes a specific code block if the user selects 1, 2, or 3.
  + **Default**: Displays an error if the user enters an invalid number.

1. **Break**: This statement exits the switch block, stopping further cases from running. If break is omitted, execution will continue to the next case.
2. **Default**: The default case runs if no other case matches.
3. **Common Errors and Solutions**

* **Error 1: Missing break Statement**  
  Omitting break causes “fall-through,” where subsequent cases execute unintentionally.  
  **Solution**: Add break; at the end of each case block if you don’t want fall-through.
* **Error 2: Using Variables in Cases**  
  Only constants can be used in cases.  
  **Solution**: Ensure case values are constants (like 1, 'A'), not variables.
* **Error 3: Misplaced default Case**  
  The default statement doesn’t need to be at the end, but it’s conventionally placed there.  
  **Solution**: Place default at the end for readability.
* **Error 4: Non-Matching Data Types**  
  Using incompatible data types in the switch expression and cases leads to unexpected results.  
  **Solution**: Ensure all case constants and the switch expression are of the same data type

1. **Key Points Recap**
2. Always use break unless you need to execute multiple cases.
3. The switch expression must be integer or character data type.
4. The default case is optional but useful for handling unexpected values.
5. Use switch when you have many conditions based on one variable—it simplifies your code.

## Observation

|  |
| --- |
|  |

## Lab Task

1. Write a program to input three integers and find the largest among them using nested if-else statements.
2. Write a program that inputs a student's marks and prints the grade based on the following criteria:
   * Marks >= 90: Grade A
   * Marks >= 80: Grade B
   * Marks >= 70: Grade C
   * Marks >= 60: Grade D
   * Marks < 60: Grade F

Use cascaded if-else statements.

3. Write a program using switch to perform basic arithmetic operations (+, -, \*, /) based on a user-selected option.

## Home Tasks

1. Write a C program that asks the user for their age and uses an if-else statement to check if they are eligible to vote (18 years or older). Print a message indicating eligibility.
2. Write a program that takes a character input from the user and uses a switch-case to check if the letter is a vowel (a, e, i, o, u). If it’s a vowel, print "Vowel"; otherwise, print "Consonant."
3. Write a program that asks the user to choose a conversion type:

1: Kilometers to Miles

2: Meters to Feet

3: Centimeters to Inches  
Based on the user’s choice, use a switch-case to perform the selected conversion and display the result.

**Lab Assignment Submission Instructions**

* 1. Screenshot your code and output and paste it into a Word file.
  2. Copy paste your code into that word file according to the Question number.
  3. Number your Word document as question number e.g.Q1.sb, Q2.sb, etc. (Q is in upper case)
  4. Create a new folder with your Name and ID e.g. Naveed Ahmed (61452)\_Lab 04 Assign.
  5. Copy all the Word document and source files into this folder.
  6. Right-click on the folder you created and create a zip file by selecting the option
  7. “Send to” and selecting “Compressed (zipped) folder” [for windows].
  8. “Create Archive” and change the option to “.zip” instead of “ .tar.gz” and click on “Create”. [for linux]
  9. Now make sure a zip file is created with your name and id e.g. Naveed Ahmed

(61452).zip.

* 1. Submit Assignment on LMS.

*Note:*

*Plagiarism is strictly prohibited. Any instance of plagiarism will result in a deduction of 100 points. Ensure that all work submitted is original.*

## Discussion and Analysis of Results

|  |
| --- |
|  |

## Conclusion

|  |
| --- |
|  |

**itch Case**

**Lab 05**

# **Loops in C Programming**

## Objective:

To understand and implement different loop constructs in C (while, do while, and for loops) and to identify suitable problems for each loop type.

## Required Equipment / tools:

Dev C++

## Introduction:

Loops are fundamental constructs in programming that enable the execution of a block of code multiple times based on a specified condition. This allows developers to automate repetitive tasks efficiently, thus enhancing productivity and reducing code redundancy. In C programming, there are three primary types of loops: for, while, and do while. Each loop has its specific use cases and advantages.

## Procedure

**While Loop**

* Repeats a block of code as long as a condition is true.

**Syntax:**

**while (condition) {**

**// Code block**

**}**

* **Use Case: When the number of iterations is unknown and depends on a condition, such as user input.**

**Do While Loop**

* Similar to the while loop but guarantees that the code block is executed at least once.

**Syntax:**

**do {**

**// Code block**

**} while (condition);**

* **Use Case:** When the action must be performed at least once, such as menu selections.

**For Loop**

* Used when the number of iterations is known beforehand.

**Syntax:**

**for (initialization; condition; increment) {**

**// Code block**

**}**

* **Use Case**: Ideal for iterating over arrays or fixed ranges.

**Implementing One By One:**

**While Loop**

**Problem:  
Count down from 5 to 1.**

**Code:**

**#include <stdio.h>**

**int main() {**

**int i = 5;**

**while (i > 0) { // Loop as long as i is greater than 0**

**printf("%d\n", i); // Print the current value of i**

**i--; // Decrement the counter**

**}**

**return 0;**

**}**

**Output:**

**5**

**4**

**3**

**2**

**1**

**Do While Loop**

**Problem:  
Print "Hello" three times.**

**Code:**

**#include <stdio.h>**

**int main() {**

**int count = 1;**

**do {**

**printf("Hello\n"); // Print the message**

**count++; // Increment the counter**

**} while (count <= 3); // Loop until count exceeds 3**

**return 0;**

**}**

**Output:**

**Hello**

**Hello**

**Hello**

**For Loop**

**Problem:**

**Print the first 5 multiples of 3.**

**Code:**

**#include <stdio.h>**

**int main() {**

**// For loop to print the first 5 multiples of 3**

**for (int i = 1; i <= 5; i++) {**

**printf("%d\n", i \* 3); // Print the multiple of 3**

**}**

**return 0;**

**}**

**Output:**

**3**

**6**

**9**

**12**

**15**

## Observation

|  |
| --- |
|  |

## Class Tasks Task 01: Sum of Integers Using While Loop

Write a program that prompts the user to enter a positive integer and calculates the sum of all integers from 1 to that number using a while loop.

**Task 02:**

**User Input Validation Using Do While Loop**

Create a program that continuously asks the user to enter a number until they enter a negative number. Use a do while loop.

**Task 03:**

**Printing Even Numbers Using For Loop**

Develop a program that prints the first 10 even numbers (0 to 18) using a for loop.

**Lab Assignment Submission Instructions**

1. Screenshot your code and output and paste it into a Word file.
2. Copy paste your code into that word file according to Question number.
3. Number your Word document as question number e.g.Q1.sb, Q2.sb, etc. (Q is in upper case)
4. Create a new folder with your Name and ID e.g. Naveed Ahmed (61452)\_Lab 04 Assign.
5. Copy all the Word document and source files into this folder.
6. Right-click on the folder you created and create a zip file by selecting the option
7. “Send to” and selecting “Compressed (zipped) folder” [for windows].
8. “Create Archive” and change option to “.zip” instead of “ .tar.gz” and click on “Create”. [for linux]
9. Now make sure a zip file is created with your name and id e.g. Naveed Ahmed

(61452).zip.

1. Submit task on LMS.

*Note:*

*Plagiarism is strictly prohibited. Any instance of plagiarism will result in a deduction of 100 points. Ensure that all work submitted is original.*

## Home Tasks

Write a program that takes a series of integer inputs from the user and performs the following:

* Use a while loop to keep accepting input until the user enters a zero (0) to stop.
* Use a do while loop to display each entered number until zero is encountered.
* Use a for loop to display the total count of numbers entered, excluding zero**.**

**Lab Assignment Submission Instructions**

1. Screenshot your code and output and paste it into a Word file.
2. Copy paste your code into that word file according to the Question number.
3. Number your Word document as question number e.g.Q1.sb, Q2.sb, etc. (Q is in upper case)
4. Create a new folder with your Name and ID e.g. Naveed Ahmed (61452)\_Lab 04 Assign.
5. Copy all the Word document and source files into this folder.
6. Right-click on the folder you created and create a zip file by selecting the option
7. “Send to” and selecting “Compressed (zipped) folder” [for windows].
8. “Create Archive” and change the option to “.zip” instead of “ .tar.gz” and click on “Create”. [for linux]
9. Now make sure a zip file is created with your name and id e.g. Naveed Ahmed

(61452).zip.

1. Submit Assignment on LMS.

*Note:*

*Plagiarism is strictly prohibited. Any instance of plagiarism will result in a deduction of 100 points. Ensure that all work submitted is original.*

## Discussion and Analysis of Results

|  |
| --- |
|  |

## Conclusion

|  |
| --- |
|  |

**Lab 06**

## 

# **Nested Loops in C Programming**

## Objective:

To learn how to use nested loop constructs including:

* Nested while loop
* Nested do-while loop
* Nested for loop

## Required Equipment / tools:

Dev C++

## Introduction

### 1.Nested Loop

A loop inside another loop is called a nested loop. The number of loops depends on the complexity of a problem. Suppose, a loop, outer loop, running n number of times consists of another loop inside it, inner loop, running m number of times. Then, for each execution of the outer loop from 1...n, the inner loop runs maximum of m times.

### 1.1 Reasons to use nested loops in programming

*Nested loops are a fundamental programming technique that allow developers to efficiently perform repetitive tasks. They are used when one loop is placed inside another loop, allowing a block of code*

### 1.2 Use of nested loops in C

to be executed multiple times under specific conditions.

* When you need to do repeating operations inside of another loop, nested loops in C are helpful.
* They frequently handle matrices or multidimensional arrays.
* For iterating through intricate data structures like nested lists or trees, nested loops can also be used.
* Nesting loops are in handy when you need to conduct combinations or permutations of items.
* They are necessary for tackling issues like searching, sorting, or graph traversal that call for several iterations.
* When you need to compare elements from one loop with elements from another loop, use the nested loop in C programming to create more complex reasoning.

1.3 Nested for loop in C

* The concept of the nested for loop in C programming language provides a versatile approach to achieve complex iterations and effectively execute repetitive tasks.
* In C, nested for loops make it easy to handle nested iterations, making jobs like browsing multidimensional arrays more simpler.
* This hierarchy enhances the readability, maintainability, and organization of the code.
* To tackle complex programming problems and investigate new avenues in software development, mastery is necessary.

## Procedure

|  |  |
| --- | --- |
| **Syntax** | **Flowchart** |
| for (initialization; condition; increment/decrement)  {  statement(s);  for (initialization; condition; increment/decrement)  {  statement(s);  ... ... ...  }  ... ... ...  } |  |

**Program: 1**

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

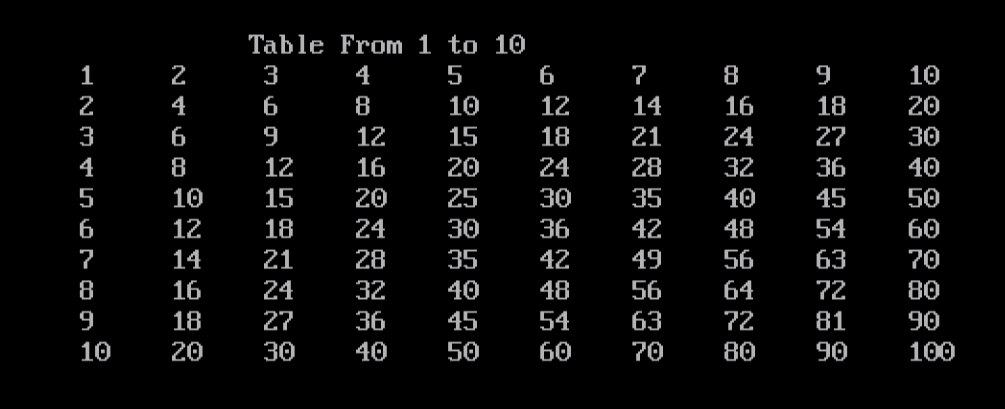
**Output:**

|  |
| --- |
| 1  1 2  1 2 3  1 2 3 4  1 2 3 4 5 |

**Program: 2**

|  |
| --- |
| /\* Program to display multiplication table from 1 to 10.\*/  #include<stdio.h>  #include<conio.h>  void main()  {  int i, j;  clrscr();  printf("\n\t\tTable From 1 to 10");  for(i=1; i<=10; i++)  {  for(j=1; j<=10; j++)  {  gotoxy(6\*i, j+2);  printf("%d\n",i\*j);  }  }  getch();  } |

**Output:**



**Program: 3**

|  |
| --- |
| Find Sum of series  #include<stdio.h>  #include<conio.h>  void main()  {  int x, n, i, j, p=1, s=0, f=1;  clrscr();  printf("Enter value of x: ");  scanf("%d",&x);  printf("Enter value of n: ");  scanf("%d",&n);  for(i=0;i<=n;i++)  {  for(j=1;j<=i;j++)  p=p\*x;//find power  for(j=1;j<=i;j++)  f=f\*j;//find factorial  s=s+p/f;  p=f=1;  }  printf("Sum of series is: %d",s);  getch();  } |

**Output:**

|  |
| --- |
| Enter value of x: 10  Enter value of n: 2  Sum of series is: 61 |

Nested while loop in c

The nested while loop in C programming offers immense control and flexibility when solving multifaceted problems by allowing a programmer to place one loop inside another.

Create complicated patterns and analyze data more easily with this framework.

As a programmer, mastering the nested while loop in C will greatly expand the ability to handle challenging situations and develop intricate solutions.

Dive into the world of C programming, and the user will find the nested while loop to be an invaluable tool in optimizing and streamlining the code to achieve extraordinary results.

|  |  |
| --- | --- |
| **Syntax** | **Flowchart** |
| while (condition1)  {  statement(s); while (condition2)  {  statement(s);  ... ... ...  }  ... ... ...  } |  |

Program: 4

|  |
| --- |
| #include <stdio.h>  int main() {  int i = 1, j = 1;  while (i <= 3)  {  printf("Outer loop iteration %d\n", i);  while (j <= 3)  {  printf(" Inner loop iteration %d\n", j);  j++;  }  j = 1; // reset j to 1 for the next iteration of the outer loop  i++;  }  return 0;  } |

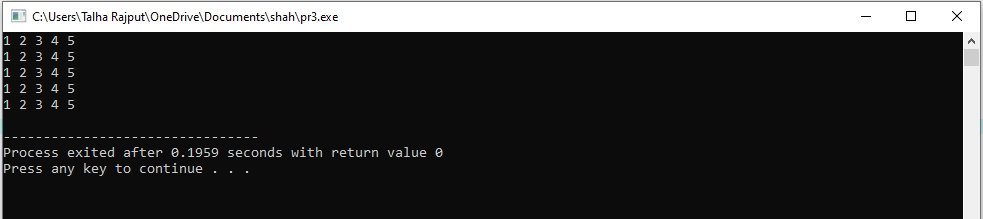
**Output:**

|  |
| --- |
| Outer loop iteration 1  Inner loop iteration 1  Inner loop iteration 2  Inner loop iteration 3  Outer loop iteration 2  Inner loop iteration 1  Inner loop iteration 2  Inner loop iteration 3  Outer loop iteration 3  Inner loop iteration 1  Inner loop iteration 2  Inner loop iteration 3 |

**Program: 5**

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

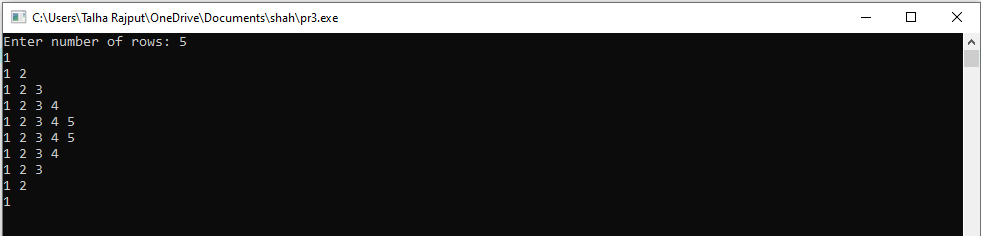
Output:



**Program: 6**

|  |
| --- |
| /\* Pattern Series using nested loop\*/  #include<stdio.h>  #include<conio.h>  void main()  {  int i,j,n;  clrscr();  printf("Enter number of rows: ");  scanf("%d",&n);  for(i=1;i<=n;i++)  {  for(j=1;j<=i;j++)  {  printf("%d ",j);  }  printf("\n");  }  for(i=n;i>=1;i--)  {  for(j=1;j<=i;j++)  {  printf("%d ",j);  }  printf("\n");  }  getch();  } |

**Output:**



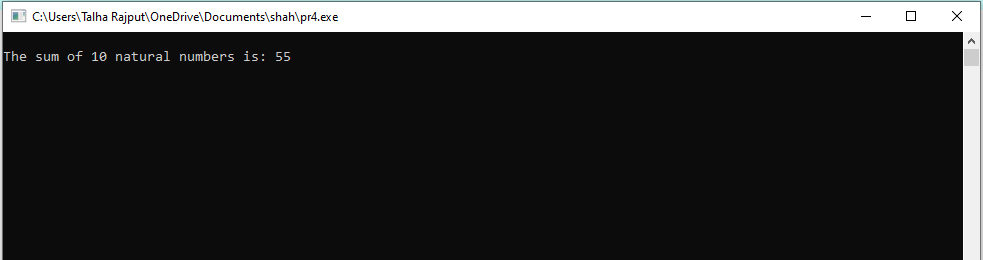
* 1. **Nested do...while loop in C**
* The nested do-while loop in C programming serves as a powerful tool that grants coders the ability to efficiently execute repeated actions based on specific conditions.
* By executing commands while requirements are met, nested while loops improve precision as well as control in complex computations.
* As the developer dives deeper into the world of C programming, mastering the art of nested do-while loops will undoubtedly enhance the ability to create dynamic and effective solutions.

|  |  |
| --- | --- |
| **Syntax** | **Flowchart** |
| do  {  statement(s); do  {  statement(s);  ... ... ...  }  while (condition2);  ... ... ...  }  while (condition1); |  |

**Program: 7**

|  |
| --- |
| /\*Calculate sum of 10 natural numbers\*/  #include<stdio.h>  #include<conio.h>  main()  {  int i=1,sum=0;  do  {  sum += i;  i++;  }  while(i<=10);  printf("\nThe sum of 10 natural numbers is: %d",sum);  getch();  } |

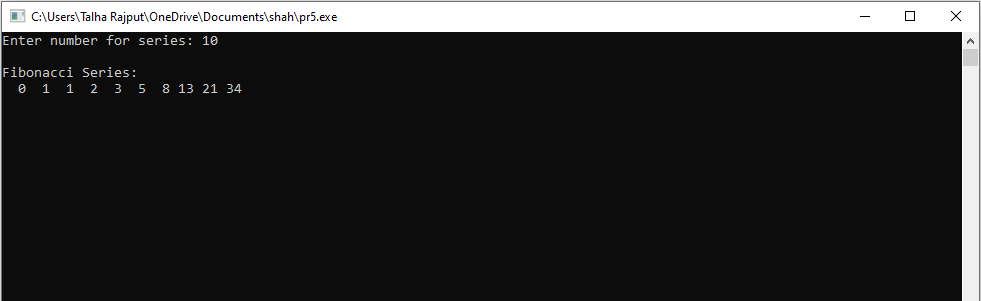
**Output:**



**Program: 8**

|  |
| --- |
| /\*Program to print Fibonacci Series 0,1,1,2,3,5,8,13,...\*/  #include<stdio.h>  #include<conio.h>  main()  {  int n,n1=0,n2=1,n3,i=3;  printf("Enter number for series: ");  scanf("%d",&n);  printf("\nFibonacci Series: ");  printf("\n%3d%3d",n1,n2);  do  {  n3 = n1+n2;  printf("%3d",n3);  n1 = n2;  n2 = n3;  i++;  }  while(i<=n);  getch();  } |

**Output:**

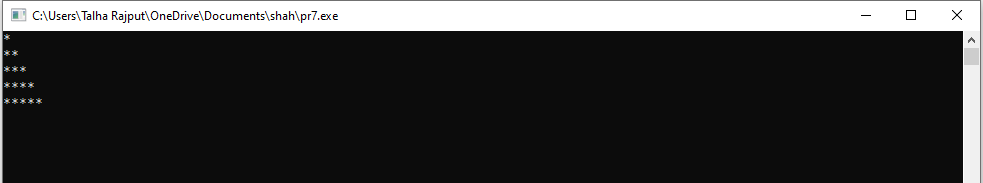


## Observation

|  |
| --- |
|  |

## Lab Task

1. Write a program to display the following pattern.



1. Write a program to check whether a number is Palindrome or not.
2. Write a program to generate Fibonacci series.
3. Write programs to display each of the following patterns

\* \* \* \* \* \* \* \* \*

\* \* \* \* \* \* \*

\* \* \* \* \*

\* \* \*

\*

**Lab Assignment Submission Instructions**

1. Screenshot your code and output and paste it into a Word file.
2. Copy paste your code into that word file according to Question number.
3. Number your Word document as question number e.g.Q1.sb, Q2.sb, etc. (Q is in upper case)
4. Create a new folder with your Name and ID e.g. Naveed Ahmed (61452)\_Lab 04 Assign.
5. Copy all the Word document and source files into this folder.
6. Right-click on the folder you created and create a zip file by selecting the option
7. “Send to” and selecting “Compressed (zipped) folder” [for windows].
8. “Create Archive” and change option to “.zip” instead of “ .tar.gz” and click on “Create”. [for linux]
9. Now make sure a zip file is created with your name and id e.g. Naveed Ahmed

(61452).zip.

1. Submit lab task on LMS.

*Note:*

*Plagiarism is strictly prohibited. Any instance of plagiarism will result in a deduction of 100 points. Ensure that all work submitted is original.*

## Home Tasks

1. Find out the sum of series 12 + 22 + …. + n2
2. If a four-digit number is input through the keyboard, write a program to obtain the sum of the first and last digit of this number.
3. Write a program to find GCD (greatest common divisor or HCF) and LCM (least common multiple) of two numbers.

**Lab Assignment Submission Instructions**

1. Screenshot your code and output and paste it into a Word file.
2. Copy paste your code into that word file according to Question number.
3. Number your Word document as question number e.g.Q1.sb, Q2.sb, etc. (Q is in upper case)
4. Create a new folder with your Name and ID e.g. Naveed Ahmed (61452)\_Lab 04 Assign.
5. Copy all the Word document and source files into this folder.
6. Right-click on the folder you created and create a zip file by selecting the option
7. “Send to” and selecting “Compressed (zipped) folder” [for windows].
8. “Create Archive” and change option to “.zip” instead of “ .tar.gz” and click on “Create”. [for linux]
9. Now make sure a zip file is created with your name and id e.g. Naveed Ahmed

(61452).zip.

1. Submit Assignment on LMS.

*Note:*

*Plagiarism is strictly prohibited. Any instance of plagiarism will result in a deduction of 100 points. Ensure that all work submitted is original.*

## Discussion and Analysis of Results

|  |
| --- |
|  |

## Conclusion

|  |
| --- |
|  |

**Lab 07**

# **Open Ended Lab**

|  |  |
| --- | --- |
| **Assessment of Open Ended Lab** | |
| **Mapped GAs** | **Bloom Taxonomy** |
| GA 5 | A2 |
| GA 2 | C3 |

## Title:

## Objective:

## Motivation:

## Concept:

## Problem Statement:

## Design / Ways & Means:

Students are required to follow the design constraints set out for the implementation of OEL:

* Introduction and Requirements
* Data Structure Selection
* Basic Implementation
* Performance Testing and Analysis
* Optimization and Advanced Features
* Extensions and Creativity

## Analysis & Reporting /Answer:

## Lab Activity:

## Deliverables:

* Background/Theory
* Procedure / Methodology
* Data Collection (If required)
* Flowchart / Block diagram
* Analysis
* Results
* Discussion on Results
* Concluding Remarks
* Reference

**Lab 08**

# **Arrays in C**

## Objective:

To understand and implement the concept of arrays in the C programming language. This lab will cover the declaration, initialization, and manipulation of arrays in C. And the implementation of algorithms using 1D and 2D Arrays.

## Required Equipment / tools:

VS Code/Dev C++/Turbo C

## Introduction:

In C programming, an array is a collection of variables of the same data type, stored in contiguous memory locations. Arrays allow efficient data storage and retrieval using a single variable name, with individual elements accessible by their index position. Arrays are useful in many programming

applications, including data processing, sorting algorithms, and search operations.

* + 1. **Key Concepts:**
* **Declaration of Arrays:** Syntax for declaring an array with a specified data type.
* **Array Indexing:** Accessing elements using their index (0-based in C).
* **Array Initialization:** Assigning values to array elements either individually or during declaration.
* **Array Manipulation:** Performing operations on arrays, like finding sums or

searching for elements.

## Procedure

* + 1. **One-Dimensional Array**

A one-dimensional array is a linear collection of elements stored in consecutive memory locations. Elements in a 1D array can be accessed using a single index.

**Step 1: Declaring and Initializing an Array**

1. **Declare an Array:** Define an integer array with a fixed size. This allocates contiguous memory to hold the elements.

**int numbers[5];**

Here, ‘int’ is the data type and ‘numbers’ is an array that can hold 5 integer elements.

1. **Initialize the Array:**

* Assign values to each element individually.
* Or, initialize the array during declaration.

// Individual Initialization

numbers[0] = 10;

numbers[1] = 20;

numbers[2] = 30;

numbers[3] = 40;

numbers[4] = 50;

// Declaration with Initialization

int numbers[] = {10, 20, 30, 40, 50};

**Step 2: Accessing Array Elements**

1. **Accessing Array Elements Directly by Their Index Number**

This method directly accesses specific elements in the array by their index number. You manually specify each index number to print the elements of an Array**.**

|  |
| --- |
| #include <stdio.h>  int main() {  int numbers[] = {10, 20, 30, 40, 50};  // Accessing elements by index  printf("Element at index 0: %d\n", numbers[0]);  printf("Element at index 1: %d\n", numbers[1]);  printf("Element at index 2: %d\n", numbers[2]);  printf("Element at index 3: %d\n", numbers[3]);  printf("Element at index 4: %d\n", numbers[4]);  return 0;  } |

1. **Accessing Array Elements Using a for Loop**

Using a loop is an efficient way to access each element sequentially, especially for

larger arrays. Here, a for loop iterates over each index to print the array elements.

|  |
| --- |
| #include <stdio.h>  int main() {  int numbers[] = {10, 20, 30, 40, 50};  int size = sizeof(numbers) / sizeof(numbers[0]);  for (int i = 0; i < size; i++) {  printf("Element at index %d: %d\n", i, numbers[i]);  }  return 0;  } |

1. **Accessing Array Elements Using a while Loop**

This approach uses a while loop instead of a for loop to access each element. It’s an alternative way to iterate through an array, often useful when the loop’s end condition

is dynamic.

|  |
| --- |
| #include <stdio.h>  int main() {  int numbers[] = {10, 20, 30, 40, 50};  int size = sizeof(numbers) / sizeof(numbers[0]);  int i = 0;  while (i < size) {  printf("Element at index %d: %d\n", i, numbers[i]);  i++;  }  return 0;  } |

* + 1. **Two-Dimensional Array**

A two-dimensional array (matrix) is an array of arrays, organized in rows and columns. Each element in a 2D array is accessed using two indices: one for the row and one for the

column.

**Step 1: Declaring and Initializing a 2D Array**

1. **Declare a 2D Array:** Define a 2D integer array specifying rows and columns.

int matrix[2][3];

Here, matrix is a 2D array with 2 rows and 3 columns.

1. **Initialize the Array:** Assign values row-by-row or initialize all elements during

declaration.

// Row-by-Row Initialization

matrix[0][0] = 1; matrix[0][1] = 2; matrix[0][2] = 3;

matrix[1][0] = 4; matrix[1][1] = 5; matrix[1][2] = 6;

// Declaration with Initialization

int matrix[2][3] = {{1, 2, 3}, {4, 5, 6}};

**Step 2: Accessing and Displaying 2D Array Elements**

1. **Accessing Elements Directly using Index Number**

You can access individual elements in a 2D array by specifying the row and column indices directly. This method is useful when you only need to work with specific

elements.

|  |
| --- |
| #include <stdio.h>  int main() {  int matrix[2][3] = {{1, 2, 3}, {4, 5, 6}};  // Access specific elements directly  printf("Element at [0][0]: %d\n", matrix[0][0]); // Output: 1  printf("Element at [0][1]: %d\n", matrix[0][1]); // Output: 2  printf("Element at [0][2]: %d\n", matrix[0][2]); // Output: 3  printf("Element at [1][0]: %d\n", matrix[1][0]); // Output: 4  printf("Element at [1][1]: %d\n", matrix[1][1]); // Output: 5  printf("Element at [1][2]: %d\n", matrix[1][2]); // Output: 6  return 0;  } |

1. **Using Nested Loops to Access Elements:**

Use nested for loops, where the outer loop iterates over rows and the inner loop iterates over columns, to access and display

each element.

|  |
| --- |
| #include <stdio.h>  int main() {  int matrix[2][3] = {{1, 2, 3}, {4, 5, 6}};  for (int i = 0; i < 2; i++) {  for (int j = 0; j < 3; j++) {  printf("Element at [%d][%d]: %d\n", i, j, matrix[i][j]);  }}  return 0;  } |

* + 1. **Sorting Algorithm by Using Arrays**

Arrays have many properties applicable on them. Hence we apply these on array using algorithms which may be categorized as follows.

1. **Searching**

A searching algorithm is a method used to locate a specific value or address within a data structure. There are two primary types of searching:

1. **By Address or Location**: This approach involves finding an element based on its memory address or index in a data structure, such as an array.
2. **By Value**: This method entails searching for an element by its value, comparing it against the elements in a data structure until a match is found or all elements are checked.

**Example**

|  |
| --- |
| **C program that prompts the user for a number, searches for it in an array, and returns its location (index). If the number is not found, it notifies the user accordingly.** |
| #include <stdio.h>  int main() {  int data[] = {5, 10, 15, 20, 25, 30, 35, 40}; // Sample data array  int size = sizeof(data) / sizeof(data[0]);  int num, found = -1;  // Prompt user for input  printf("Enter a number to search for: ");  scanf("%d", &num);  // Search for the number in the array  for (int i = 0; i < size; i++) {  if (data[i] == num) {  found = i;  break;  }  }  // Output the result  if (found != -1) {  printf("Number %d found at index %d.\n", num, found);  } else {  printf("Number %d not found in the data.\n", num);  }  return 0;  } |

1. **Input**

In the context of data structures and algorithms, **input** refers to the method of organizing data for processing. Two common types of input organization are

1. **FIFO (First In, First Out)**: This is a method where the first element added to the data structure is the first one to be removed. It operates like a queue, where items are processed in the order they arrive.
2. **FILO or LIFO (First In Last Out)**: This method allows the last element added to the data structure to be the first one removed. It operates like a stack, where the most recently added item is the first to be processed.

|  |
| --- |
| **This program takes input at each iteration, updates the array accordingly, and displays the state of the array after each input.** |
| #include <stdio.h>  int main(void) {  int arr[5] = {0, 0, 0, 0, 0};    for (int a1 = 0; a1 < 5; a1++) {  // Shift existing elements to the right  for (int a2 = 4; a2 > 0; a2--) {  arr[a2] = arr[a2 - 1];  }  // Take new input and place it at the beginning of the array  printf("Enter Value for FILO: ");  scanf("%d", &arr[0]);  // Display the current state of the array  printf("Array after input %d:\n", a1 + 1);  for (int a3 = 0; a3 < 5; a3++) {  printf("%d ", arr[a3]);  }  printf("\n\n");  }  // Final display of array  printf("Final array state:\n");  for (int d = 0; d < 5; d++) {  printf("%d ", arr[d]);  }    return 0;  } |

1. **Bubble Sort Algorithm**

Bubble Sort is a simple sorting algorithm in C that repeatedly steps through a list, compares adjacent elements, and swaps them if they are in the wrong order. This process is repeated until the list is sorted in ascending or descending order.

1. **Ascending Order**: Arranges data from smallest to largest (e.g., 1, 2, 3) or alphabetically from A to Z.
2. **Descending Order**: Arranges data from largest to smallest (e.g., 9, 8, 7) or alphabetically from Z to A.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | 3 | 8 | 4 | 6 |

**Initial Array**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | 3 | 8 | 4 | 6 |

**Step 01**

Compare 1st and 2nd number and if the 2nd number is smallest the it will swap.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 3 | 5 | 8 | 4 | 6 |

**Step 02**

After swapping, the array above will reflect the final arrangement of elements. Now, it will move forward to the next array element and compare it with the third elements. In this case, the third element is greater, so no swap will occur.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 3 | 5 | 8 | 4 | 6 |

**Step 03**

Now it moves forward and starts comparing with the 4th element. Since 4 is smaller than 8, they will be swapped.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 3 | 5 | 4 | 8 | 6 |

**Step 4**

After swapping, it moves forward and starts comparing the next element. Now, 6 is smaller than 8, so they are swapped.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 3 | 5 | 4 | 6 | 8 |

**Step 5**

This step will be repeated until no swaps are needed, indicating that the entire array is sorted. Now, we will compare the elements of the starting array again. In this case, 5 is greater than 3, so no swap will occur.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 3 | 5 | 4 | 6 | 8 |

**Step 6**

It will now move forward and start comparing the 2nd element with the 3rd element. In this case, the 3rd element (4) is smaller than the 2nd element (5), so they will be swapped.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 3 | 4 | 5 | 6 | 8 |

**Step 7**

The array above is now the final sorted array. Comparisons will continue until the end of the array, but the remaining elements are already sorted accordingly.

|  |
| --- |
| **This program ask the user to enter the 10 elements and print the updated array elements by arranging in ascending and descending order.** |
| #include <stdio.h>  int main() {  int array[10], asc[10], desc[10];  int temp, i, j;  printf("Enter 10 integers:\n");  for(i = 9; i >= 0; i--) {  printf("Enter value %d: ", 10 - i);  scanf("%d", &array[i]);  }  for(i = 0; i < 10; i++) {  asc[i] = array[i];  desc[i] = array[i];  }  for(i = 0; i < 9; i++) {  for(j = 0; j < 9 - i; j++) {  if(asc[j] > asc[j + 1]) {  // Swap the elements  temp = asc[j];  asc[j] = asc[j + 1];  asc[j + 1] = temp;  }  }  }  for(i = 0; i < 9; i++) {  for(j = 0; j < 9 - i; j++) {  if(desc[j] < desc[j + 1]) {  // Swap the elements  temp = desc[j];  desc[j] = desc[j + 1];  desc[j + 1] = temp;  }  }  }  printf("\nOriginal array (FILO order):\n");  for(i = 0; i < 10; i++) {  printf("%d ", array[i]);  }  printf("\n\nArray in ascending order:\n");  for(i = 0; i < 10; i++) {  printf("%d ", asc[i]);  }  printf("\n\nArray in descending order:\n");  for(i = 0; i < 10; i++) {  printf("%d ", desc[i]);  }  printf("\n");  return 0;  } |

## Observation

|  |
| --- |
|  |

## Lab Task

1. Write a C program to store the temperatures recorded over 7 days in an array. Calculate and display the average temperature of the week. Also, find and display the highest and lowest temperatures recorded.
2. Write a C program that takes an array of integers as input and outputs the array in reverse order. The program should work for arrays of any size up to 100 elements.
3. Write a C program that takes an array of integers and its size as inputs, and sorts the array in ascending order using the Bubble Sort algorithm. Display the sorted array in the main function.

**Lab Task Submission Instructions**

1. Screenshot your code and output and paste it into a Word file.
2. Copy paste your code into that word file according to the Question number.
3. Number your Word document as question number e.g.Q1.sb, Q2.sb, etc. (Q is in upper case)
4. Create a new folder with your Name and ID e.g. Naveed Ahmed (61452)\_Lab 04 Assign.
5. Copy all the Word document and source files into this folder.
6. Right-click on the folder you created and create a zip file by selecting the option
7. “Send to” and selecting “Compressed (zipped) folder” [for windows].
8. “Create Archive” and change option to “.zip” instead of “ .tar.gz” and click on “Create”. [for linux]
9. Now make sure a zip file is created with your name and id e.g. Naveed Ahmed

(61452).zip.

1. Submit lab task on LMS.

*Note:*

*Plagiarism is strictly prohibited. Any instance of plagiarism will result in a deduction of 100 points. Ensure that all work submitted is original.*

## Home Tasks

1. Write a program to record the temperatures (in Celsius) for a week (7 days). Store these temperatures in an array and then calculate and display the average temperature for the week.
2. Develop a program that stores the grades of 5 students in an array. After storing the grades, calculate and display the highest grade and the average grade of the students.
3. Write a C program that takes an array of integers and its size as inputs, and sorts the array in descending order using the Bubble Sort algorithm. Display the sorted array in the main function.

**Lab Assignment Submission Instructions**

1. Screenshot your code and output and paste it into a Word file.
2. Copy paste your code into that word file according to Question number.
3. Number your Word document as question number e.g.Q1.sb, Q2.sb, etc. (Q is in upper case)
4. Create a new folder with your Name and ID e.g. Naveed Ahmed (61452)\_Lab 04 Assign.
5. Copy all the Word document and source files into this folder.
6. Right-click on the folder you created and create a zip file by selecting the option
7. “Send to” and selecting “Compressed (zipped) folder” [for windows].
8. “Create Archive” and change option to “.zip” instead of “ .tar.gz” and click on “Create”. [for linux]
9. Now make sure a zip file is created with your name and id e.g. Naveed Ahmed

(61452).zip.

1. Submit Assignment on LMS.

*Note:*

*Plagiarism is strictly prohibited. Any instance of plagiarism will result in a deduction of 100 points. Ensure that all work submitted is original.*

## Discussion and Analysis of Results

|  |
| --- |
|  |

## Conclusion

|  |
| --- |
|  |

**Lab 09**

# **Functions in C**

## Objective:

- What are functions in C language

- Why we use functions

- How to define and use functions

## Required Equipment / tools:

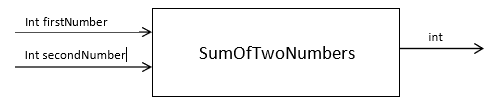
VS Code/Dev C++/Turbo C

## Introduction:

### Function

A function is a group of statements or set of instructions that together perform a specific task. Every C program has at least one function, which is main(). There are two types of functions Built-in functions e.g. printf(), scanf() and User define functions.

Example of user defined functions: Sum of two numbers:



### **Why we use functions**

· The length of a source program can be reduced by using functions at suitable places.

· It is easy to divide and separate a program into parts that improves readability and maintainability.

· A function may be used by many other programs as we use printf() (Reusability).

### **How to use functions**

1. Function Prototype / Declaration

2. Function Definition

3. Function Call

### **Function Prototype / Declaration**

Function prototype or function declaration refers to informing the compiler the signature (the name and parameter details) of a function that will be defined and called later. A function declaration must be done ONCE only and is generally placed before defining the main() function. If a function will not return anything the return data type will be **void** otherwise any data type required. A general form of a C function declaration is as follows.

return\_type function\_name (parameter list);

### **Function Definition**

Function definition refers to the implementation of the “declared” function. The signature will obviously be the same as stated in the function declaration. A general form of a C function definition is as follows.

**Syntax**

**Function Call** While creating a C function, you give a definition of what the function must do. To use a function, you must call that function to perform the defined task.

### **Types** of user defined functions:

1. Function with no arguments and no return values

2. Function with no arguments and one return value

3. Function with arguments and no return values

4. Function with arguments and return value

## Procedure

**Example 01**

**Function with no arguments and no return values**

/\*

program to calculate the area of a square

function with no argument and no return value

\*/

#include <stdio.h>

void area();

int main() {

area(); //function call

return 0;

}

void area(){

int side, square\_area;

printf("Enter the side of the square : ");

scanf("%d",&side);

square\_area = side\*side;

printf("Area of the square with side %d is %d",side, square\_area);

}

**Example 02**

**Function with no arguments and one return value**

/\*

program to calculate the area of a square

function with no argument and one(1) return value

\*/

#include <stdio.h>

int area(); //note that we have use int here instead of void

int main() {

int result = 0;

result = area(); //function call

printf("Area of the square is %d",result);

return 0;

}

int area(){

int side, square\_area;

printf("Enter the side of the square : ");

scanf("%d",&side);

square\_area = side\*side;

return square\_area;

}

**Example 03**

**Function with arguments and no return values**

/\*

program to calculate the area of a square

function with one(1) argument and no return value

\*/

#include <stdio.h>

void area(int);

int main() {

int side = 0;

printf("Enter the side of the square : ");

scanf("%d",&side);

area(side); //function call

return 0;

}

void area(int side){

int square\_area = side\*side;

printf("Area of the square is %d",square\_area);

}

**Example 04**

**Function with arguments and return value**

/\*

Function with arguments and return value

\*/

#include <stdio.h>

int sumOfTwoNumbes(int, int); //function declaration

int main() {

int n1, n2, sum = 0;

printf("Enter first Number : ");

scanf("%d",&n1);

printf("Enter second Number : ");

scanf("%d",&n2);

sum = sumOfTwoNumbes(n1, n2); // function call. call by value

printf("Sum of %d and %d is %d",n1,n2,sum);

return 0;

}

//Function definition

/\*Remember that function parameters are local varibles to the

functions and only remain active until the function is alive.\*/

int sumOfTwoNumbes(int firstNumber, int secondNumber){

int sum = 0;

sum = firstNumber + secondNumber;

return sum;

}

**Passing Arrays to Functions:**

We can pass arrays to function as arguments.

#include <stdio.h>

int calculateSum(int arr[], int size) {

int sum = 0;

for (int i = 0; i < size; i++) {

sum += arr[i];

}

return sum;

}

int main() {

int numbers[] = {1, 2, 3, 4, 5};

int size = sizeof(numbers) / sizeof(numbers[0]);

int sum = calculateSum(numbers, size);

printf("The sum of the array elements is: %d\n", sum);

return 0;

}

Note: When passing an array to a function, only the reference to the first element is passed, not the entire array. Hence, modifications to the array in the function affect the original array.

|  |  |
| --- | --- |
| Observation  |  | | --- | |  |  Lab Task Task01(task01.c):  Write a function which takes three numbers input and returns the average as output and use this function in your program.  Task02(task02.c):  Write a function to find the minimum among three numbers.  Tasl 03(task 03.c):  Write a function which takes an array of 7 days temperature record and returns the day with a maximum temperature record. |

## Home Tasks

**Problem01**

Mr. A purchases online groceries from ABC superstore, ABC superstore offers free delivery on orders above Rs. 1000, otherwise they charge Rs. 150 as delivery charges. Write a function to calculate total delivery charges.

**Problem02**

Write functions to find the least significant digit and most significant digits of a three digit number.

**Problem03**

Write a function to accept a 2D array as input which contains the marks of students in multiple subjects. Calculate the total marks for each student and determine who wins the first position.

## Discussion and Analysis of Results

|  |
| --- |
|  |

## Conclusion

|  |
| --- |
|  |

**Lab 10**

# **Recursion in C**

## Objective:

The objective of this lab is to introduce the concept of recursion, specifically focusing on how the function call stack operates during recursive calls. Through this lab, students will gain an understanding of stack operations like rolling (push) and unrolling (pop), and explore potential issues like stack overflow in recursive functions.

## Required Equipment / tools:

VS Code/Dev C++/Turbo C

## Introduction:

In C programming, recursion is a powerful technique that allows functions to call themselves to solve problems by breaking them down into smaller sub-problems. The C language relies on a function call stack to manage these recursive calls. Each time a function is called, a new stack frame is created in memory to store the function's variables and return address. Recursive functions in C keep adding new frames to the stack ("stack rolling") until a base condition is met, after which the stack begins to "unroll," removing frames as each recursive call completes. While recursion is efficient for certain problems, excessive recursion depth in C can lead to stack overflow due to limited stack memory, highlighting the importance of managing recursive calls carefully.

* + 1. **Key Concepts:**
* **Understanding the Function Call Stack:**

Learning how C uses the call stack to manage recursive function calls.

* **Stack Rolling (Push) and Stack Unrolling (Pop):**

Observing how new stack frames are added and removed during recursive calls.

* **Implementing Recursive Functions:**

Writing and testing recursive functions, such as calculating factorials, to see recursion in action.

* **Handling Stack Overflow:**

Recognizing and preventing stack overflow errors caused by excessive recursion depth.

## Procedure

* + 1. **Understanding the Function Call Stack in Recursion**

The function call stack is a memory structure that tracks active function calls in a program. Every time a function is called, a new "stack frame" is added (or **pushed**) to the stack, containing:

* The function’s parameters
* Local variables
* The return address, which tells the program where to go back to after the function call is completed

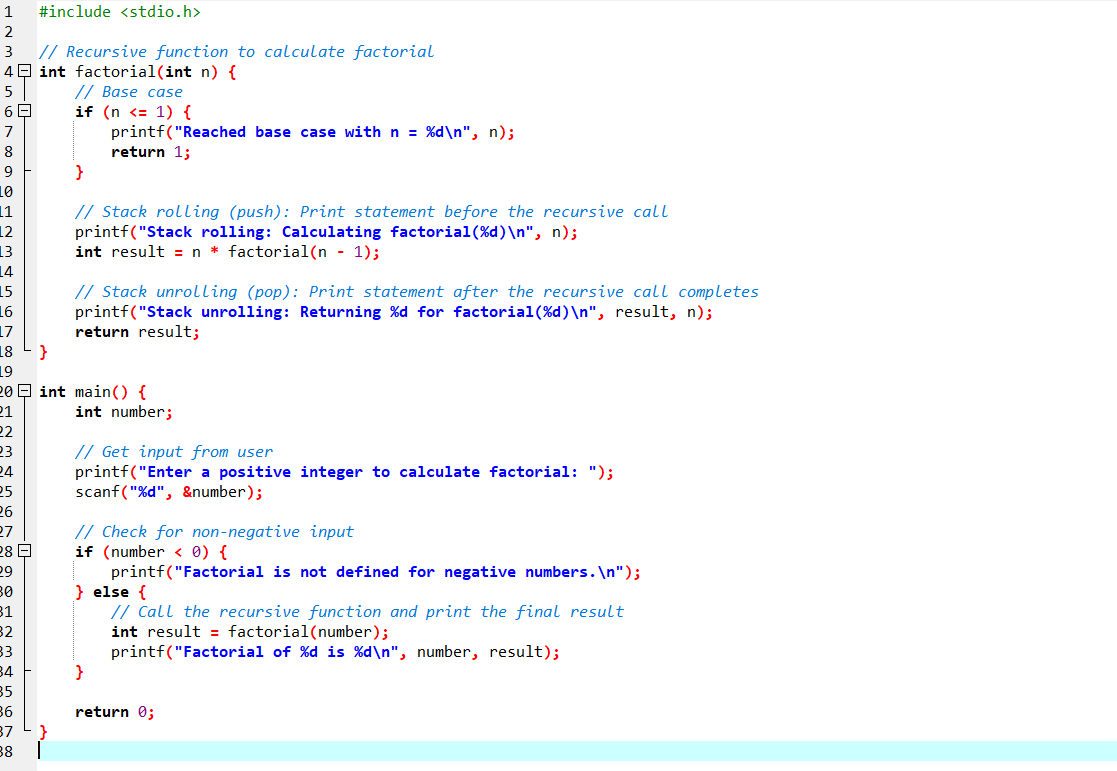
Each function completes by returning its result to the previous function, after which its frame is **popped** from the stack. This structure is crucial for recursion, as each recursive call creates a new frame.

**1.2.2 Writing a Recursive Function – Factorial Calculation**

The factorial of a number n (denoted n!) is a classic example of recursion:

* **Base case**: n=0n = 0n=0 or n=1n = 1n=1, where n!=1n! = 1n!=1
* **Recursive case**: n!=n×(n−1)!n! = n \times (n - 1)!n!=n×(n−1)!

The following code in C to calculate the factorial of a number, including printf statements to observe stack rolling and unrolling:



**Explanation of Code and Concepts**

1. **Stack Rolling (Push)**:

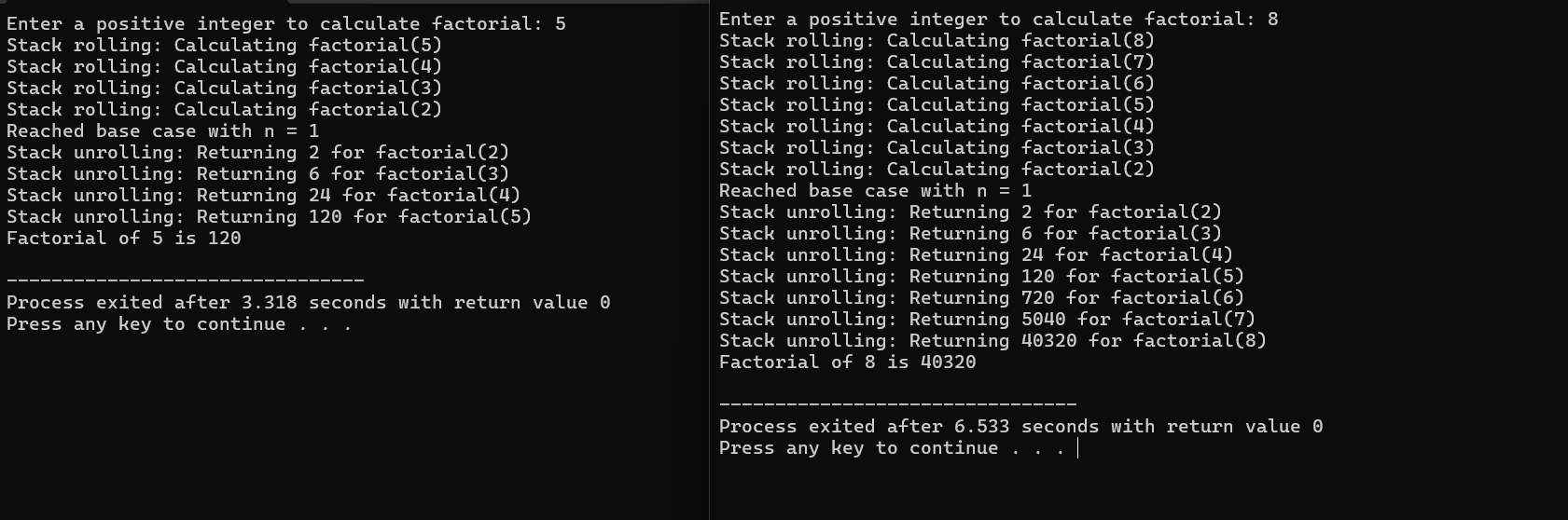
Each call to factorial creates a new frame. As factorial calls itself with decreasing values of n, new frames are added to the stack. These calls continue until the base case is reached (n <= 1), which prevents infinite recursion.

1. **Base Case**:

The base case is when n is 0 or 1, at which point the function returns 1. This stops further recursive calls and begins the process of unrolling the stack.

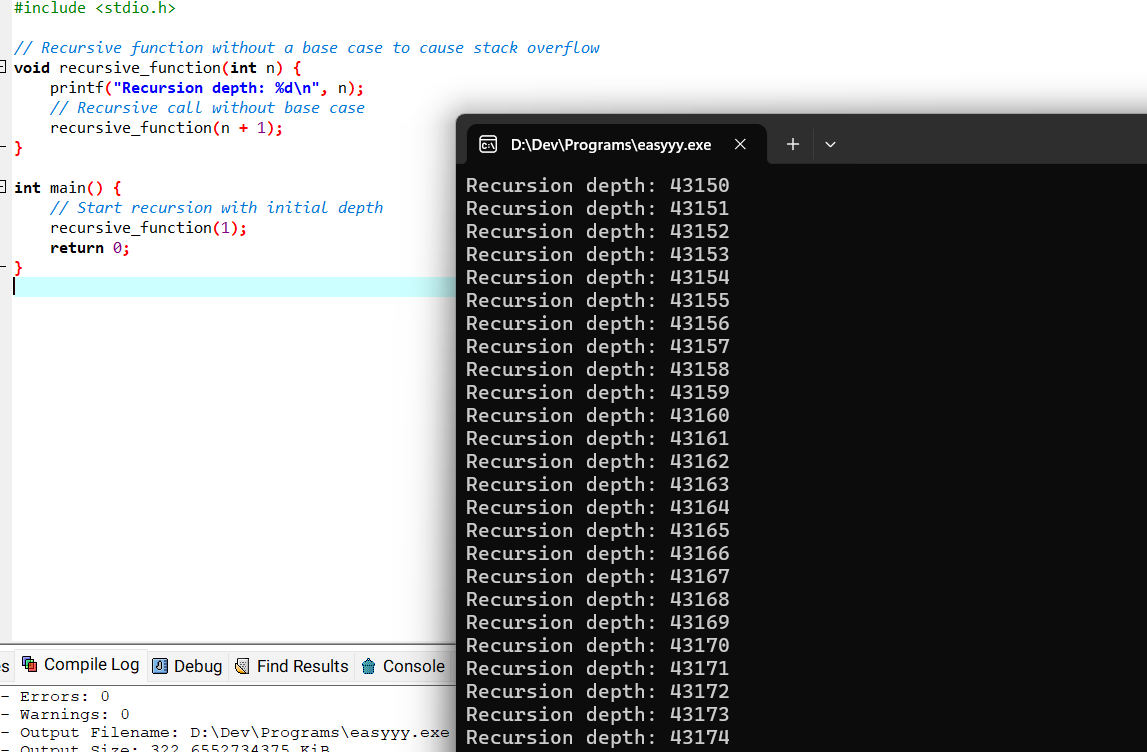
1. **Stack Unrolling (Pop)**

Once the base case returns 1, each function call completes in reverse order. As each call completes, it returns its result to the previous frame, which is then removed from the stack.

**Output:**  


**1.2.3 Stack Overflow in Recursion**

If too many frames are pushed onto the stack without reaching a base case, the program may encounter stack overflow. Each system has a limited stack size, so recursive functions should have a well-defined base case to avoid infinite recursion.

The following code demonstrates stack overflow by deliberately removing the base case in a recursive function. This causes the recursion to continue indefinitely, eventually exhausting the stack memory.  
  


**In this code:**

* The function recursive\_function calls itself with an incremented value of n, but without a base case to stop the recursion.
* Eventually, this continuous recursion will consume all available stack space, causing a stack overflow. The program will terminate with an error, such as a "Segmentation fault" or "Stack overflow" message, depending on your system.

**1.2.3 Stack Underflow in Recursion**

While stack underflow is rare in C, it would theoretically occur if you tried to pop an element from an empty stack. In C, the function call stack is automatically managed, so explicit underflow handling is usually unnecessary. However, stack underflow is a more common issue in low-level stack management (e.g., with manual data structures) rather than function call management in C.

## Observation

|  |
| --- |
|  |

## Lab Task

1. Write a recursive function in C to calculate the sum of all numbers from 1 to a given positive integer n. Implement a base case to stop the recursion at n = 1, where the function should return 1. Display the process of stack rolling and unrolling by printing each function call.

*Example*: If n = 5, the output should be 15 (1 + 2 + 3 + 4 + 5).

1. Implement a recursive function in C to find the GCD of two positive integers a and b using the Euclidean algorithm:

* If b is zero, return a (base case).
* Otherwise, call the function recursively with b and a % b.

Test your function with different values of a and b and observe the call stack's behavior.

1. Write a recursive function in C to count the number of digits in a positive integer n. For example, if n = 12345, the output should be 5. Use the following approach:

* Base case: if n is less than 10, return 1.
* Recursive case: return 1 + count\_digits(n / 10) to count each digit.

Print each recursive call to observe stack rolling and unrolling.

**Lab Task Submission Instructions**

1. Screenshot your code and output and paste it into a Word file.
2. Copy paste your code into that word file according to Question number.
3. Number your Word document as question number e.g.Q1.sb, Q2.sb, etc. (Q is in upper case)
4. Create a new folder with your Name and ID e.g. Naveed Ahmed (61452)\_Lab 04 Assign.
5. Copy all the Word document and source files into this folder.
6. Right-click on the folder you created and create a zip file by selecting the option
7. “Send to” and selecting “Compressed (zipped) folder” [for windows].
8. “Create Archive” and change option to “.zip” instead of “ .tar.gz” and click on “Create”. [for linux]
9. Now make sure a zip file is created with your name and id e.g. Naveed Ahmed

(61452).zip.

1. Submit lab task on LMS.

*Note:*

*Plagiarism is strictly prohibited. Any instance of plagiarism will result in a deduction of 100 points. Ensure that all work submitted is original.*

## Home Task

1. Create a recursive function in C to calculate the nth term in the Fibonacci sequence, where the Fibonacci sequence is defined as:

* F(0) = 0 and F(1) = 1 (base cases)
* For n > 1: F(n) = F(n-1) + F(n-2)

Test your function with different values of n. Observe the function call stack and see how recursion handles calculating the same term multiple times. Discuss how this could be optimized.

1. Write a recursive function in C to reverse a string. The function should take the string and its length as parameters, and swap characters recursively until the entire string is reversed.

* Base case: If the left index is greater than or equal to the right index, stop the recursion.
* Recursive case: Swap the left and right characters, then recursively call the function with updated indices.

1. Write a recursive function in C to calculate the power of a number x raised to y (i.e., x^y). Use the following approach:

* Base case: If y = 0, return 1.
* Recursive case: Return x \* power(x, y - 1).

**Lab Assignment Submission Instructions**

1. Screenshot your code and output and paste it into a Word file.
2. Copy paste your code into that word file according to Question number.
3. Number your Word document as question number e.g.Q1.sb, Q2.sb, etc. (Q is in upper case)
4. Create a new folder with your Name and ID e.g. Naveed Ahmed (61452)\_Lab 04 Assign.
5. Copy all the Word document and source files into this folder.
6. Right-click on the folder you created and create a zip file by selecting the option
7. “Send to” and selecting “Compressed (zipped) folder” [for windows].
8. “Create Archive” and change option to “.zip” instead of “ .tar.gz” and click on “Create”. [for linux]
9. Now make sure a zip file is created with your name and id e.g. Naveed Ahmed

(61452).zip.

1. Submit Assignment on LMS.

*Note:*

*Plagiarism is strictly prohibited. Any instance of plagiarism will result in a deduction of 100 points. Ensure that all work submitted is original.*

## Discussion and Analysis of Results

|  |
| --- |
|  |

## Conclusion

|  |
| --- |
|  |

**Lab 11**

# **Understanding Pointers in C**

## Objective:

* Understand the concept of pointers and their use in C programming.
* Learn how to declare, initialize, and use pointers to access and manipulate memory.
* Explore how pointers enable "pass by reference" to allow functions to modify actual variables.
* Study advanced applications of pointers, including pointer arithmetic and dynamic memory allocation.

## Required Equipment / tools:

Dev C++

## Introduction

Pointers are one of the most powerful and essential features in the C programming language. A pointer is a variable that stores the memory address of another variable, enabling programs to directly access and manipulate memory. By using pointers, C provides a level of control over hardware and memory that is not available in many high-level languages.

**Why Use Pointers?**

Efficient Memory Management: Pointers allow for direct access and modification of memory, which can improve performance.

**Dynamic Memory Allocation**: With pointers, memory can be allocated during runtime using functions like **malloc()**, **calloc()**, and **free().**

**Pass by Reference:** Pointers enable functions to modify variables directly by passing their memory addresses, avoiding the overhead of copying large structures or arrays.

**Data Structure Implementation**: Pointers are fundamental for implementing advanced data structures like linked lists, trees, and graphs.

**Low-Level Programming:** Pointers allow access to memory-mapped hardware, making them indispensable for embedded systems and operating system development.

**Syntax**

To declare a pointer, use the \* symbol with a data type:

**variables type \*pointer\_name;**

* The \* indicates that the variable is a pointer.
* The pointer must be initialized with the address of a variable before use, which is done using the address-of operator (&).

**Example of variables**

int \*p; // Pointer to an integer

float \*q; // Pointer to a float

char \*r; // Pointer to a character

## Procedure

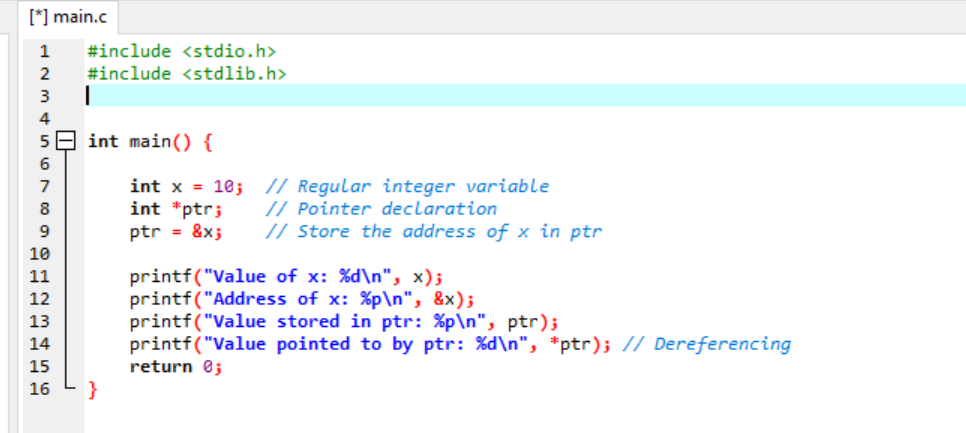
**Pointer Basics**

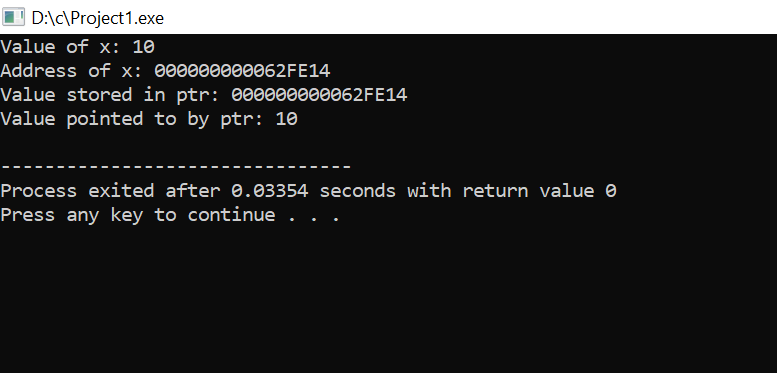
**Declaring and Using Pointers**

**Assigning Addresses:** Assign the address of a variable to a pointer using the & operator.

**Dereferencing:** Use the \* operator to access or modify the value at the memory address stored in the pointer.

Example:

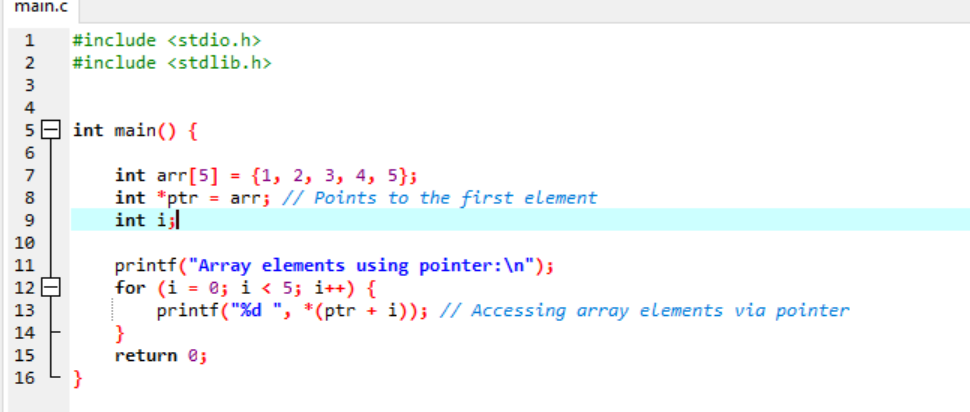


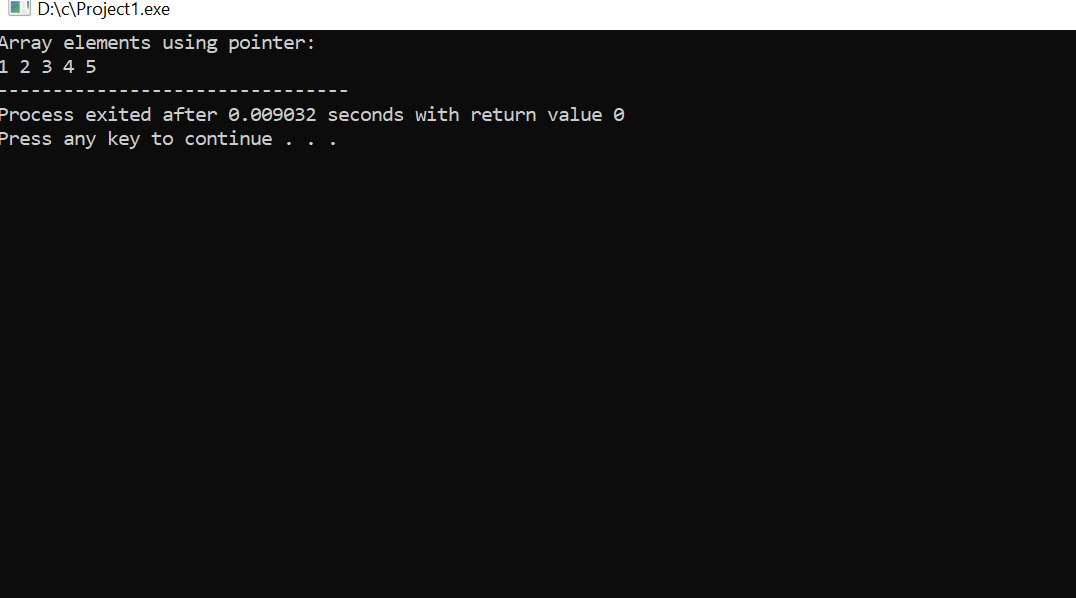


**Pointers with Arrays**

Pointers and arrays are closely related in C. An array name is essentially a pointer to the first element of the array.

Example:

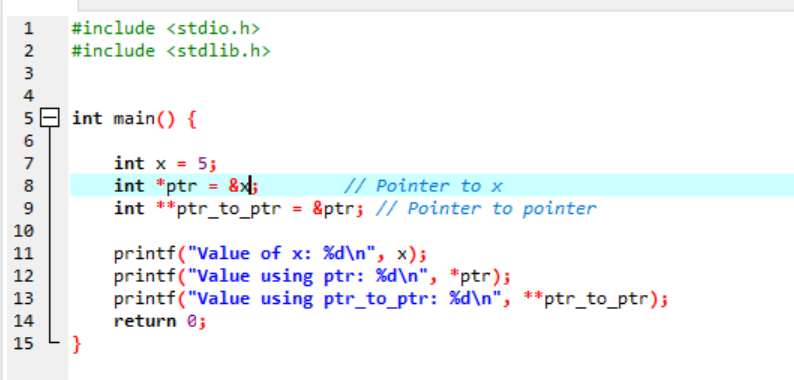


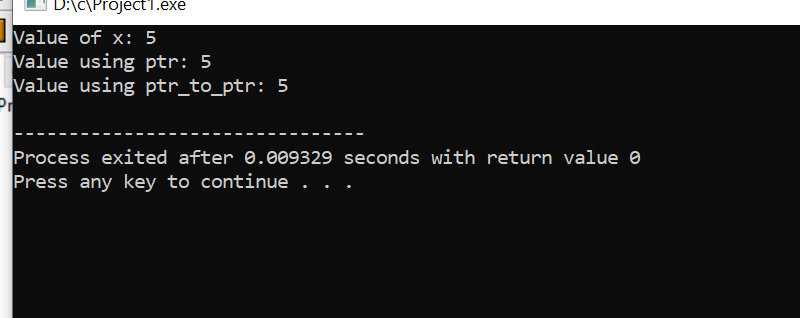


**Pointer to Pointer**

You can also have pointers to pointers (double pointers), which store the address of another pointer.

Example:





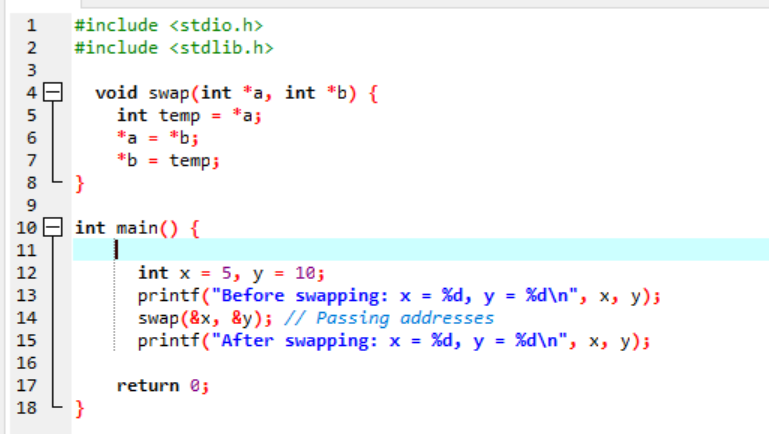
**Passing Pointers as Function Arguments (Pass by Reference)**

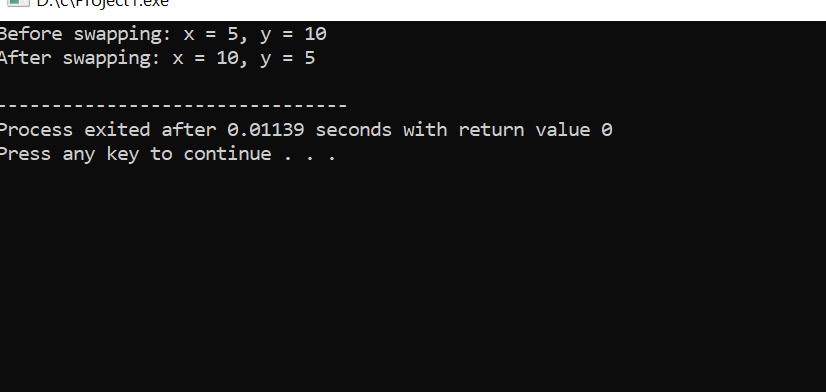
**Why Use Pointers in Functions?**

By passing a pointer to a function, the function can modify the actual variable rather than working with a copy.

This is referred to as "pass by reference."

Example : Swapping Two Numbers





## Observation

|  |
| --- |
|  |

## Lab Tasks

1. Write a program to declare an integer variable, assign a value, and use a pointer to display the value and its address.
2. Write a program to calculate the sum of all elements in an array using pointers.
3. Write a program to demonstrate the use of a double pointer to access the value of a variable.
4. Implement a program to swap two numbers using a pointer.

**Lab Task Submission Instructions**

1. Screenshot your code and output and paste it into a Word file.
2. Copy paste your code into that word file according to Question number.
3. Number your Word document as question number e.g.Q1.sb, Q2.sb, etc. (Q is in upper case)
4. Create a new folder with your Name and ID e.g. Naveed Ahmed (61452)\_Lab 04 Assign.
5. Copy all the Word document and source files into this folder.
6. Right-click on the folder you created and create a zip file by selecting the option
7. “Send to” and selecting “Compressed (zipped) folder” [for windows].
8. “Create Archive” and change option to “.zip” instead of “ .tar.gz” and click on “Create”. [for linux]
9. Now make sure a zip file is created with your name and id e.g. Naveed Ahmed

(61452).zip.

1. Submit lab task on LMS.

*Note:*

*Plagiarism is strictly prohibited. Any instance of plagiarism will result in a deduction of 100 points. Ensure that all work submitted is original.*

## Home Tasks

Q1: Write a program to declare an integer variable and a pointer. Perform the following operations:

1. Assign a value to the integer variable and assign its address to the pointer.
2. Print the address of the variable and the pointer's address.
3. Modify the value of the integer variable using the pointer and print the updated value.

Q2: Write a program to reverse an array in place using pointers. Pass the array's base address to a function that performs the reversal using pointer arithmetic.

Q3: Write a program to create a structure to store information about a student (name, roll number, and marks). Use a pointer to access and display the structure's data.

**Lab Assignment Submission Instructions**

1. Screenshot your code and output and paste it into a Word file.
2. Copy paste your code into that word file according to Question number.
3. Number your Word document as question number e.g.Q1.sb, Q2.sb, etc. (Q is in upper case)
4. Create a new folder with your Name and ID e.g. Naveed Ahmed (61452)\_Lab 04 Assign.
5. Copy all the Word document and source files into this folder.
6. Right-click on the folder you created and create a zip file by selecting the option
7. “Send to” and selecting “Compressed (zipped) folder” [for windows].
8. “Create Archive” and change option to “.zip” instead of “ .tar.gz” and click on “Create”. [for linux]
9. Now make sure a zip file is created with your name and id e.g. Naveed Ahmed

(61452).zip.

1. Submit Assignment on LMS.

*Note:*

*Plagiarism is strictly prohibited. Any instance of plagiarism will result in a deduction of 100 points. Ensure that all work submitted is original.*

## Discussion and Analysis of Results

|  |
| --- |
|  |

## Conclusion

|  |
| --- |
|  |

**Lab 12**

## 

# **Strings in C**

## Objective:

* Understand how a string constant is stored in a character array.
* Learn the use of character arrays (strings) in C programming.
* Understand and use the %s placeholder in printf() and scanf() functions for strings.
* Apply loops for manipulating strings stored in character arrays.

**Prerequisites**:

* Basic knowledge of arrays and loop structures in C.
* Familiarity with printf() and scanf() functions.

## Required Equipment / tools:

Dev C++

## Introduction:

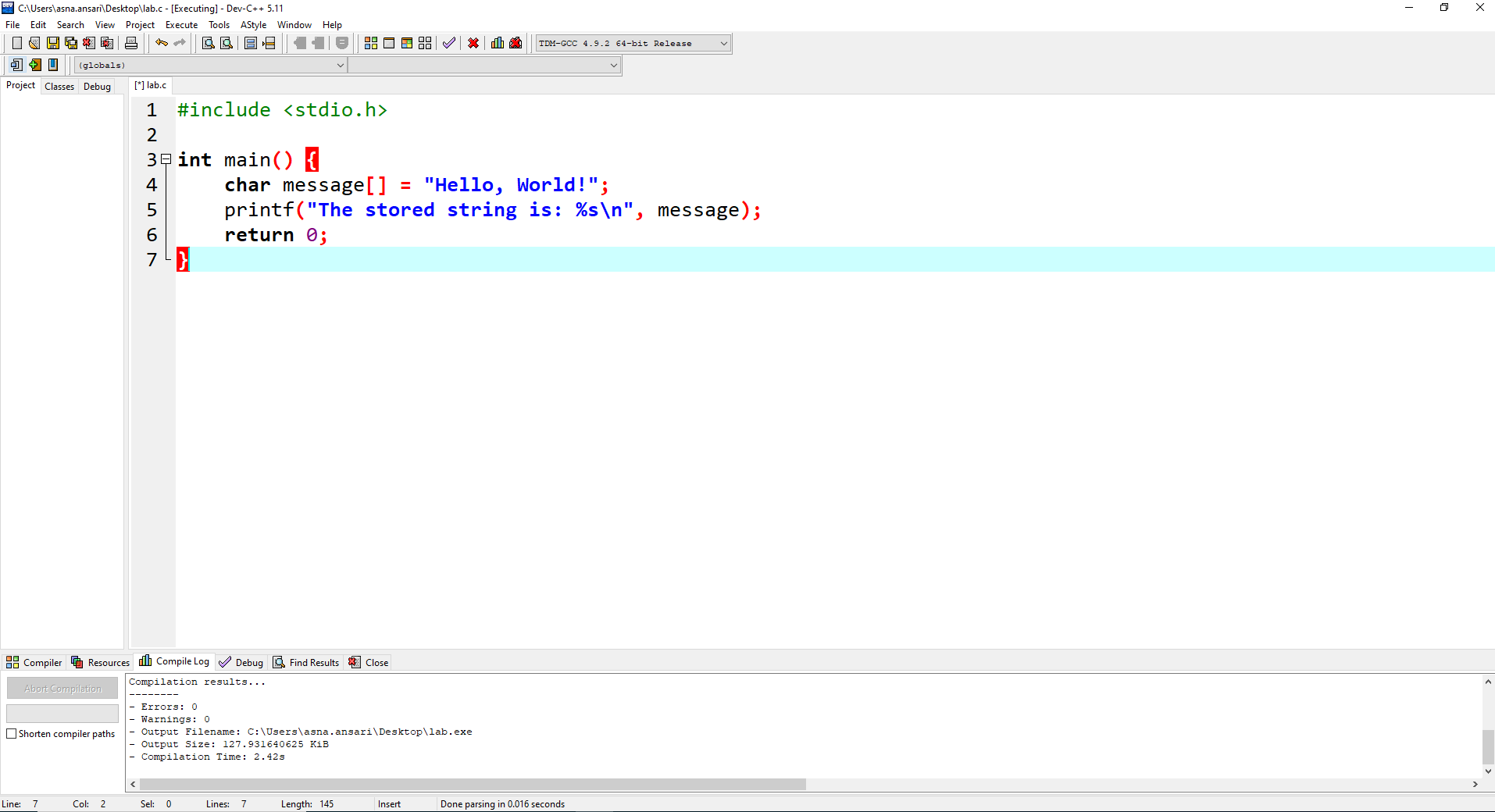
This lab focuses on understanding and working with strings in C programming. Strings in C are stored as arrays of characters, where each character occupies a sequential memory location, ending with a null character (`'\0'`) to signify the string’s termination. Through this lab, students will gain hands-on experience with character arrays, explore how string constants are stored, learn to use the `%s` placeholder with `printf()` and `scanf()` for handling strings, and practice using loops for performing various string manipulations. This foundational knowledge will be essential for managing text data effectively in C programs

## Procedure

**How a String Constant is Stored in a Character Array:**

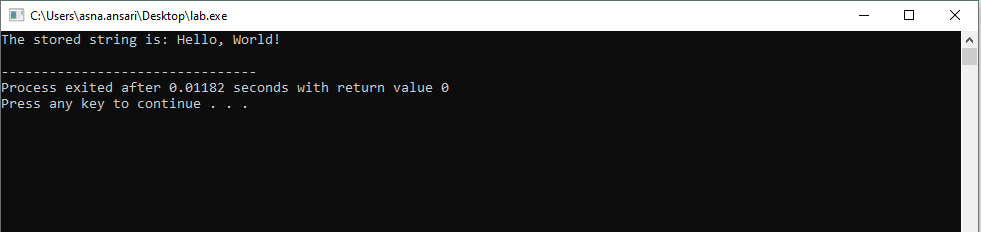
In C, a string is stored as an array of characters, with each character taking one byte of memory. A null character ('\0') is automatically added at the end of the string to indicate its termination. This null terminator helps functions like printf() know when to stop reading characters.

**Program 1: Storing String Constant in a Character Array**



*Figure 1*

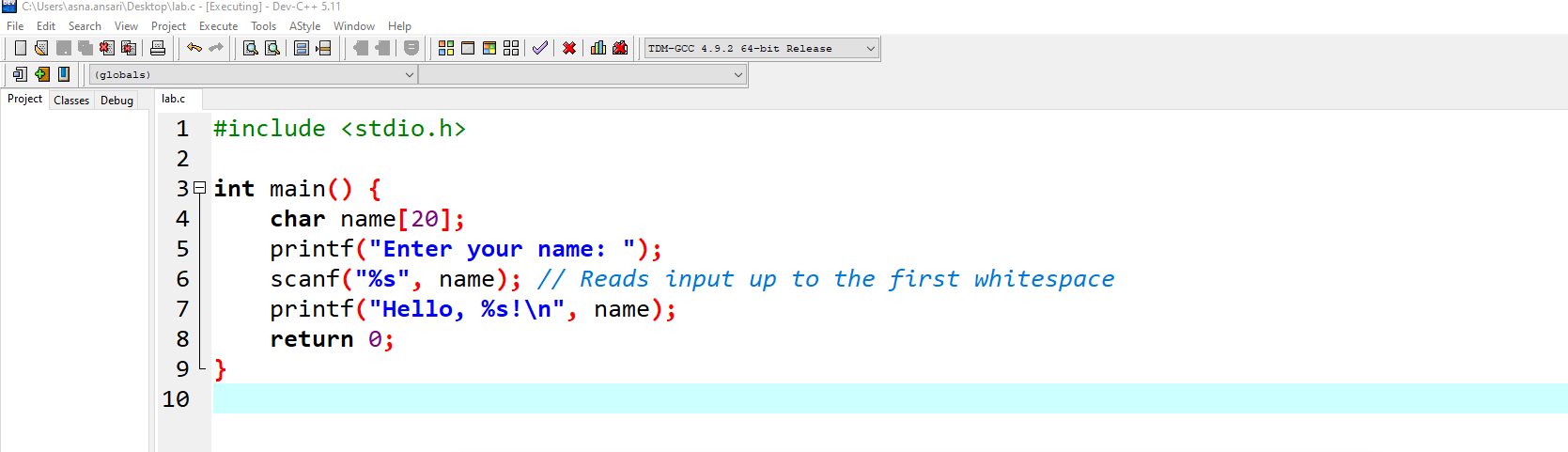
**Output #1:**



* 1. **Using Character Arrays (Strings) with %s Placeholder:**

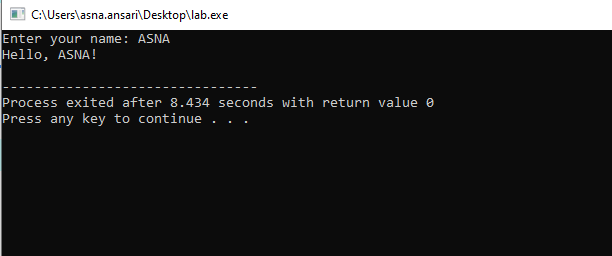
The %s placeholder is used to display or take input of strings in C. When used with printf(), it prints the entire string up to the null terminator. When used with scanf(), it reads the input until the first whitespace is encountered.

**Program 2: Using Character Arrays (String) with %s Placeholder**



*Figure 2*

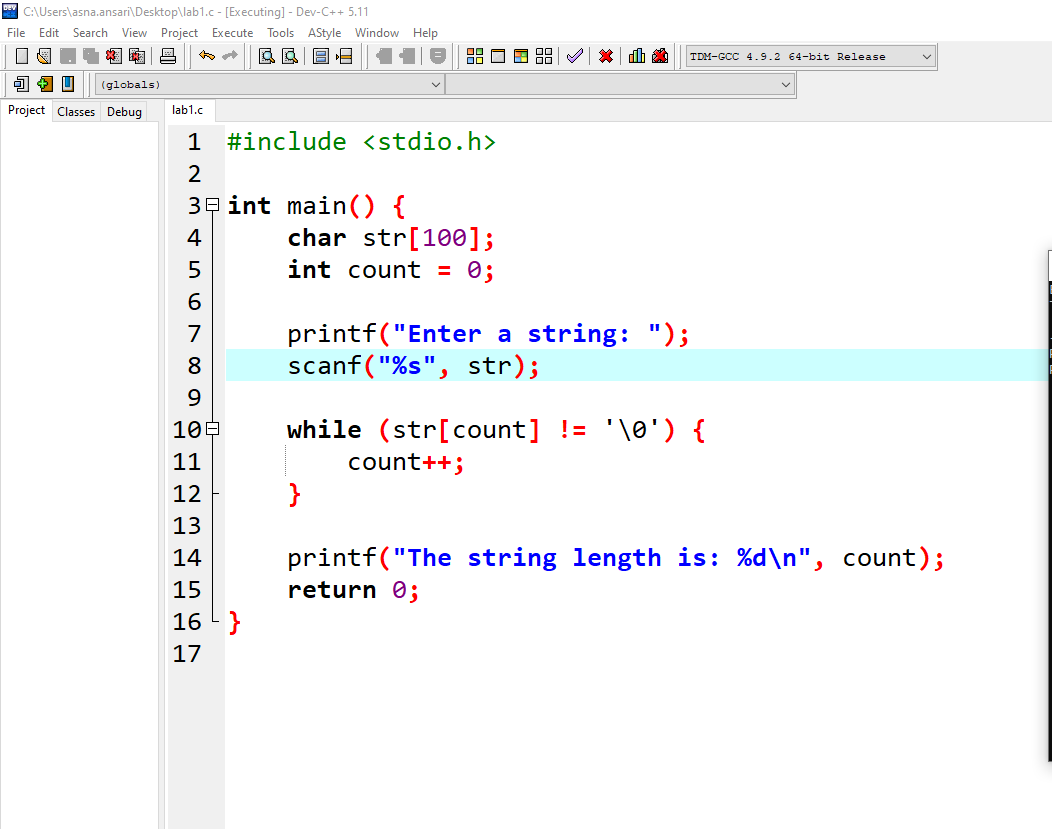
**Output #2:**



* 1. **String Manipulations Using Loops:**

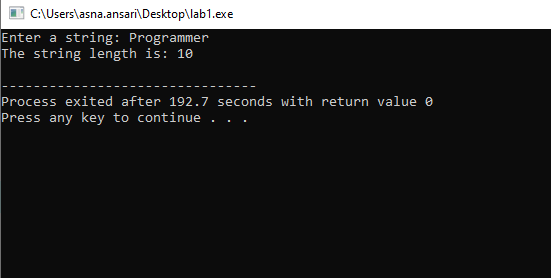
Loops enable us to access each character in a string and perform manipulations like counting, converting cases, or reversing the string. By using a while or for loop, each character can be processed individually until the null terminator is encountered.

**Program 3: Count the Number of Characters in a String**

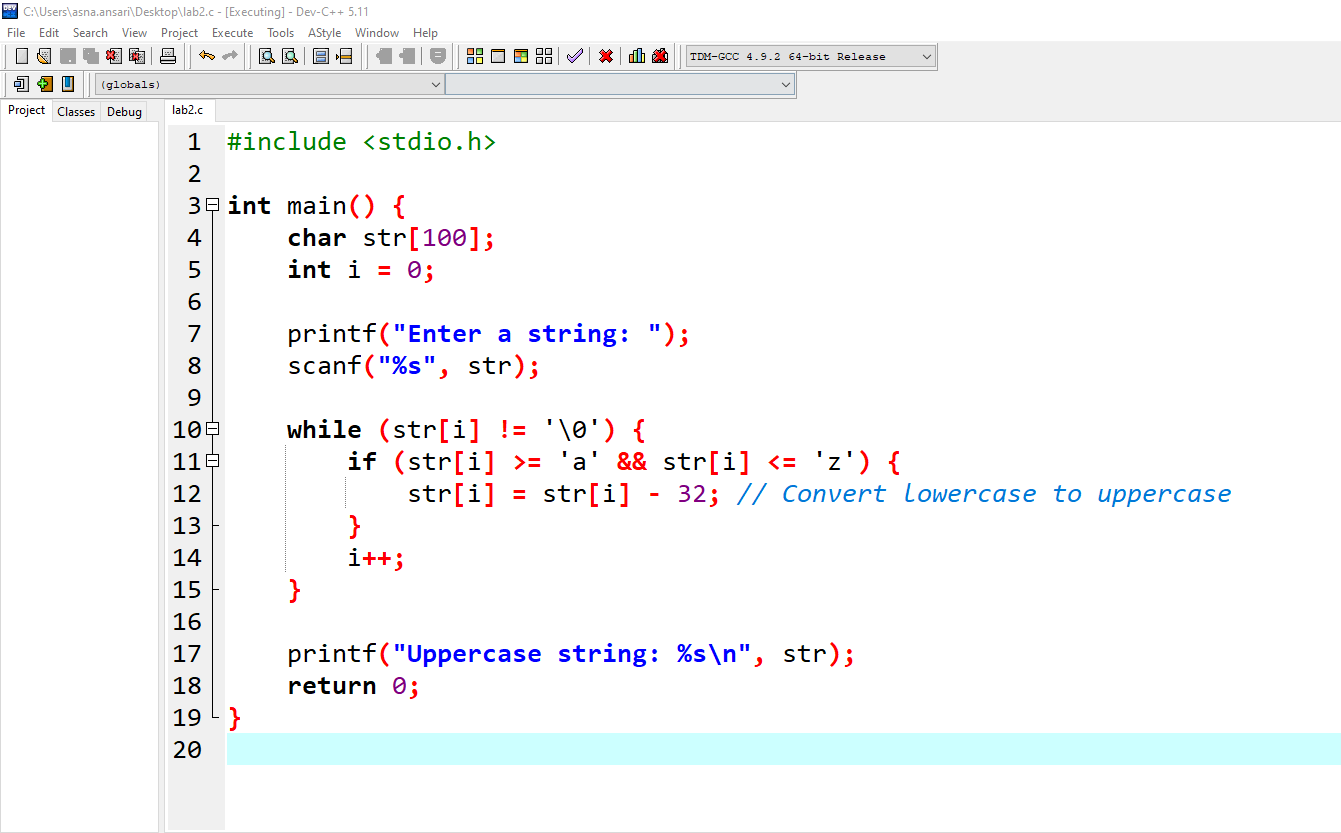
****

*Figure 3*

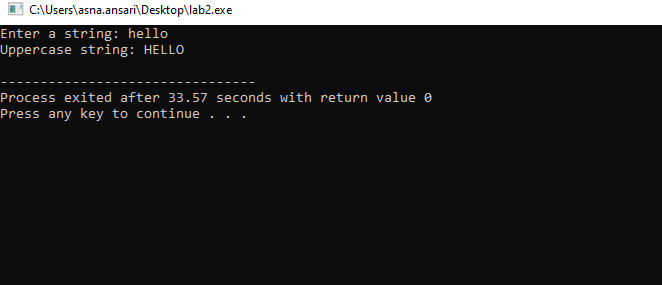
**Output #3:**



**Program 4: Convert a String to Uppercase**



**Output #4:**



## Observation

|  |
| --- |
|  |

## Lab Task

1. Simulate a simple user registration system. Prompt the user to enter their first name and last name separately. Store each name in a separate character array, then combine them into a full name with a space in between. Display the full name using printf().
2. Write a C program that prompts the user to enter their full name and then prints a greeting message (e.g., "Welcome, [Name]!"). Use %s in both scanf() to read input and printf() to display output.
3. Write a C program that takes a string as input and prints each character of the string on a new line. For example, if the input string is "Programming", the output should display each letter on a separate line.

**Lab Task Submission Instructions**

1. Screenshot your code and output and paste it into a Word file.
2. Copy paste your code into that word file according to Question number.
3. Number your Word document as question number e.g.Q1.sb, Q2.sb, etc. (Q is in upper case)
4. Create a new folder with your Name and ID e.g. Naveed Ahmed (61452)\_Lab 04 Assign.
5. Copy all the Word document and source files into this folder.
6. Right-click on the folder you created and create a zip file by selecting the option
7. “Send to” and selecting “Compressed (zipped) folder” [for windows].
8. “Create Archive” and change option to “.zip” instead of “ .tar.gz” and click on “Create”. [for linux]
9. Now make sure a zip file is created with your name and id e.g. Naveed Ahmed

(61452).zip.

1. Submit lab task on LMS.

*Note:*

*Plagiarism is strictly prohibited. Any instance of plagiarism will result in a deduction of 100 points. Ensure that all work submitted is original.*

## Home Tasks

1. Write a C program that prompts the user to enter a sentence and then prints the sentence with each word reversed but in the original order. For example, if the input is "Hello World", the output should be "olleH dlroW".
2. Write a C program that prompts the user to enter a sentence, then identifies and prints the longest word in the sentence. If there are multiple words with the same maximum length, the program should print the first one that appears.

For example, if the input is "C programming is challenging", the output should be "programming".

1. Write a program that takes a string input and removes all vowels from it. For example, if the input is "Programming", the output should be "Programming".

**Lab Assignment Submission Instructions**

1. Screenshot your code and output and paste it into a Word file.
2. Copy paste your code into that word file according to Question number.
3. Number your Word document as question number e.g.Q1.sb, Q2.sb, etc. (Q is in upper case)
4. Create a new folder with your Name and ID e.g. Naveed Ahmed (61452)\_Lab 04 Assign.
5. Copy all the Word document and source files into this folder.
6. Right-click on the folder you created and create a zip file by selecting the option
7. “Send to” and selecting “Compressed (zipped) folder” [for windows].
8. “Create Archive” and change option to “.zip” instead of “ .tar.gz” and click on “Create”. [for linux]
9. Now make sure a zip file is created with your name and id e.g. Naveed Ahmed

(61452).zip.

1. Submit Assignment on LMS.

*Note:*

*Plagiarism is strictly prohibited. Any instance of plagiarism will result in a deduction of 100 points. Ensure that all work submitted is original.*

## Discussion and Analysis of Results

|  |
| --- |
|  |

## Conclusion

|  |
| --- |
|  |

**Lab 13**

# **File Handling in C**

## Objective:

To understand file handling concepts in C programming, including file creation, reading, writing, and appending using standard library functions.

## Required Equipment / tools:

VS Code/Dev C++/Turbo C

## Introduction:

File handling allows a program to interact with files stored on the disk. Files provide a way to store data permanently, beyond the program's execution. In C, the stdio.h library provides functions for file operations.

**1.1.1 Key File Modes:**

Here is the information presented in a tabular format for better understanding:

|  |  |  |
| --- | --- | --- |
| **File Mode** | **Purpose** | **Behavior** |
| **"r"** | Read | Open a file for reading. Returns NULL if the file doesn’t exist. |
| **"w"** | Write | Open a file for writing. Creates the file if it doesn’t exist, overwrites if it does. |
| **"a"** | Append | Open a file for appending. Adds data to the end without overwriting existing content. |
| **"r+"** | Read and Write | Open a file for both reading and writing. The file must exist. |
| **"w+"** | Write and Read | Open a file for both reading and writing. Creates or overwrites the file. |
| **"a+"** | Append and Read | Open a file for both reading and appending. |

## Procedure

**1. FILE fopen(const char filename, const char mode)** This function opens a file and allows you to read, write, or append data to it. It returns a pointer to the file, which you can use for further operations

* **Parameters:**
  + filename: Name of the file (e.g., "data.txt").
  + mode: Mode in which the file is opened (e.g., "r" for read, "w" for write, "a" for append).

|  |
| --- |
| **Example** |
| #include <stdio.h>    int main() {  FILE \*file = fopen("example.txt", "w");  if (file == NULL) {  printf("Error opening file.\n");  return 1;  }  fprintf(file, "Hello, World!\n");  fclose(file);  return 0;  } |

**2. fclose(FILE file)**

This function closes an open file and ensures that any changes made to it are saved. It also releases the resources used by the file.

|  |
| --- |
| **Example** |
| FILE \*file = fopen("example.txt", "r");  if (file != NULL) {  // Perform file operations...  fclose(file);  } |

**3. fprintf(FILE file, const char format, ...)**

This function writes text to a file in a specific format, just like printf writes text to the screen.

|  |
| --- |
| **Example** |
| #include <stdio.h>    int main() {  FILE \*fptr;    fptr = fopen("filename.txt", "w");  fprintf(fptr, "Some text");  fptr = fopen("filename.txt", "a");  fprintf(fptr, "\nHi everybody!");  fclose(fptr);  return 0; |

**4. fgets(char str, int size, FILE file)**

This function reads one line of text from a file and stores it in a string. It stops when it reaches a newline or the specified number of characters.

|  |
| --- |
| **Example** |
| #include <stdio.h>  int main() {  FILE \*fptr;  fptr = fopen("filename.txt", "r");  char myString[2];  fgets(myString, 2, fptr);  printf("%s", myString);  fclose(fptr);  } |

**5. fscanf(FILE file, const char format, ...)** Reads formatted data from a file. This function reads data from a file based on a specified format, just like scanf reads input from the user. Similar to scanf but for files.

* **Parameters:**
  + file: Pointer to the file to read from.
  + format: Format string to specify the expected input.

|  |
| --- |
| **Example** |
| #include <stdio.h>  int main() {  FILE \*file = fopen("data.txt", "r");  if (file == NULL) {  printf("Error opening file.\n");  return 1;}  char name[20];  int age;  fscanf(file, "%s %d", name, &age);  printf("Name: %s, Age: %d\n", name, age);  fclose(file);  return 0;} |

## Observation

|  |
| --- |
|  |

## Lab Task

1. Write a C program that takes input from the user (name, age, and grade) and writes this data to a file called student\_info.txt. Use the fprintf() function for writing structured data.

2. Write a C program that reads and displays the content of a file named employee\_details.txt line by line using the fgets() function.

3. Write a C program that opens a file named student\_data.txt for writing. The program should then close the file.

**Lab Task Submission Instructions**

**1.** Screenshot your code and output and paste it into a Word file.

**2.** Copy paste your code into that word file according to Question number.

**3.** Number your Word document as question number e.g.Q1.sb, Q2.sb, etc. (Q is in upper case)

**4.** Create a new folder with your Name and ID e.g. Naveed Ahmed (61452)\_Lab 04 Assign.

**5.** Copy all the Word document and source files into this folder.

**6.** Right-click on the folder you created and create a zip file by selecting the option

**7.** “Send to” and selecting “Compressed (zipped) folder” [for windows].

**8.** “Create Archive” and change option to “.zip” instead of “ .tar.gz” and click on “Create”. [for linux]

**9.** Now make sure a zip file is created with your name and id e.g. Naveed Ahmed

(61452).zip.

**10.** Submit lab task on LMS.

*Note:*

*Plagiarism is strictly prohibited. Any instance of plagiarism will result in a deduction of 100 points. Ensure that all work submitted is original.*

## Home Tasks

1. Read from a text file using fscanf(). Read the integer present in the program.txt file and print it onto the screen.

2. Write a C program that appends new information about a product (name, price, and quantity) to an existing file inventory.txt using the append mode ("a").

3. Create a program that reads data from a file called numbers.txt containing integers, calculates the sum of all numbers, and displays the result.

**Lab Task Submission Instructions**

**11.** Screenshot your code and output and paste it into a Word file.

**12.** Copy paste your code into that word file according to Question number.

**13.** Number your Word document as question number e.g.Q1.sb, Q2.sb, etc. (Q is in upper case)

**14.** Create a new folder with your Name and ID e.g. Naveed Ahmed (61452)\_Lab 04 Assign.

**15.** Copy all the Word document and source files into this folder.

**16.** Right-click on the folder you created and create a zip file by selecting the option

**17.** “Send to” and selecting “Compressed (zipped) folder” [for windows].

**18.** “Create Archive” and change option to “.zip” instead of “ .tar.gz” and click on “Create”. [for linux]

**19.** Now make sure a zip file is created with your name and id e.g. Naveed Ahmed

(61452).zip.

**20.** Submit lab task on LMS.

*Note:*

*Plagiarism is strictly prohibited. Any instance of plagiarism will result in a deduction of 100 points. Ensure that all work submitted is original.*

## Discussion and Analysis of Results

|  |
| --- |
|  |

## Conclusion

|  |
| --- |
|  |

**Lab 14**

## 

# **Revision And Question Answer**