



<h1>CL1002</h1> <h2>Programming Fundamentals</h2>	<h1>Lab 03</h1> <h2>Introduction to GitHub and C Programming Language</h2>
---	--

**NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES**

---

Fall 2025

## AIMS AND OBJECTIVES

---

### Aims

1. Teach students how to setup your profile on GitHub and creating your first repository.
2. Equip students with the foundational knowledge needed to write and understand C programs.
3. Enable students to develop efficient and organized code using C language constructs.
4. Foster the ability to apply C programming skills to solve real-world problems.
5. Prepare students for more advanced topics in programming by solidifying their understanding of basic C concepts.

### Objectives

1. Introduce students to GitHub and setting up their profile with their first repository.
2. Introduce the basic syntax and structure of the C programming language.
3. Teach students how to write and compile simple C programs.
4. Familiarize students with fundamental C programming concepts, such as variables and data types.
5. Develop problem-solving skills by writing C programs to solve basic computational problems.

## INTRODUCTION

---

In this lab we will be covering the following topics

1. GitHub
2. Creating a Branch in a Repository
3. Creating a Pull Request of the created branch
4. Merging branches to main repository
5. Introduction to Integrated Development Environment (IDE).
6. Introduction to C-Programming Language (Basic Structure, Inputs and Outputs, Variables, Data types, Format specifiers, escape sequences, Precision)

## OBJECTIVE: 1 GITHUB

GitHub is a web-based platform that provides version control using Git, allowing multiple people to collaborate on projects, track changes, and manage code repositories efficiently. It's widely used by developers for both open-source and private projects.

### CREATE AN ACCOUNT ON GITHUB

To get started first login to GitHub (if you already have an account).

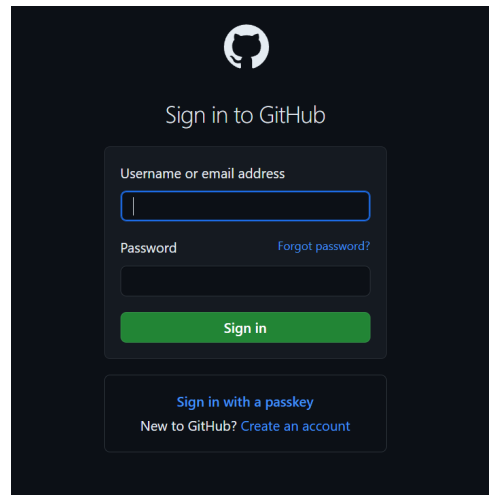


Figure: 1(a) GitHub Login Page

If not, then click on the **create an account** option and enter the necessary credentials to get started. Once logged in you should see a screen something like.

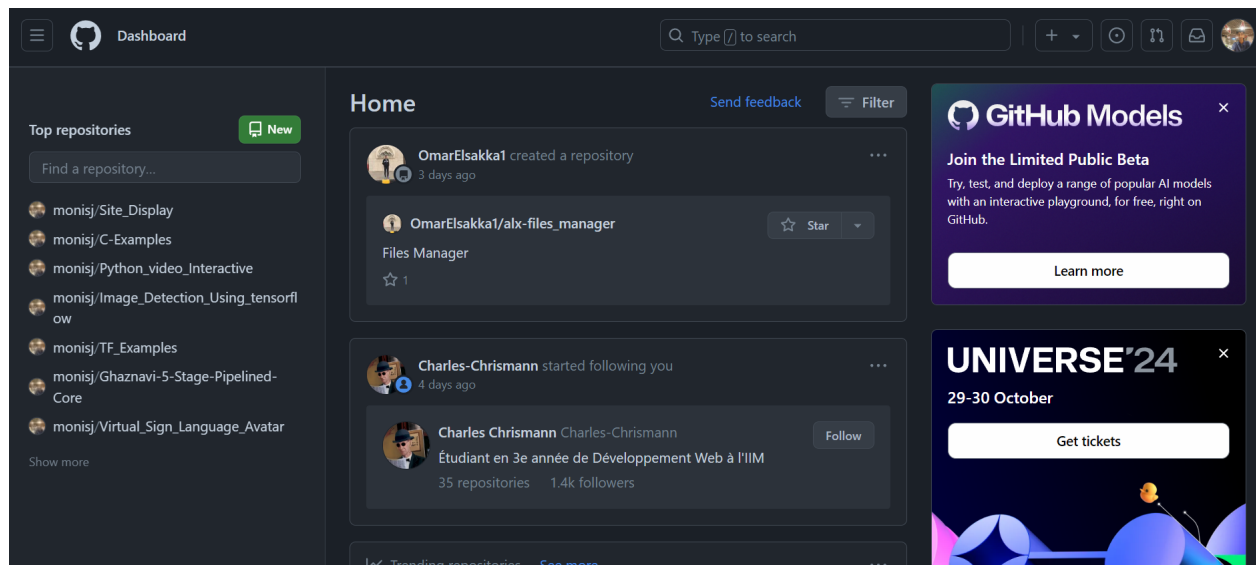


Figure: 1(b) GitHub Homepage

Of course, mine has many repositories so when performing your screen would look a bit different. On the top left section where you see the new button, click on it and you should see a window something like this.

The screenshot shows the GitHub 'Create a new repository' interface. It features a dark theme with white text. At the top, the title 'Create a new repository' is followed by a subtitle and a link to 'Import a repository'. A note states 'Required fields are marked with an asterisk (\*)'. The form is divided into several sections: 'Repository template' with a 'No template' dropdown; 'Owner' with a user selection dropdown; 'Repository name' with a text input and a green availability check; 'Description' with an optional text input; and 'Access to your Repository' with radio buttons for 'Public' and 'Private'. Red boxes highlight the 'No template' dropdown, the 'Repository name' input and its feedback message, the 'Description' input, and the 'Public/Private' access options. Red text annotations provide additional context: 'Template box for saved templates' for the template dropdown, 'Add your Repository name here' for the repository name input, and 'Description about the Repository' for the description input.

**Create a new repository**

A repository contains all project files, including the revision history. Already have a project repository elsewhere?  
[Import a repository.](#)

*Required fields are marked with an asterisk (\*).*

**Repository template**

No template ▾

Template box for saved templates

Start your repository with a template repository's contents.

**Owner \***

monisj ▾

**Repository name \***

skills-introduction-to-githl

skills-introduction-to-github is available.

Add your Repository name here

Great repository names are short and memorable. Need inspiration? How about studios-meme ?

**Description (optional)**

My clone repository

**Access to your Repository**

☒ **Public**  
Anyone on the internet can see this repository. You choose who can commit.

☐ **Private**  
You choose who can see and commit to this repository.

Figure: 1(c) Creating New Repository

When creating a repository, you are required to fill/select at least three sections of the page. The first red box is if you want to use already saved templates (These are already made repositories with some files and branches already stored in them). The second box is your repositories name, you can name your repository anything you like but stick to proper naming conventions and name your repository according to what project you are working on, GitHub will indicate you if your repository name is unique or not. The Third box is optional and is used to display a small description about your repository. Lastly the access box and this can be crucial depending on your project. If you intend to display your project to everyone then you would need to select public access but if you choose to hide your project before releasing it to the public a private option is available and only you and those who are contributing to your project can see the repository.

### Initialize this repository with:

☒ **Add a README file**  
This is where you can write a long description for your project. [Learn more about READMEs.](#)

### Add .gitignore

.gitignore template: **None**

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

### Choose a license

License: **Apache License 2.0**

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

This will set **main** as the default branch. Change the default name in your [settings](#).

*You are creating a public repository in your personal account.*

**Create repository**

**This will add a readme file to your Repository**

**Select a template for which type of files you want to track**

**Licenses define what other people can/cannot do with your work**

Figure: 1(d) Additional details in Repository

Lastly there also exists three more optional options when creating a repository. The First check box is a detailed description about your project, sort of like an actual MUST-READ FILE when installing pirated software. We will cover this later down in the manual on how we can modify this. The second box is for tracking which types of files to ignore, choose none and all files will be tracked otherwise you can select which type of files to exclude from tracking. Lastly a license is not compulsory for this lab but mandatory when releasing your project to the world as it defines the boundaries of what the end user can/cannot do with your work. There are many versions of this license so read and explore accordingly. Once all the necessary boxes are filled click on the **Create repository** button.

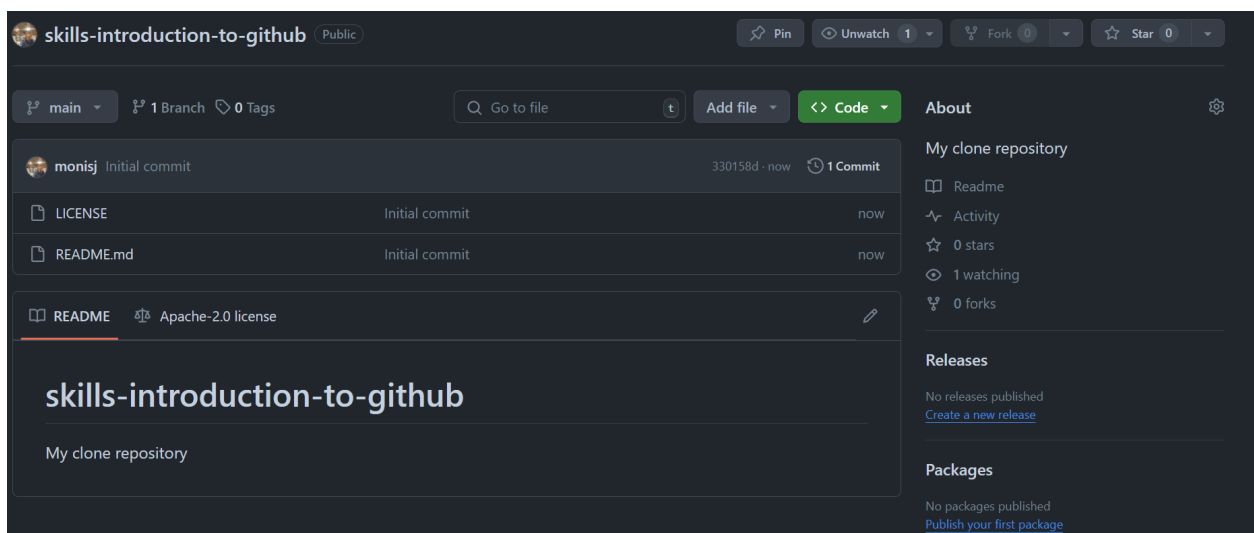


Figure: 1(e) Repository Homepage

You should see a screen something like this. The bottom section is the readme file, the Top right is the description for the repository and the center is where you will see your files which you have uploaded. Right now, there are only two files as we created a license and a readme file.

## COMMIT A FILE

**What is a commit?** A commit is a set of changes to the files and folders in your project. A commit exists in a branch and is responsible for tracking what files were uploaded, deleted or modified in the branch.

To get started On the < > **Code** tab in the header menu of your repository, make sure you're on your new branch my-first-branch.

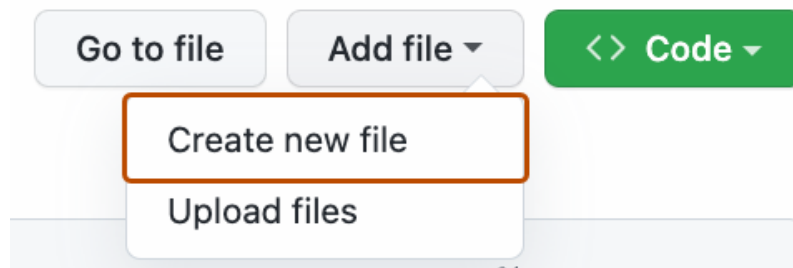


Figure: 1(f) Creating a File

Select the **Add file** drop-down and click **Create new file**.



Figure 1(g) Commit Changes in the File

In the **Name your file...** field, enter PROFILE.md.

In the **Enter file contents here** area, Write a text i.e. Programming Fundamentals Fall 2025

Click **Commit changes...** in the upper right corner above the contents box.

You should see a window pop something like this down below.

## MARKDOWN LANGUAGE FOR README FILES

Markdown is a lightweight markup language that you can use to add formatting elements to plaintext text documents. GitHub uses its own flavor of Markdown, which includes some additional features beyond the standard Markdown. Here's a guide to some of the basic elements:

Before we start click on the pencil icon in the readme section of your GitHub Repo

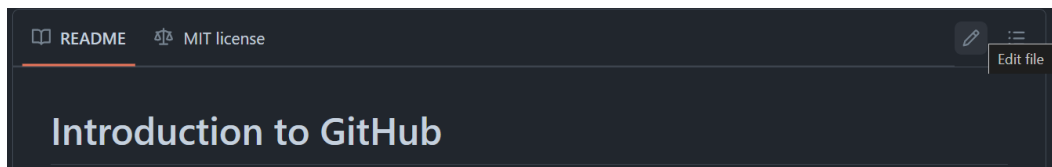


Figure: 1(h) Headline of the Readme File

Once clicked you should see a window of something like this

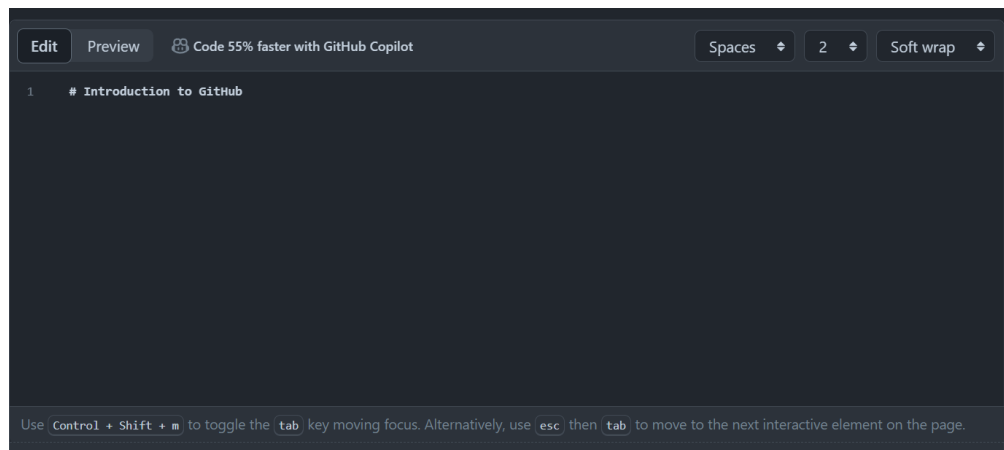


Figure 1(i) Editable Readme File Page

Here you are going to add changes so your readme file could look fancy and professional . There is a preview page so whatever you change you make to your file you can see the effect in real-time.

## Headings

To create a heading, add one to six # symbols before your heading text. The number of # you use will determine the hierarchy level and typeface size of the heading.


```
01    # A first-level heading
02    ## A second-level heading
03    ### A third-level heading
```

# A first-level heading

## A second-level heading

### A third-level heading

Figure: 2 Headings of Different Sizes

When you use two or more headings, GitHub automatically generates a table of contents that you can access by clicking  within the file header. Each heading title is listed in the table of contents, and you can click a title to navigate to the selected section.

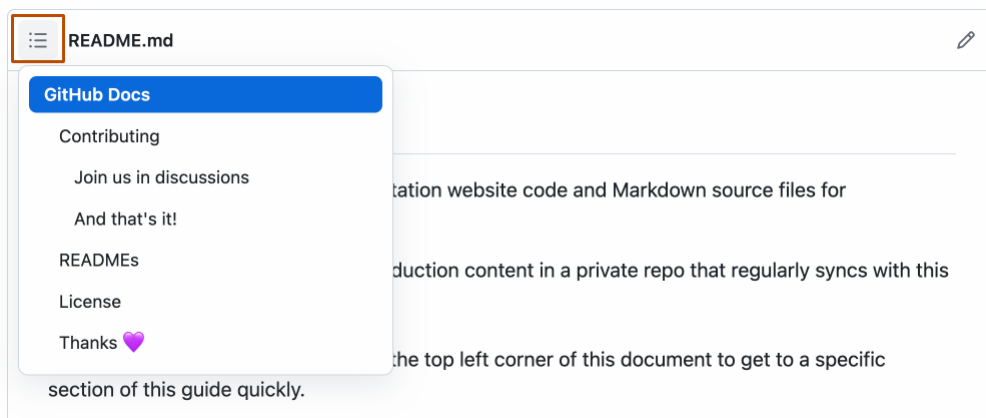


Figure: 1(k) Table of Contents in Readme File

## Styling text

You can indicate emphasis with bold, italic, strikethrough, subscript, or superscript text in comment fields and .md files.

Style	Syntax	Keyboard shortcut	Example	Output
Bold	**	Command+B (Mac)	**This is bold text**	<b>This is bold text</b>
	** or __	or Ctrl+B (Windows/Linux)		



Style	Syntax	Keyboard shortcut	Example	Output
Italic	<b>** or _</b> _	Command+I (Mac) or Ctrl+I (Windows/Linux)	_This text is italicized_	<i>This text is italicized</i>
Strikethrough	~~ ~~	None	~~This was mistaken text~~	<del>This was mistaken text</del>
Bold and nested italic	<b>**</b> <b>** and _</b> _	None	<b>**This text is _extremely_ important**</b>	<b>This text is <i>extremely</i> important</b>
All bold and italic	<b>*** ***</b>	None	<b>***All this text is important***</b>	<b><i>All this text is important</i></b>
Subscript	<code>&lt;sub&gt;</code> <code>&lt;/sub&gt;</code>	None	This is a <code>&lt;sub&gt;subscript&lt;/sub&gt;</code> text	This is a <sub>subscript</sub> text
Superscript	<code>&lt;sup&gt;</code> <code>&lt;/sup&gt;</code>	None	This is a <code>&lt;sup&gt;superscript&lt;/sup&gt;</code> text	This is a <sup>superscript</sup> text

#### Example

```

01    # Introduction to GitHub
02    **This text is bold**\
03    *This text is italics*\
04    ***This text is both bold and italics***\
05    ~~OOPS I made an error~~

```

#### Output

```

Introduction to GitHub

This text is bold
This text is italics
This text is both bold and italics
OOPS I made an error

```

Figure: 1(L) Output of Example

Note the ‘\’ is used for newline. Heading do not require this.

## Quoting text

You can quote text with a `>` symbol.

```
01 Text that is not a quote
06 > Text that is quote
```

## Quoting code

You can call out code or a command within a sentence with single backticks. The text within the backticks will not be formatted. You can also press the Command+E (Mac) or Ctrl+E (Windows/Linux) keyboard shortcut to insert the backticks for a code block within a line of Markdown.

```
07 Use `git status` to list all new or modified files that haven't yet been committed. 08
Some basic Git commands are:
  ` ` `
09
10 git status
11 git add
12 git commit
  ` ` `
13
```

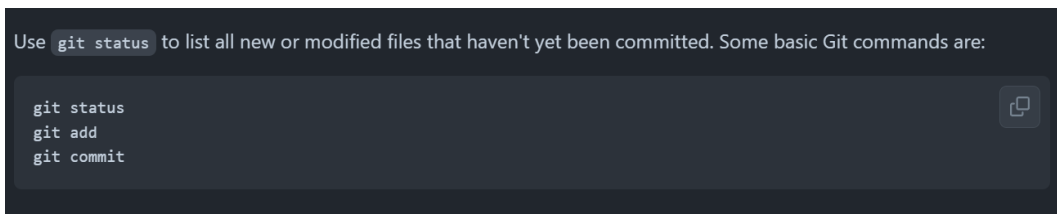


Figure: 1(m) Some basic Git Commands

## Lists

You can make an unordered list by preceding one or more lines of text with `-`, `*`, or `+`.

```
14 - George Washington
15 * John Adams
16 + Thomas Jefferson
```

- George Washington
- John Adams
- Thomas Jefferson

Figure 1(n) Output of Unordered Lists

To order your list, precede each line with a number.

```
01    1. James Madison
17    2. James Monroe
18    3. John Quincy Adams
```

1. James Madison
2. James Monroe
3. John Quincy Adams

Figure: 3 Output of Ordered Lists

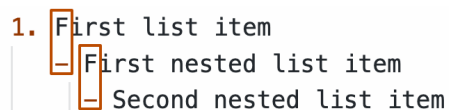
## Nested Lists

You can create a nested list by indenting one or more list items below another item.

To create a nested list using the web editor on GitHub or a text editor that uses a monospaced font, like [Visual Studio Code](#), you can align your list visually. Type space characters in front of your nested list item until the list marker character (- or \*) lies directly below the first character of the text in the item above it.

```
19  1. First list item
20    - First nested list item
21      - Second nested list item
```

Output



1. First list item  
- First nested list item  
- Second nested list item

1. First list item
  - First nested list item
    - Second nested list item

Figure 1(p) Output of Nested Lists

## Task lists

To create a task list, preface list items with a hyphen and space followed by [ ]. To mark a task as complete, use [x].

```
01  - [x] #739
02  - [ ] https://github.com/octo-org/octo-repo/issues/740
```

03 - [ ] Add delight to the experience when all tasks are complete  
:tada:

- ☒ ☒ Convert text into issues #739
- ☐ ☒ Keep issue state and checkboxes in sync #740
- ☐ Add delight to the experience when all tasks are complete 🎉

Figure: 1(q) Output of Task Lists

## Images

You can display an image by adding! and wrapping the alt text in []. Alt text is a short text equivalent of the information in the image. Then, wrap the link for the image in parentheses ().

![Screenshot of a comment on a GitHub issue showing an image, added in the Markdown, of an Octocat smiling and raising a tentacle.](<https://myoctocat.com/assets/images/base-octocat.svg>)

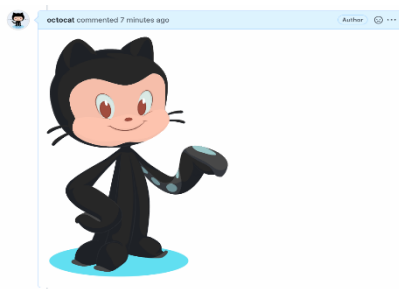


Figure: 1(r) Output of Images

Note that direct web link and link to your repository path where the image is located can be utilized in this manner.

## OBJECTIVE: 2 CREATE A BRANCH IN THE REPOSITORY

**What is a branch?** A branch is a parallel version of your repository. By default, your repository has one branch named **main** and it is the definitive branch. Creating additional branches allows you to copy the **main** branch of your repository and safely make any changes without disrupting the **main** project. Many people use branches to work on specific features without affecting any other parts of the project.

Branches allow you to separate your work from the **main** branch. In other words, everyone's work is safe while you contribute.

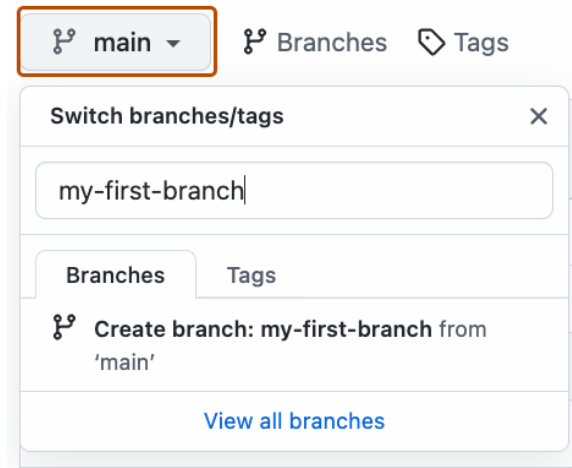


Figure 2(a) Creating Branch in the Repository

While in your repository click on the main tab and enter your branch name and hit the enter key or select the **create branch** option. {Type a suitable name}. You should see a screen something like this.

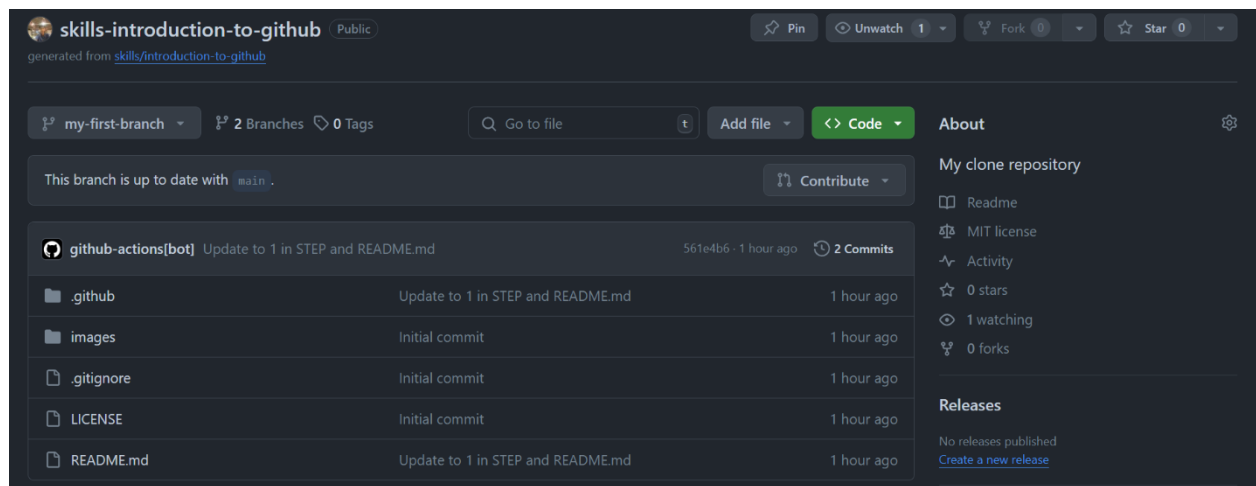


Figure: 2(b) my-first branch

When creating a branch, it duplicates all the content that is in the main branch to the branch you have created. You can create as many branches as you like. To go back to your main branch just use the drop down menu.

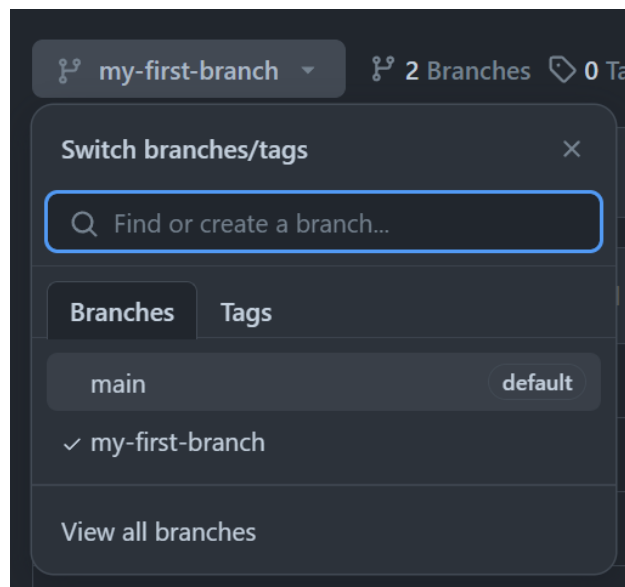


Figure: 2(c) my-first branch

Note that anything that is updated in the main branch will prompt a message in the other branches that it is x commits away (The x represents the number of commits in that particular branch) and anything that is updated in the other branch needs to be merged with the main branch to update the main repository.

## OBJECTIVE: 3 OPENING A PULL REQUEST

*Nice work making that pull request!*

Now that you have made a change to the project and created a commit, it's time to share your proposed change through a pull request!

**What is a pull request?:** Collaboration happens on a [pull request](#). The pull request shows the changes in your branch to other people and allows people to accept, reject, or suggest additional changes to your branch. In a side-by-side comparison, this pull request is going to keep the changes you just made on your branch and propose applying them to the main project branch.

You may have noticed after your commit that a message displayed indicating your recent push to your branch and providing a button that says **Compare & pull request**.

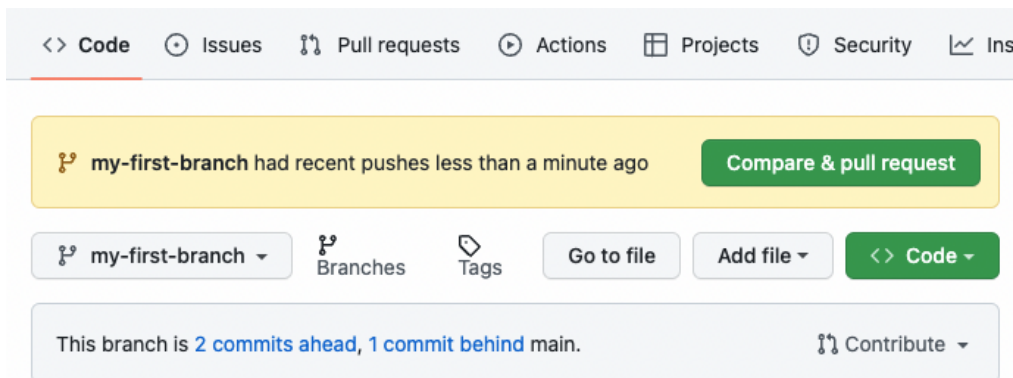


Figure: 3(a) Pulling a Request

To create a pull request automatically, click **Compare & pull request**, and then skip to step 6 below. If you don't click the button, the instructions below walk you through manually setting up the pull request.

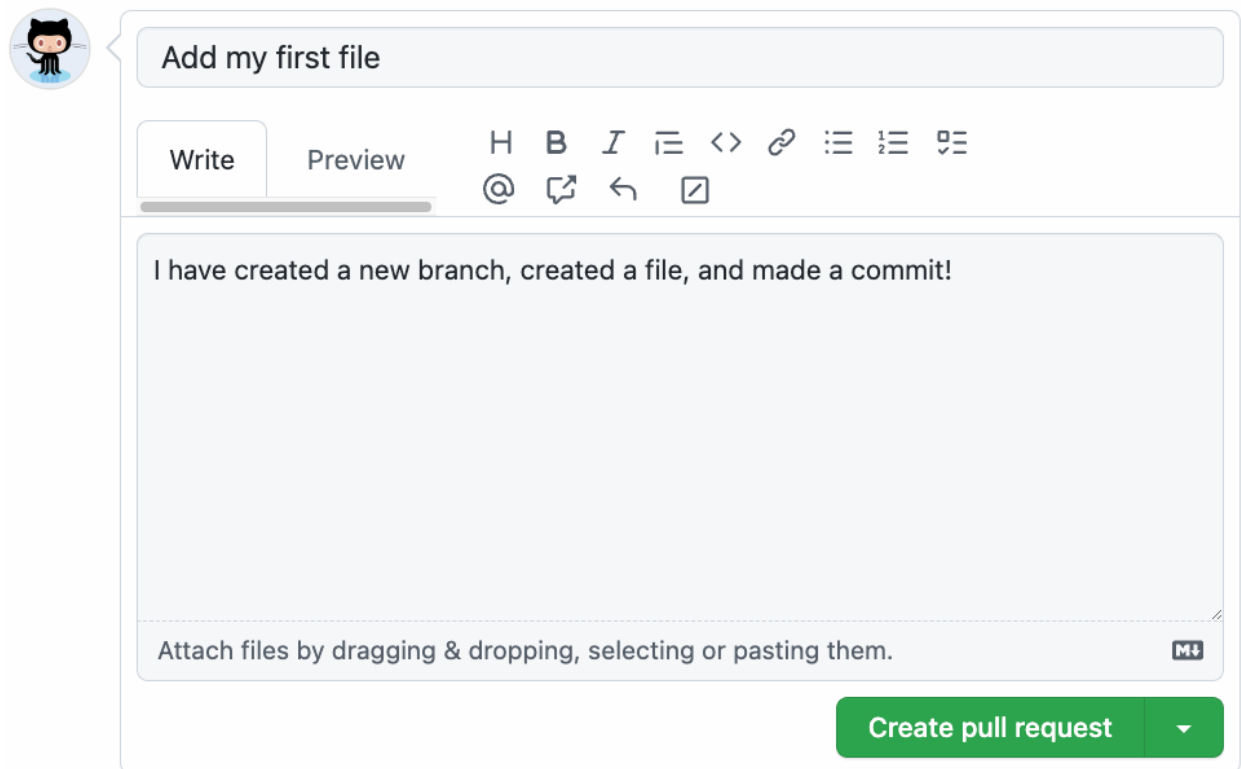
1. Click on the **Pull requests** tab in the header menu of your repository.
2. Click **New pull request**.
3. In the **base:** dropdown, make sure **main** is selected.
4. Select the **compare:** dropdown, and click my-first-branch.



Figure: 3(b) Merging Branches

5. Click Create pull request.
6. Enter a title for your pull request. By default, the title will automatically be the name of your branch. For this exercise, let's edit the field to say Add my first file.

7. The next field helps you provide a description of the changes you made. Here, you can add a description of what you've accomplished so far. As a reminder, you have: created a new branch, created a file, and made a commit.



The screenshot shows the GitHub 'Add my first file' dialog box. It features a GitHub Octocat icon in the top left corner. The title bar reads 'Add my first file'. Below the title bar, there are two tabs: 'Write' (selected) and 'Preview'. To the right of the tabs is a rich text editor toolbar with icons for heading (H), bold (B), italic (I), list (≡), code (<>), link (🔗), table (📊), quote (🗣️), and undo (↶). Below the toolbar is a large text area containing the text 'I have created a new branch, created a file, and made a commit!'. At the bottom of the text area, there is a placeholder text 'Attach files by dragging & dropping, selecting or pasting them.' and a small icon for attaching files. At the bottom right of the dialog box, there is a green button labeled 'Create pull request' with a dropdown arrow.

Figure: 3(c) New Branch Created

8. Click Create pull request. You will automatically be navigated to your new pull request



## OBJECTIVE: 4 MERGE A PULL REQUEST

**What is a merge?** A [merge](#) adds the changes in your pull request and branch into the main branch. For more information about merges, see "[Merging a pull request.](#)"

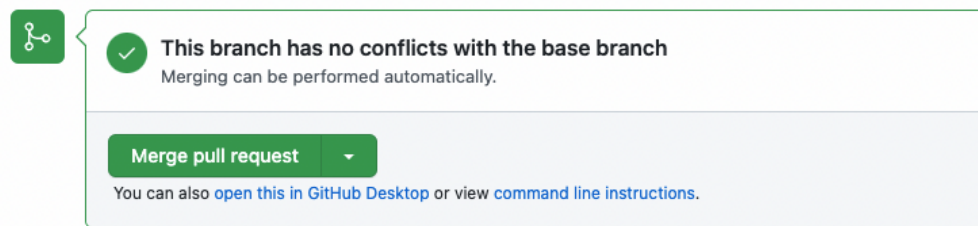


Figure: 4(a) Merge Pull Request

You would have seen a message like this. If no conflicts occur press the Merge pull request button. Click confirm. Once your branch has been merged, you don't need it anymore. To delete this branch, click **Delete branch**.

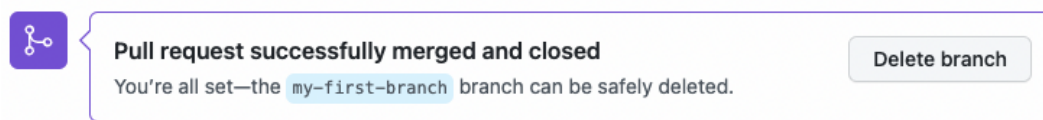


Figure: 4(b) Pull Request Merged Successfully

**Note:** deleting a branch helps to simplify your repository but keeping it is no harm as you can still reuse it for testing your work without modifying the main branch.

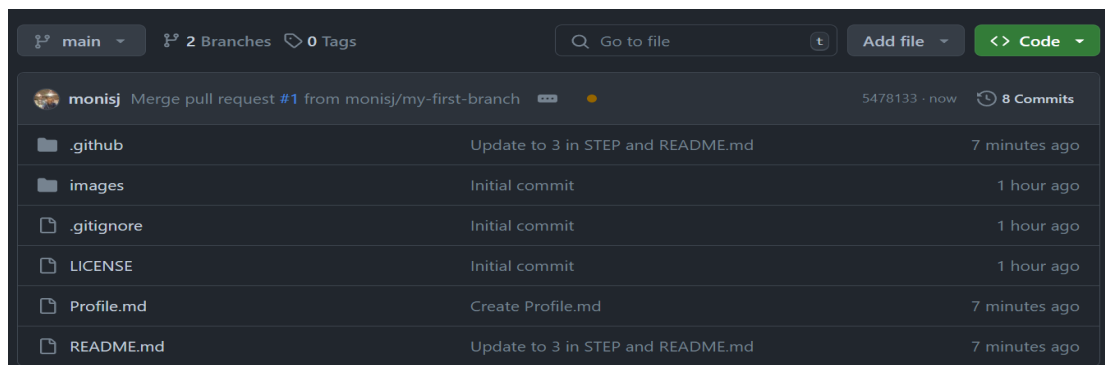


Figure: 4(c) Homepage of GitHub Profile

You should see that all the contents from the other branch are merged into the main branch.

### Some problems to solve:

- Create a branch with the name being your roll number.
- Create a Problem.md file which contains an item list in which an algorithm is defined for finding even and odd numbers.

- Create a pull request and merge the request as well. (Attach screenshots)

## OBJECTIVE: 5 INTRODUCTION TO INTEGRATED DEVELOPMENT ENVIRONMENT

---

- IDE stands for Integrated Development Environment. It is a software application that provides facilities for developing software.
- It consists of tools such as source code editor, automation tools, and debugger.
- For C language programming we will be using Dev-C++.
- Dev-C++ is a full-featured Integrated Development Environment (IDE) for the C/C++ programming language.
- Dev-C++ is free software and is distributed under the GNU General Public License. Thus we can distribute or modify the IDE freely.
- “Bloodshed Software” originally developed it, and Orwell has forked it after it was abandoned by Bloodshed in 2006.
- As similar IDEs, it offers to the programmer a simple and unified tool to edit, compile, link, and debug programs.

### INSTALLATION AND CONFIGURATION OF IDE

---

We can get the appropriate installable for dev-C++ IDE from <https://www.bloodshed.net/>

- Let’s see the entire installation process now. We have used the installable that comes along with the [C++ compiler](#).

**The stepwise installation for Dev-C++ is given below.**

- The first step while we start the installer is to select the language of our choice as shown in the below screenshot.

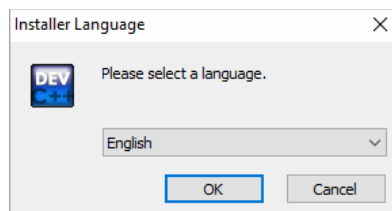


Figure: 5(a) Selecting Installation Language

- Once you select the appropriate language, you have to agree to the license agreement that pop-ups next.

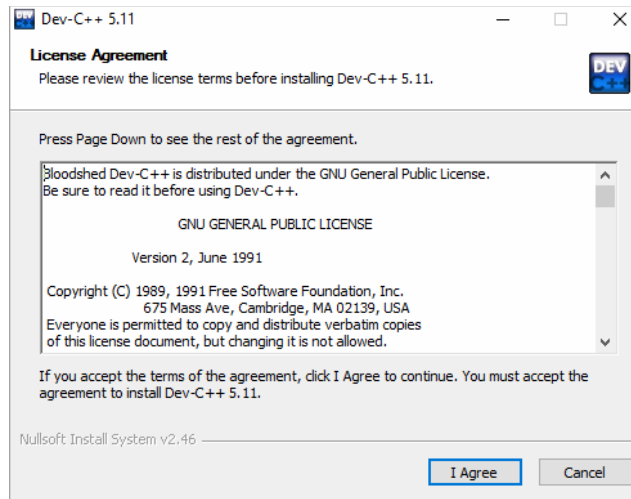


Figure: 5(b) Agreeing License Agreement

- Next, we are asked to select the components that we need to install as a part of the dev-C++ installation.

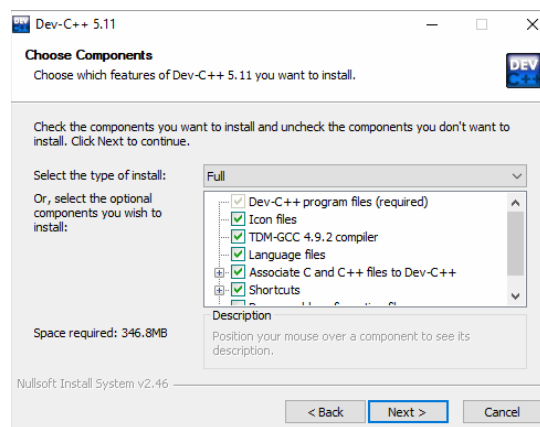


Figure: 5(c) Selecting type of Installation

As shown in the above screenshot, we are provided with a list of components available for installation and a checkbox against each component. We can check/uncheck each box to indicate which components to install. Click next once the components are selected.

- Now the installer prompts the user for the destination folder where the dev-C++ files/libraries etc. are to be copied.

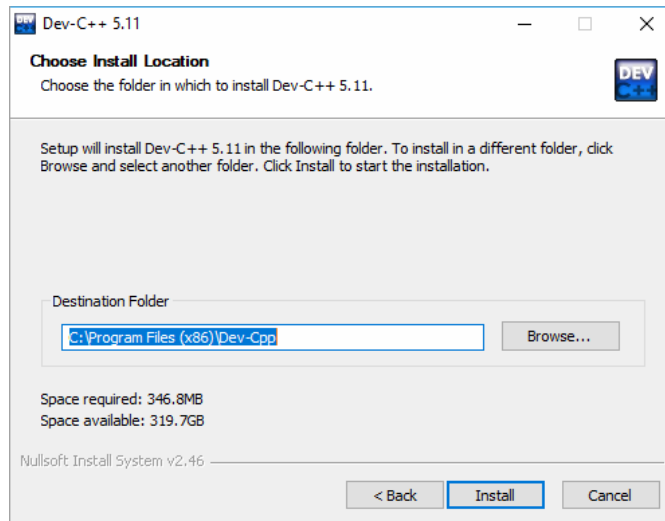


Figure: 5(d) Browsing Installation Location

Once we provide the destination folder path, click on Install.

## **DEVELOPMENT USING THE DEV C++ IDE**

---

### **CREATE A NEW PROJECT**

- To create a new project or source file in dev-C++ we need to follow the below steps:
- Click File -> New->Project
- Here, we can specify the project name. Make sure to select the “Empty Project” and also to check the “C++ Project” button.

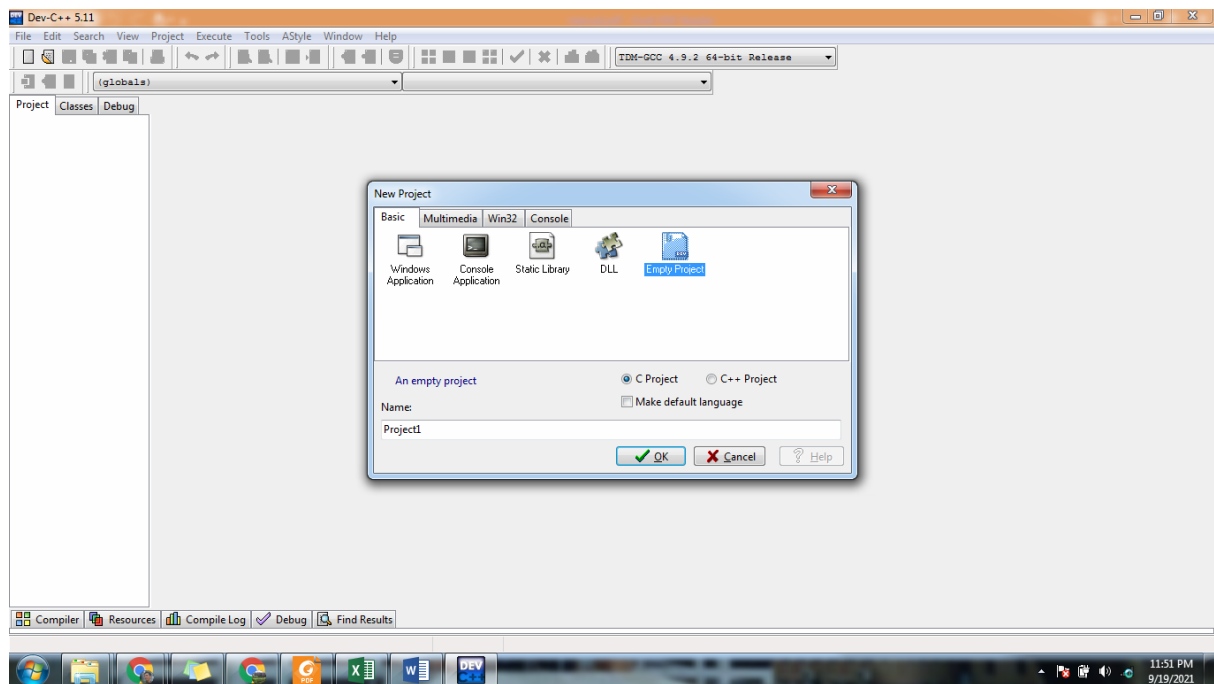


Figure: 5(e) Main Interface of Dev C++

- Once the entire information is provided, we can click ok and the IDE will ask for the path where the project is to be saved. When this is done, a workspace will open with the project explorer on the left-hand side that shows the project we just created.

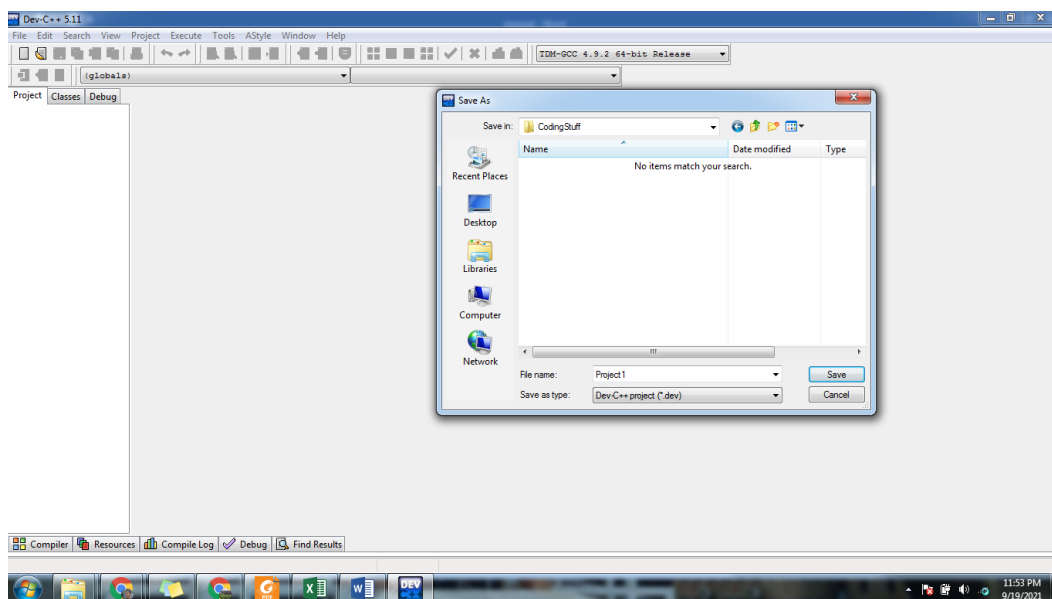


Figure 5(f) Saving Project

## ADD A SOURCE FILE TO A PROJECT

- Add a new file by clicking **Project ->New File** or Right-click on **Project Name** in the project explorer and click **New File**
- Save the file using CTRL+S or save option in file menu give it a name and choose extension c

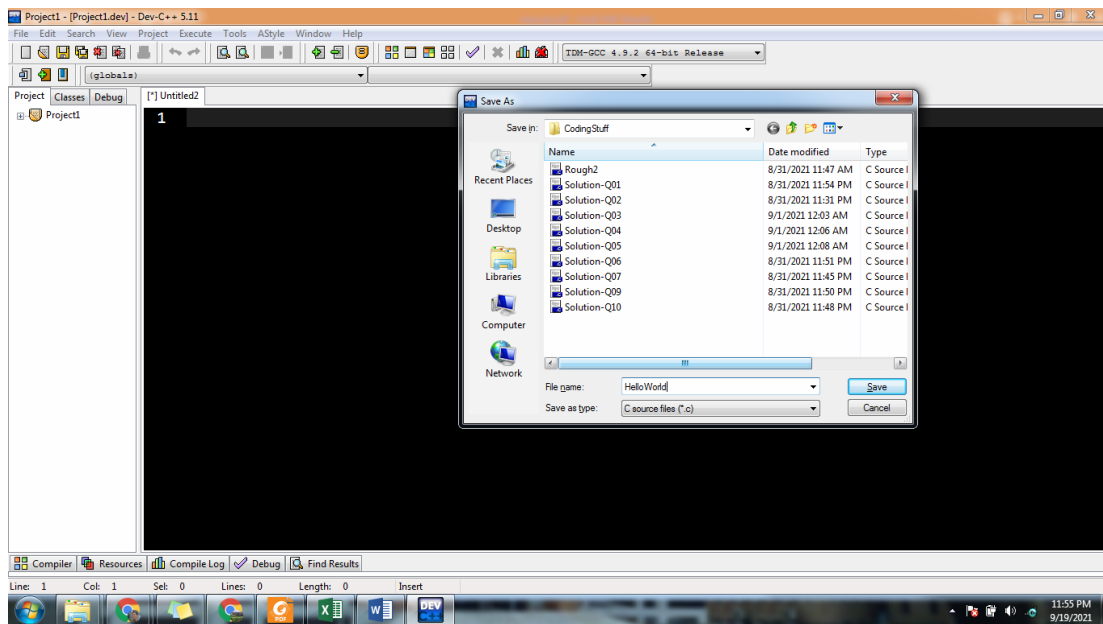


Figure: 5(g) Adding Source File to Project

## OBJECTIVE: 6 INTRODUCTION TO C PROGRAMMING LANGUAGE

### BASIC STRUCTURE OF C PROGRAM

- This is the traditional “Hello World” program in C language

```
1. #include<stdio.h>
2.
3. main(){
4. printf("Hello World");
5. }
```

### Understanding Basic Structure of C Program

- This program uses (that is, it ‘includes’) code from the C-language ‘standard input/output library, stdio, using this statement:

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

- The code that starts with the name `main` is the ‘main function’ – in other words, it is the first bit of code that runs when the program runs. The function name is followed by a pair of parentheses. The code to be run is enclosed between a pair of curly brackets:

```
05 main() {  
  
}
```

- In this case, the code calls the C `printf` function to print the string (the piece of text) between double-quotes.

```
06 printf("hello world");
```

## COMPILE/BUILD

---

When we have all the code ready for the project, we will now compile and build the project.

**Follow the below steps to build and execute the dev C++ project:**

- To compile the project, click **Execute -> Compile** (or click F9).
- We can see the compilation status in the “**Compile Log**” tab in the workspace.
- If there are any errors whether syntax or linker errors, then they will appear in the compiler tab.

Once the project is compiled successfully, we need to run it.

## EXECUTE PROJECT

---

- Click on **Execute ->Run.** ( or click F10)
- The console window that gives us the output will be shown in the below screenshot.

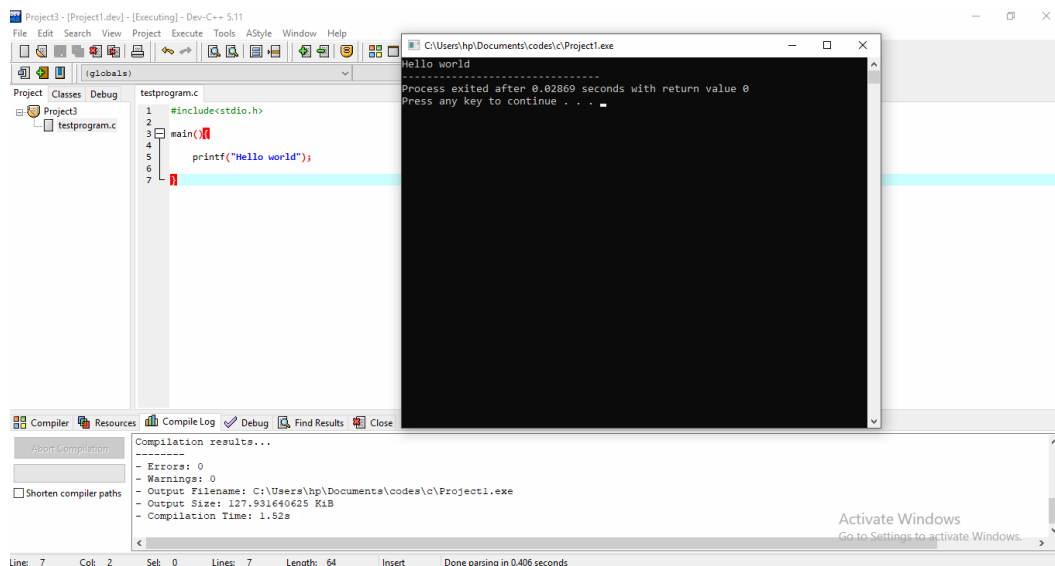


Figure: 6(a) Executing First Program

## FLOW OF EXECUTION OF C PROGRAM

- C is a high-level programming language developed in 1972 by Dennis Ritchie at AT&T Bell Laboratories.
- Programs in C typically go through six phases to be executed. These are: edit, preprocess, compile, link, load and execute.

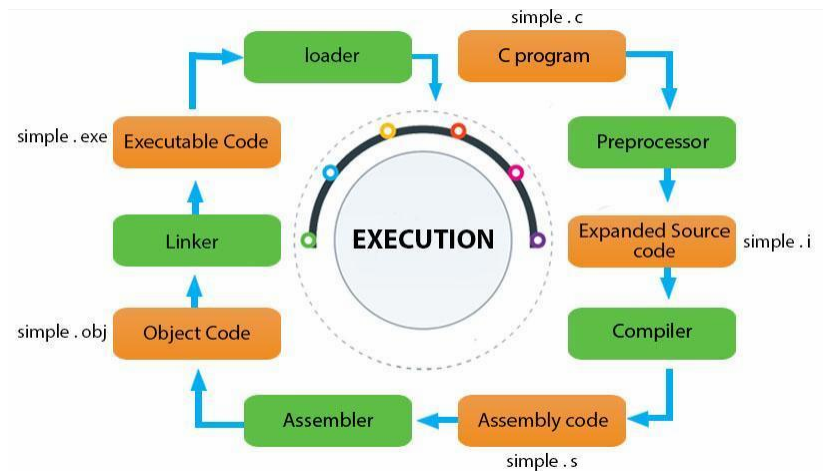


Figure: 6(b) Flow of Execution of C Program

1. C program (source code) is sent to **preprocessor** first. The preprocessor is responsible to convert preprocessor directives into their respective values. The preprocessor generates an expanded source code.
2. Expanded source code is sent to **compiler** which compiles the code and converts it into assembly code.

Computer programs are written using high-level programming languages. The source code is converted into machine-understandable machine code. A compiler is used for this conversion. Compiler is a translator that converts the source code from high-level programming language to a lower level machine language in order to create an executable program.

3. The assembly code is sent to **assembler** which assembles the code and converts it into object code. Now a simple.obj file is generated.
4. The object code is sent to **linker** which links it to the library such as header files. Then it is converted into executable code.

Library functions are not a part of any C program but of the C software. Thus, the compiler doesn't know the operation of any function, whether it is printf or scanf. The definitions of these functions are stored in their respective library which the compiler should be able to link. This is what the Linker does. So, when we write #include, it includes stdio.h library which gives access to Standard Input and Output. The linker links the object files to the library functions and the program becomes a .exe file. A simple.exe file is generated which is in an executable format.

5. The executable code is sent to **loader** which loads it into memory and then it is executed. After execution, output is sent to console.



Whenever we give the command to execute a particular program, the loader comes into work. The loader will load the .exe file in RAM and inform the CPU with the starting point of the address where this program is loaded.

## COMPILING A C PROGRAM WITHOUT IDE

We usually use a compiler with a graphical user interface, to compile our C program. This can also be done by using cmd.

**STEP 1:** Run the command 'gcc -v' to check if you have a compiler installed. If not you need to download a gcc compiler and install it. You can search for cmd in your windows system to open the command prompt.



```
Microsoft Windows [Version 10.0.17134.829]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\User> gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=C:/TDM-GCC-64/bin/./libexec/gcc/x86_64-w64-mingw32/5.1.0/lto-wrapper.exe
Target: x86_64-w64-mingw32
Configured with: ../../src/gcc-5.1.0/configure --build=x86_64-w64-mingw32 --enable-targets=all --enable-languages=ada,c,c++,fortran,lto,objc,obj-c++ --enable-libgomp --enable-lto --enable-graphite --enable-cxx-flags=-DWINPTHREAD_STATIC --disable-build-with-cxx --disable-build-poststage1-with-cxx --enable-libstdcxx-debug --enable-threads=posix --enable-version-specific-runtime-libs --enable-fully-dynamic-string --enable-libstdcxx-threads --enable-libstdcxx-time --with-gnu-ld --disable-werror --disable-nls --disable-win32-registry --prefix=/mingw64tdm --with-local-prefix=/mingw64tdm --with-pkgversion=tdm64-1 --with-bugurl=http://tdm-gcc.tdragon.net/bugs
Thread model: posix
gcc version 5.1.0 (tdm64-1)

C:\Users\User>
```

Figure: 6(c) Loading GCC Compiler using CMD

**Step 2:** Create a c program and store it in your system.

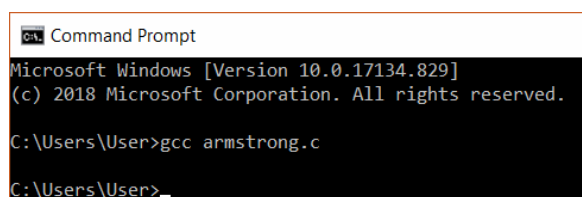
**STEP 3:** Change the working directory to where you have your C program. You can do that by using the command 'cd', which changes the directory. We need to pass the name of the directory in which the program is stored.

**Example:** cd Desktop

Our program is already in the user directory so we don't have to change it.

**STEP 4:**

The next step is to compile the program. To do this we need to use the command gcc followed by the name of the program we are going to execute. In our case, we will use Armstrong.c. Note that Armstrong.c is the name of the file.



```
Microsoft Windows [Version 10.0.17134.829]
(c) 2018 Microsoft Corporation. All rights reserved.

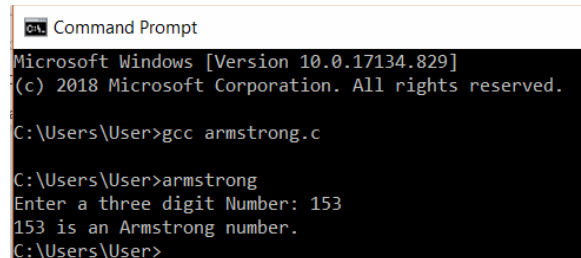
C:\Users\User>gcc armstrong.c
C:\Users\User>
```

Figure: 6(d) Compiling C Program using CMD

After this, an executable file will be created in the directory that your c file exists in. Eg: Armstrong.exe

#### STEP 5:

In the next step, we can run the program. This is done by simply giving the name of the executable file without any extension. On giving this we will get an output. Here, our Armstrong code is executed and we got output for this code.



```
Command Prompt
Microsoft Windows [Version 10.0.17134.829]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\User>gcc armstrong.c

C:\Users\User>armstrong
Enter a three digit Number: 153
153 is an Armstrong number.
C:\Users\User>
```

Figure: 6(e) Running C Program using CMD

## DATA TYPES IN C

- In C programming, data types are declarations for variables. This determines the type and size of data associated with variables.

Here's a table containing commonly used types in C programming for quick access.

Data Type	Description	Size(bytes)
int	Integers are whole numbers that can have both zero, positive and negative values but no decimal values.  It can take 232 distinct states from -2147483648 to 2147483647.	at least 2, usually 4
float	Floating type variables can hold real numbers precision of 6 digits	4
double	Floating type variables can hold real numbers with precision of 14 digits	8
char	Character data type allows a variable to store only one character.	1

bool	A boolean data type that can hold one of two values: true (1) or false (0).	1
void	A special data type that signifies the absence of a value. It is most often used with functions that do not return a value or with generic pointers.	

## VARIABLES IN C

---

- In programming, a variable is a container (storage area) to hold data temporarily.
- To indicate the storage area, each variable should be given a unique name (identifier). Variable names are just the symbolic representation of a memory location.

### Syntax:

A variable can be declared and initialized using following syntax:

```
datatype variable_name = value ;
```

### Example:

```
int playerScore = 95;
```

### Rules for naming a variable

- A variable name can only have letters (both uppercase and lowercase letters), digits and underscore.
- The first letter of a variable should be either a letter or an underscore.
- There is no rule on how long a variable name (identifier) can be. However, you may run into problems in some compilers if the variable name is longer than 31 characters.
- C is a strongly typed language. This means that the variable type cannot be changed once it is declared.

Note: You should always try to give meaningful names to variables. For example: firstName is a better variable name than fn.

## FORMAT SPECIFIERS

---

- Format Specifiers are strings used in the formatted input and output.
- The format specifiers are used in C to determine the format of input and output.
- Using this concept the compiler can understand that what type of data is in a variable while taking input using the scanf() function and printing using printf() function.

Some of the commonly used Format Specifiers are given below:

Format specifier	Description
%d or %i	It is used to print the signed integer value where signed integer means that the variable can hold both positive and negative values.
%u	It is used to print the unsigned integer value where the unsigned integer means that the variable can hold only positive value.
%o	It is used to print the octal unsigned integer where octal integer value always starts with a 0 value.
%x	It is used to print the hexadecimal unsigned integer where the hexadecimal integer value always starts with a 0x value. In this, alphabetical characters are printed in small letters such as a, b, c, etc.
%X	It is used to print the hexadecimal unsigned integer, but %X prints the alphabetical characters in uppercase such as A, B, C, etc.
%f	It is used for printing the decimal floating-point values. By default, it prints the 6 values after '.'.
%e / %E	It is used for scientific notation. It is also known as Mantissa or Exponent.
%g	It is used to print the decimal floating-point values, and it uses the fixed precision, i.e., the value after the decimal in input would be exactly the same as the value in the output.
%p	It is used to print the address in a hexadecimal form.
%c	It is used to print the unsigned character.
%s	It is used to print the strings.
%ld	It is used to print the long-signed integer value.

```

1  #include <stdio.h>
2
3  main() {
4      //printing character data
5      char ch = 'A';
6      printf("Printing character data: %c\n", ch);
7
8      //print decimal or integer data with d and i
9      int x = 45, y = 90;
10     printf("Printing decimal data: %d\n", x);
11     printf("Printing Integer data: %i\n", y);
12
13     //print float value
14     float f = 12.67;
15     printf("Printing floating point data: %f\n", f);
16
17     //print in scientific notation
18     printf("Print in scientific notation: %e\n", f);
19
20     //print in octal format
21     int a = 67;
22     printf("Printing data in octal format: %o\n", a);
23
24     //print in hex format
25     printf("Printing data in hex format: %x\n", a);
26
27     float z=3.8;
28     printf("Float value of y is: %g\n ", z);
29
30     printf("Address value of y in hexadecimal form is: %p\n", &y);
31
32     char str[] = "Hello World";
33     printf("%s\n", str);
34
35     printf("shift to the right 20 characters: %20s\n", str); //
36
37 }

```

Figure 6(f) Format Specifier Program in C

## Output

```

D:\Programs\Format Specifiers 2.exe
Printing character data: A
Printing decimal data: 45
Printing Integer data: 90
Printing floating point data: 12.670000
Print in scientific notation: 1.267000e+001
Printing data in octal format: 103
Printing data in hex format: 43
Float value of y is: 3.8
Address value of y in hexadecimal form is: 000000000022FE38
Hello World
shift to the right 20 characters:      Hello World

Process exited after 0.03353 seconds with return value 55
Press any key to continue . . .

```

Figure 6(g) Output of Format Specifier Program

## STANDARD INPUT AND OUTPUT STATEMENTS IN C

---

In C, standard input and output are handled by a set of functions from the `<stdio.h>` library, allowing programs to read data from the keyboard and display results on the console. These functions are categorized as either formatted (using format specifiers) or unformatted (for single characters or strings).

### Formatted I/O

#### Input: `scanf()`

The `scanf()` function is used to read and store formatted input from the standard input (`stdin`). Its syntax is `scanf("format string", &variable);`, where the format string specifies the type of data to be read.

**Syntax:** `scanf("format_string", &variable);`

- `format_string`: Contains format specifiers (e.g., `%d` for integers, `%f` for floats) and defines the layout of the input.
- `&variable`: The address of the variable where the input value will be stored.

### EXAMPLE

---

```
07 #include <stdio.h>
08
09 int main() {
10     int age;
11     float height;
12     printf("Enter your age and height: ");
13     scanf("%d %f", &age, &height);
14     printf("You are %d years old and %.2f meters tall.\n", age,
15     height);
16     return 0;
17 }
```

#### Output: `printf()`

The `printf()` function is used to write formatted output to the standard output (`stdout`).

**Syntax:** `printf("format string", arguments);`

- `format_string`: A string containing literal text, format specifiers, and escape sequences.
- `arguments`: Variables or values that correspond to and replace the format specifiers.

## EXAMPLE

---

```
17 #include <stdio.h>
18
19 int main() {
20     int number = 10;
21     char character = 'A';
22     printf("The number is %d and the character is %c.\n", number, character);
23     return 0;
24 }
```

## Unformatted I/O

### Character I/O: getchar() and putchar()

- `getchar()`: Reads and returns a single character from the input.
- `putchar()`: Writes a single character to the output.

## EXAMPLE

---

```
25 #include <stdio.h>
26
27 int main() {
28     char ch;
29     printf("Enter a character: ");
30     ch = getchar(); // Read a single character
31     printf("You entered: ");
32     putchar(ch); // Print a single character
33     putchar('\n'); // Print a newline
34     return 0;
35 }
```

### String I/O: gets() and puts()

- `gets()`: Reads a line of text (including spaces) from `stdin` into a character array. This function is deprecated and unsafe due to buffer overflow vulnerabilities. Use `fgets()` instead.
- `puts()`: Writes a string to `stdout`, automatically adding a newline character at the end.

## EXAMPLE

---

```
36 #include <stdio.h>
37
38 int main() {
39     char name[50];
40     printf("Enter your name: ");
```

```

41     fgets(name, 50, stdin); // Safer alternative to gets() 42
    printf("Hello, ");
43     puts(name); // Prints the string with a newline
44     return 0;
45 }

```

## ESCAPE SEQUENCE

- Escape Sequences are non-printing characters.
- When a character is preceded by a backslash (\), it is called an escape sequence and it has a special meaning to the compiler. For example, \n in the following statement is a valid character and it is called a new line character.

Escape Sequence & Description
\t Inserts a tab in the text.
\b Inserts a backspace in the text.
\n Inserts a newline in the text.
\r Inserts a carriage return in the text.
\f Inserts a form feed in the text.
\' Inserts a single quote character in the text.
\" Inserts a double quote character in the text.
\\ Inserts a backslash character in the text.
\? Inserts a question mark in the text.



\a

Play beep or Alarm

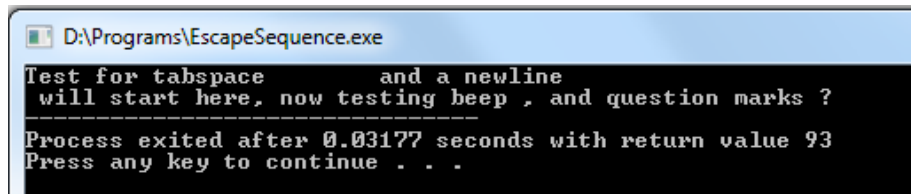
## EXAMPLE

---

```
1 #include <stdio.h>
2 int main() {
3     char ch1='\t';
4     char ch2 = '\n';
5     printf( "Test for tabspace %c and a newline %c will start here, now testing beep \a, and question marks \?", ch1, ch2);
6
7 }
```

Figure: 6(h) Escape Sequence Program in C

## Output:



```
D:\Programs\EscapeSequence.exe
Test for tabspace      and a newline
will start here, now testing beep , and question marks ?
-----
Process exited after 0.03177 seconds with return value 93
Press any key to continue . . .
```

Figure: 6(l) Output of Escape Sequence Program in C

## PRECISION SETTING IN C

---

Precision is specified by the number of digits after the decimal point for the outputs for float as well as double numbers.

If precision is not specified, it would be according to the default setting in the computer which is generally 6 digits.

The precision may be specified in the format specifiers place by a period(.) followed by a positive number equal to the number of digits desired.

## EXAMPLE

```
#include<stdio.h>

int main()
{
    double a=2.55555588889999;

    printf("before setting the precision\nnumber is: %lf",a);

    printf("after setting the precision\nnumber is: %.14lf",a);
}
```

Figure: 6(j) Precision Setting Program in C

### Output:

```
before setting the precision
number is: 2.555556after setting the precision
number is: 2.55555588889999
-----
Process exited after 0.04312 seconds with return value 55
Press any key to continue . . .
```

Figure: 6(k) Output of Precision Setting Program in C

The above will display 14 digits after the decimal point after setting the precision otherwise it will display only 6 characters even though double datatype precision is 14 digits as seen in datatypes section.

## Lab 03 – Class Tasks

### Task 1: GitHub Profile Setup

Create a GitHub account (if you don't already have one) and update your profile by adding your full name, a short bio, and a professional profile picture.

### Task 2: First Repository

Create a new repository named **PF-Lab03-[YourRollNumber]**. Add a **README.md** file describing yourself and your programming interests.

### Task 3: Branch and Commit

Create a new branch in your repository named **lab03-branch**. Inside it, create a file named **aboutme.md** containing your name, degree program, and one hobby. Commit the file.

### Task 4: Pull Request and Merge

Raise a pull request to merge your **lab03-branch** into the **main** branch. Successfully merge and delete the branch after merging.

### Task 5: Markdown Formatting

Edit your **README.md** to include at least:

- One heading
  - One ordered list
  - One unordered list
  - One bold + italic statement
- Commit the changes.

