Human Computer Interaction

Instructor: Rajagopalan Jayakumar

SOEN 6751

Assignment 2

:Team Members:

ID	Name
40071697	Bhavpreet Kaur
40080104	Subhodip Ray
40059367	Subhannita Sarcar
40088890	Nancy Goyal
40092781	Roohani Naik
40094053	Hesamedin Dadgar

Table of Contents

Conceptual Designs

Design 1

1. Transforming the requirements into conceptual design

- ❖ Selection of user type
  - ➢ **What does it do?**
    It modifies the interface and further functionalities according to the selected user type among the three available choices (Novice, Typical, Expert).
  - ➢ **How does it behave?**
    When the user selects a user type, the corresponding functionalities are made available in the view and other higher functionalities are grayed out unless the user type is changed.
  - ➢ **What does it look like?**
    The software presents three user modes - Novice, Typical and Expert on installation of the software. This user mode is set to be the default user mode throughout the lifetime of the installed instance, unless the user selects 'Options' in the top menu bar and chooses to change user mode. At this point, the current window will prompt for saving the written program (if there is any written unsaved changes), close the current window and launch a new instance.
- ❖ Write, save, link and execute code
  - ➢ **What does it do?**
    It facilitates the user to edit code in the editor, save the code in a named file with the proper extension, run the code which is a combination of linking the code and compiling with the GCC compiler.
  - ➢ **How does it behave?**
    When the user writes the code using standard input methods, the written code becomes visible on the editor. The save option when selected will save the written code in memory. The run option when selected will link and execute code.
  - ➢ **What does it look like?**
    The application has a simple text area where the written code becomes visible. The code is also colour formatted,i.e., identifiers, keywords and literals appear in different colours. In order to save the code, the user has to select File in the top menu bar and then select save option. A window prompts the user to set the filename where the code will be saved. The run button is also visible in the top menu bar, which executes the code and displays compilation messages in the terminal that open up in a different window.

- ❖ Debug code

- ➢ **What does it do?**
  This feature debugs the written code and lets the user know of any compilation messages.
- ➢ **How does it behave?**
  The debug feature identifies the errors in the code using the debug options that are set by the user. It uses the GCC compiler to check for errors in the code.
- ➢ **What does it look like?**
  The top menu bar has an icon which when selected, will prompt the user to enter the debug options. Once the user confirms the debug options, the debug message will be displayed on the terminal.

❖ Generate code

- ➢ **What does it do?**
  This feature generates generic code snippets for reuse.
- ➢ **How does it behave?**
  When the generate code feature is used, a code snippet corresponding to the option selected will appear automatically on the editor.
- ➢ **What does it look like?**
  The user has to select options from the top menu bar and then select the Generate code option. A second drop down menu will display the types of codes that can be generated (if-else, for loop, etc). Once the code type is selected, the corresponding snippet will be visible in the text area where code is written.

❖ Optimize code

- ➢ **What does it do?**
  This feature makes the code more efficient and makes sure it adheres to coding conventions.
- ➢ **How does it behave?**
  It detects code smells in the written code and modifies the code accordingly to display the most efficient code.
- ➢ **What does it look like?**
  The user has to select options from the top menu bar and then select the Optimize code feature. Once this is selected, the written code in the text area will change itself to show the optimized code.

❖ Developer options

- ➢ **What does it do?**
  It adds new developer options and makes them available for implementation on the written code.
- ➢ **How does it behave?**
  Once the user adds and defines a new developer option, the compiler saves it for future use. When the user selects the newly added developer option to be implemented, the compiler executes the code according to the new options.

- ➢ **What does it look like?**

The user must select developer option and will be prompted to either add a new developer option or use from the list of already existing developer options. If an existing developer option is selected, the compiler runs the code using this feature. If the user chooses to add a new developer option, he/she is prompted to enter the acronym and the definition of the developer option

2. Selection of interface metaphor

Metaphors	**Description**
Plus sign ‘+’	To create a new file
Two overlapping rectangles	Open an existing file
Question mark ‘?’	Open a detailed section outlining the functionality of the system, i.e., help section
Grayed out sections	To indicate disabled options
Green triangle on button	Execute the written code
Bug icon on button	Debug the code
Vertical sliding bar with arrows on either end	Drag to scroll the text area or use the arrow buttons at either ends to move up or down the text area

3. Selection of interaction type

The **instructing** interaction type is used in this interface since the user is clearly instructing the application to get tasks done. For example, the user selects the save option that is visible on the interface to let the application know that this file is to be saved.

Design 2

1. Transforming the requirements into conceptual design

- Selection of user type
    - **What does it do?**
      This screen will give an option to choose a default user type for the first time based on the level of the skills of the user.
    - **How does it behave?**

The screen will take the input from the user to choose its default user type and will save it accordingly for further presenting the options according to the chosen type.

- ➢ **What does it look like?**
  The interaction will have three radio buttons i.e Novice User, Typical User, Expert User. It will also show a bar showing the scale from where the user can get an idea on which option to choose.

❖ Create or Load File

- ➢ **What does it do?**
  This screen will ask the file name and its location from the user. The user can either give a new file name and select the path where to save the file or can load an existing file.
- ➢ **How does it behave?**
  Depending on the user's choice, if the user enters a filename that does not exist before, it will save the file at the chosen location. If the user selected the existing file location from browse, it will populate the text box with the selected file name.
- ➢ **What does it look like?**
  There will be one label for file name and a text box in front of it. Below it, there will be another label as location and a text box where the location will be populated when selected through the browse button.

❖ Main Module

- ➢ **What does it do?**
  When the user selects a skill level, an according home screen with all the required options is loaded. The screen has a header with the options based on the chosen skill level. Further the screen is divided into two parts with one footer. The footer allows you to add or remove options with a checkbox and an option panel to change the user type, on extreme right, which will take the user to a new screen with options to decide on user type. The middle part of the screen is divided into two panels, the left part of will always load with basic program structure of C/C++. The user can write or load the code in this section of the screen. The right part is for showing the results of the compiled code.
- ➢ **How does it behave?**
  The functionality of various different options on the top menu bar is described below-

- **New File -** This option will allow the user to create a new file with default name or with a name of his choice.
- **Compile -** This option will compile the written program on the left pane and will show the errors if found any, otherwise it will give a message compiled successfully.
- **Run -** After successfully compiling the program, this option can be used. It will run the compiled code and will give the output of the program accordingly in the second window.

- **Debug -** This option can step through the source code line-by-line or even instruction by instruction. You may also watch the value of any variable at run-time. In addition, it also helps to identify the place and the reason making the program crash.
- **Generate Code** - This option will generate default code snippet to be written in a file like default class structure, main function, constructors etc.
- **Optimize Code -** This feature will allow the user to make the written code more efficient and more formatted. It can allow renaming a variable name in the whole file.
- **Save** - This feature will allow the user to save the current code written in the writing code area.
- **All Options -** This feature will list all the options which are not available to novice user type by default so that a user can add it into his/her screen by enabling a particular option.
- **Change User Type -** This feature will provide the user to change his default settings according to his/her improved skills or if he/she needs to lower it. The user can change his/her user type and accordingly those advanced features will appear to use.

➢ **What does it look like?**

The main module is divided into three parts. Top bar will provide options according to the user type selected. Options like new file, compile, run, debug, save will be loaded as default in all user types. Generate code and optimize code option buttons will be loaded for typical and expert users only. Developer options are loaded only for the expert users. The footer panel of the screen will have the option of changing the user type.

2. Selection of interface metaphor

The following interface metaphors were chosen keeping in mind the familiarity and description provided by the user in the requirement gathering. Icons and graphics are used predominantly because they are illustrious and understandable.

Metaphors	Description
● Emoticons ● Bar chart	To help users to choose the appropriate skill level, when choosing the user type
● Tooltips	For all text boxes, to help users understand what is needed
● Note box/ symbols	To provide advice before a decision. Eg. ● when selecting a user type, there would be note for the user about the option to change user type afterwards ● When creating or saving files, a note about the default location would be

	provided.
Different shapes on button ● Green triangle ● Yellow circle ● Red square ● Blue ant	These shapes on the buttons would help user understand the actions they take ● Green triangle : Run ● Yellow Circle : Compile ● Blue ant: Debug ● Red square: Stop debug
Colored buttons with graphics ● Save blue button ● Folder button ● Blank page with a star in the corner	These graphics are popular and are usually known to the user which would aid the user to understand the usage of the buttons

3. Selection of interaction type

The **instructing** interaction type was selected because in this system, the users are the one having a dialog with the system. Users are instructing the system in a number of ways: typing command shortcuts, or codes, selecting options from menus or check boxes etc.

Design 3

1. Transforming the requirements into conceptual design

❖ Welcome Page

➢ What does it do?

Shows a greeting message and Smart GCC logo and has a button to let th
user skip right into the main page or input their username/password or to the
signup page

➢ How does it behave?

If the User previously set up their profile and logged in before at least once or they can input their credentials to go to their previously setup home page or else take them to the signup page

➢ What does it look like?

Very standard login page with a flat color background and the transparent logo on it and two labeled empty spaces for login credentials with a signup button and a straight forward arrow to log in

❖ Sign up

➢ What does it do?

Ask some basic information from user like their name and preferred username and to set their experience level and come up with a password to complete their profile

➢ How does it behave?

It stores the inputs that user gave and saves them in memory so their profile is distinguished from others and load the appropriate version of the interface for them depending on the level of experience they put in their profile

➢ What does it look like?

An empty form with normal empty spaces for name and username input , a choice selection with a small description for their experience level and a pre check space for password input that doesn't allow passwords weaker than standard

❖ Home page

➢ What does it do?

It's the main interface and it should be specific for each type of user depending on their skill level, it allows user to have some options on buttons at the ready like run code, debug code, save, load, export, import, etc…. and it has a workspace that user can input their code in with a small console at the bottom for error reporting and debugging

➢ How does it behave?

The commands on the drop down will specifically do as their name for ex : undo will undo the last taken action . debug : will run a debug on the written code and so on and the run the code button will open a command console to show the user the results of the running code if it has no errors

➢ What does it look like?

The tabs on the top menu would be a drop down with a header and options relevant to that header with one click access in them, buttons would be just below the top headers and the bottom console can be modified in size for better observation

❖ Developer options

➢ What does it do?

It allows the user to access higher functions and options within the Smart GCC for special use

➢ How does it behave?

It will run functions like code optimization and code refactoring that are normally not required for everyday uses of novice and typical users

➢ What does it look like?

It would be the last header on top of the interface and a drop down with single click access

❖ Add / Remove option

➢ What does it do?

It allows user to add any extra options that they don't have in their interface easily accessible for future use

➢ How does it behave?

    It will store the selected option in their profile memory and adds the option as a new button the next time they log in with their profile
  - ➢ What does it look like?
    On the developer options drop down when the user right clicks on a command it would be a pop up on the cursor that has an add button on it
- ❖ What is this?
  - ➢ What does it do?
    It will give a short and brief description on a command for less experienced users.
  - ➢ How does it behave?
    It will describe in a single line what that command does and a small example
  - ➢ What does it look like?
    When the cursor moves on a command and right click on it, it would be in shape of a question mark

2. Selection of interface metaphor

Metaphors	**Description**
● Right sided arrow	Logging on with the input username and password
● Small ant	To debug the code the user wrote in the workspace
● Red exclamation mark	For errors the user encounters in their code
● A red pentagon	When they pressed the run button and the code is running the ant button will turn into a red pentagon
● Stop sign	For stop running a code which will turn the run button to ant icon and close the run console that shows the results
● Yellow light bulb	For small errors that does not risk the runnability of the code

3. Selection of interaction type

The “Instructing interaction type” would be used in this design since every command that users runs has a “what is this” feature and the interface contains a debug feature which allows them to single out their mistake and redo the incorrect part.

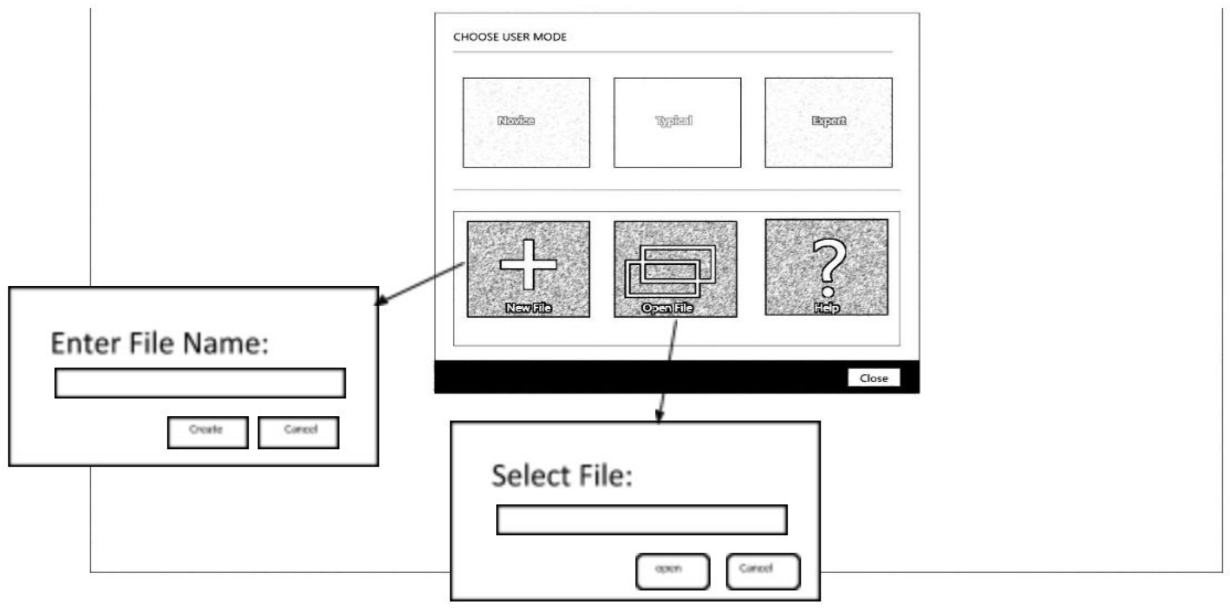
Prototypes

Prototype 1

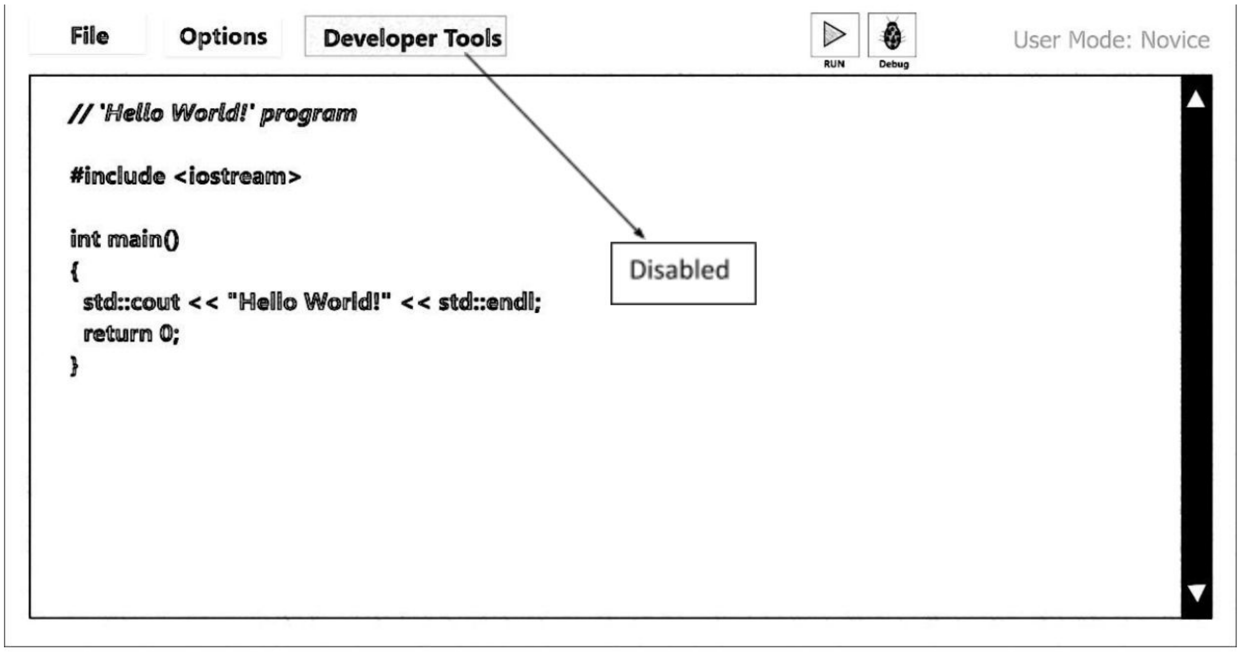
CARD BASED PROTOTYPE

The card based prototypes organizes the entire information into chunks of content that is easy to understand. This is especially useful in showcasing a prototype of Smart GCC because the entire application can be broken down into the possible screens that the user may see depending on the functions that are used. Therefore, the reason behind choosing card based prototype for the first prototype is to represent each component of this interaction design and resolve any issues. It will also serve as a reflection and help support alternative design ideas.

Home screen -

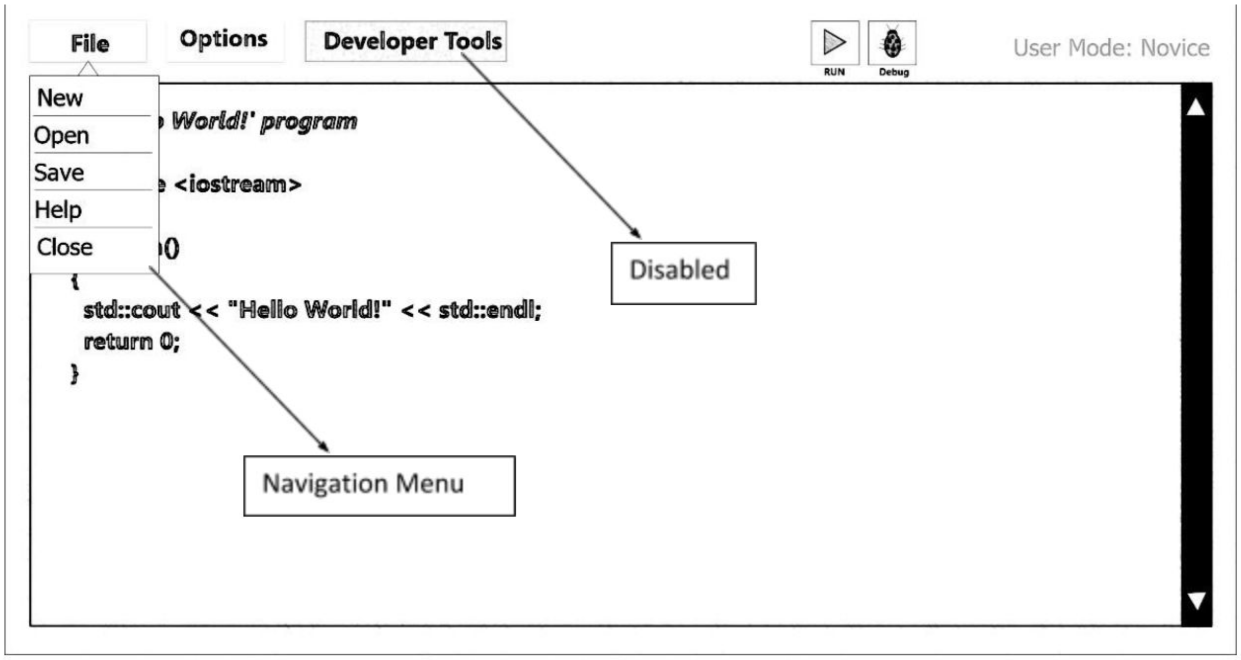


The main editor page for novice user -



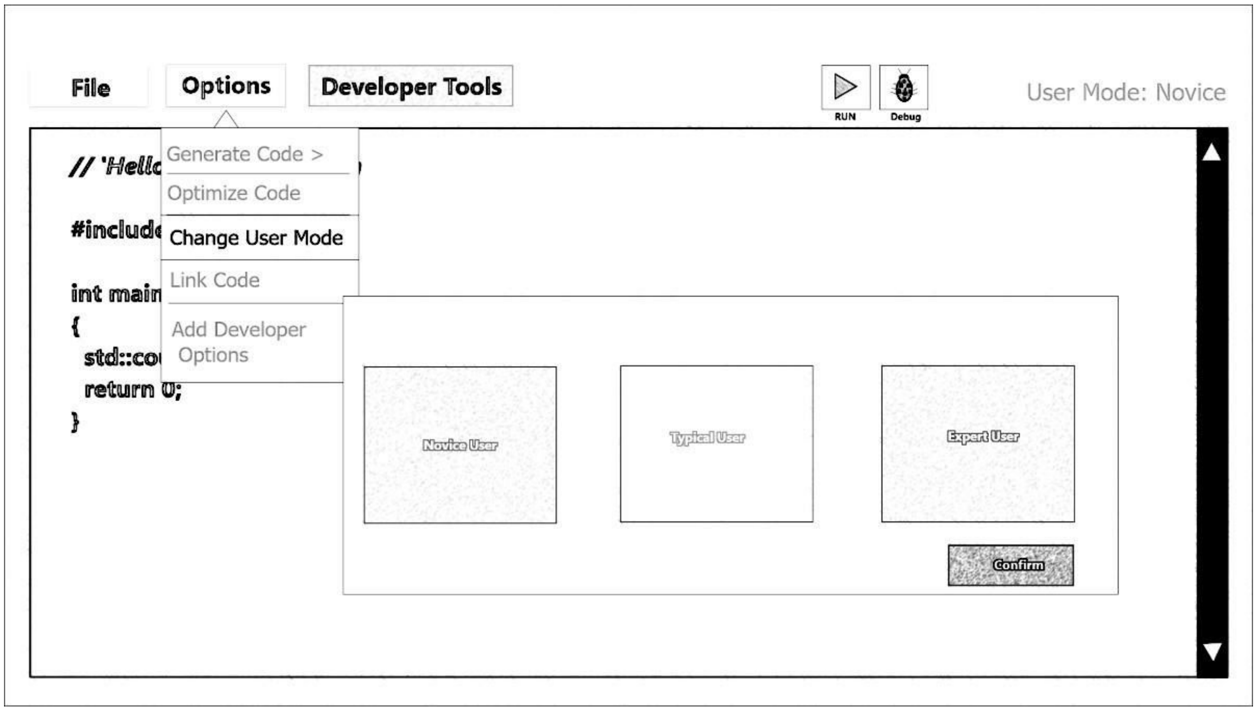
```
// 'Hello World!' program

#include <iostream>

int main()
{
  std::cout << "Hello World!" << std::endl;
  return 0;
}
```

File menu for novice -

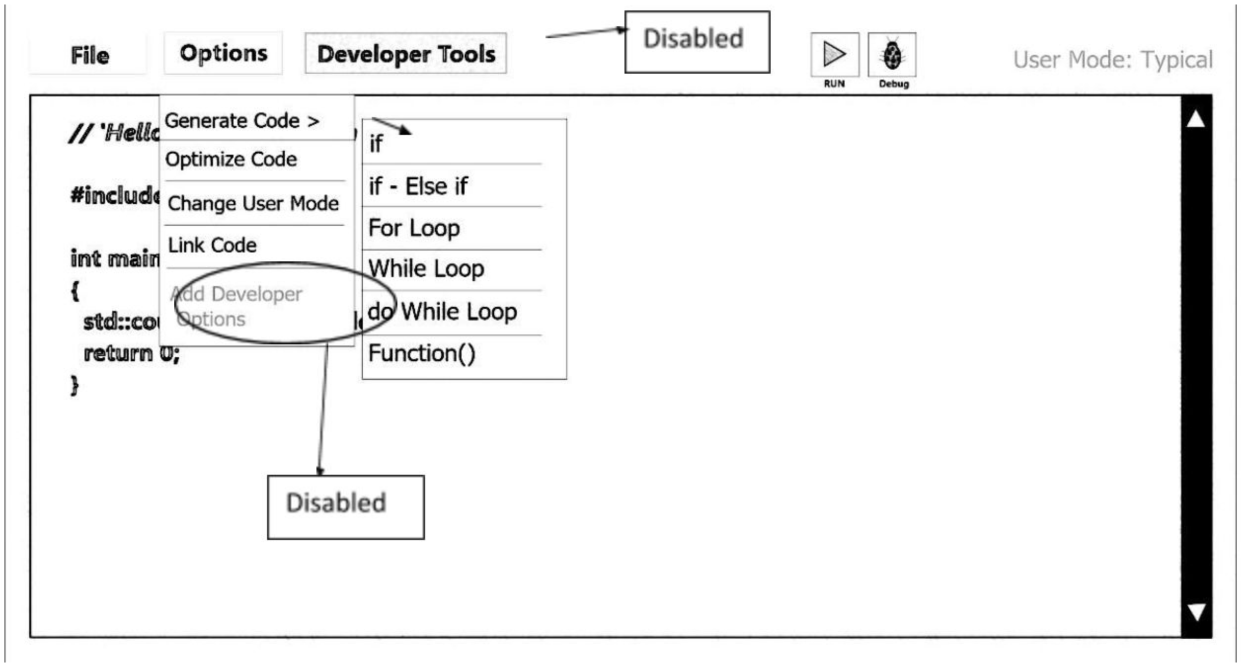


```
World!' program

e <iostream>

0
{
  std::cout << "Hello World!" << std::endl;
  return 0;
}
```

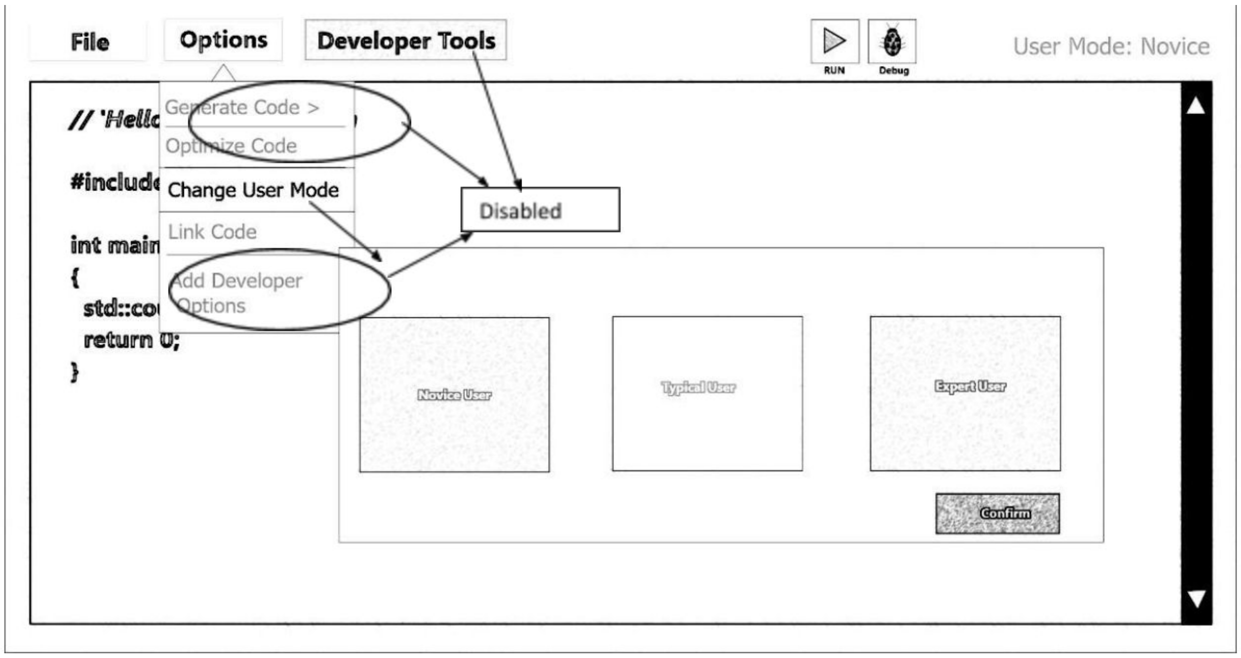
Options menu for novice -



Options menu for typical user -

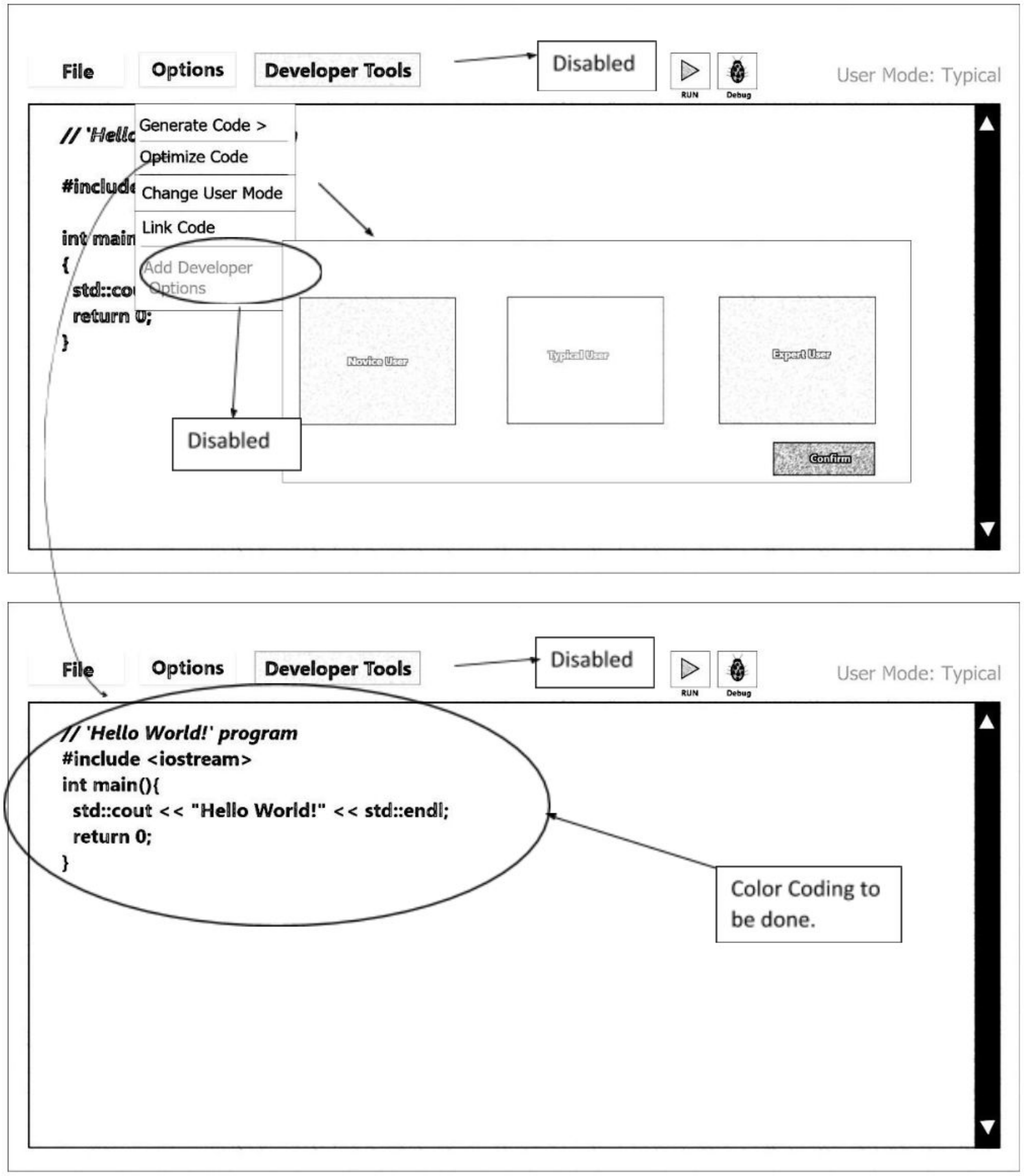


Changing the user type for novice user -



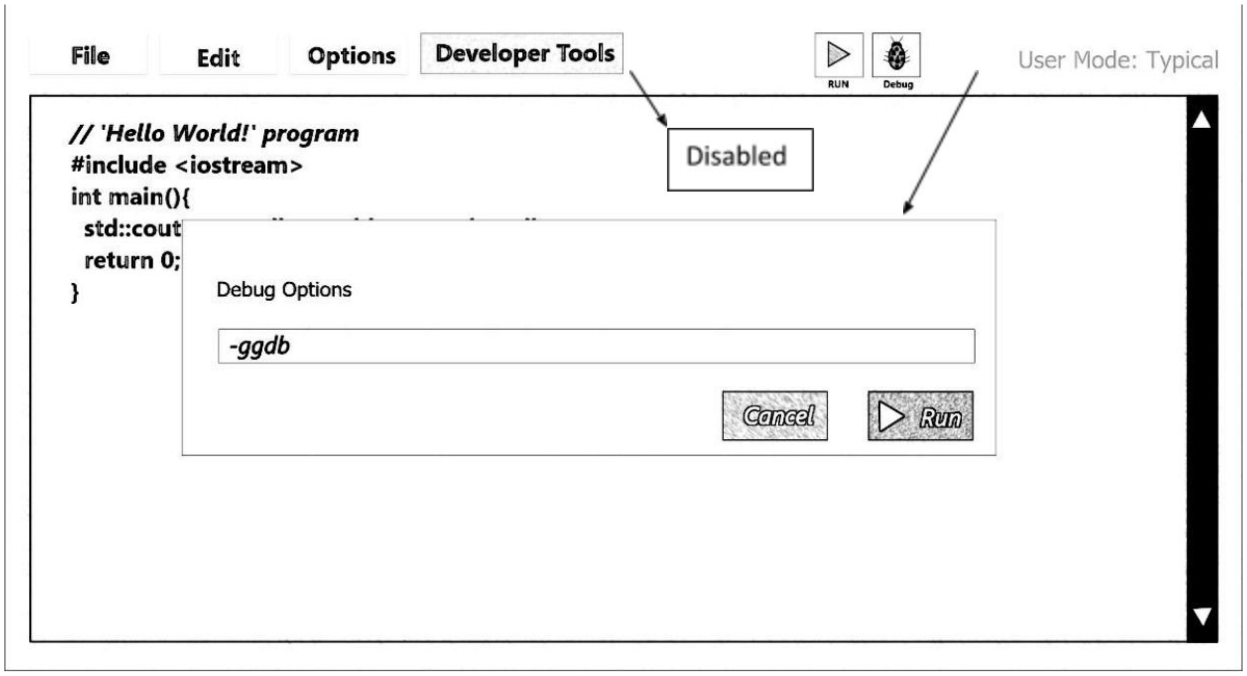
```
// 'Hello
#include
int main
{
  std::co
  return 0;
}
```

Optimizing code for typical user -



```
// 'Hello World!' program
#include <iostream>
int main(){
  std::cout << "Hello World!" << std::endl;
  return 0;
}
```

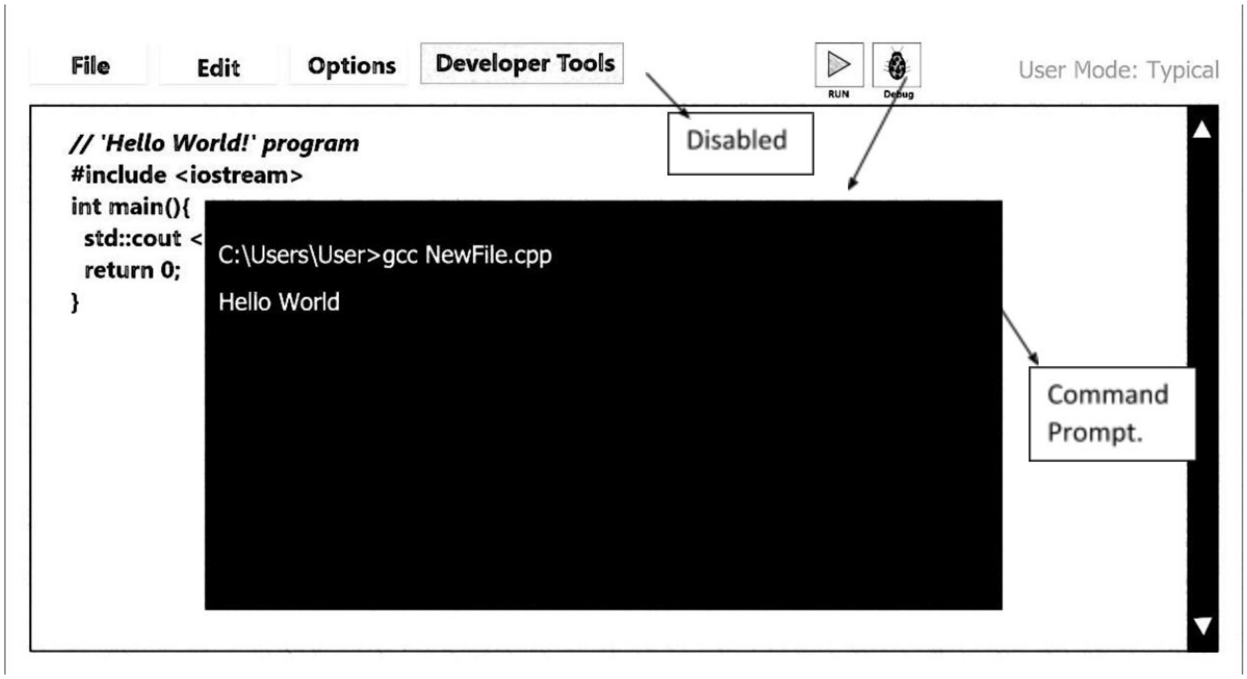
Debugging for typical user -



```
// 'Hello World!' program
#include <iostream>
int main(){
  std::cout
  return 0;
}
```

```
-ggdb
```

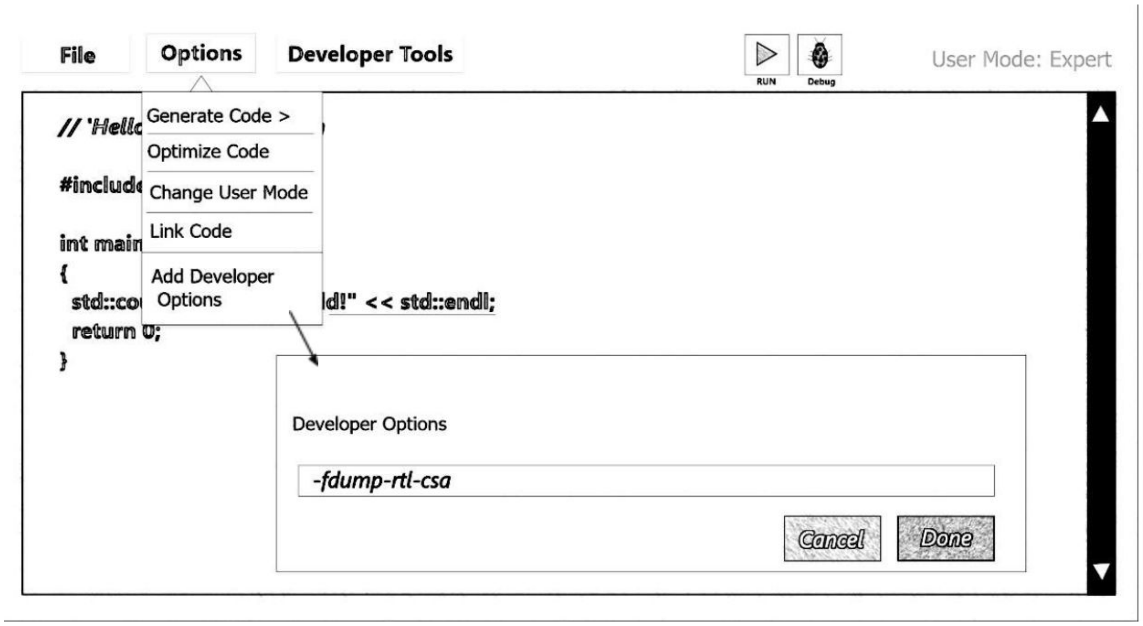
Executing code for typical user -



```
// 'Hello World!' program
#include <iostream>
int main(){
  std::cout <
  return 0;
}
```

```
C:\Users\User>gcc NewFile.cpp
Hello World
```

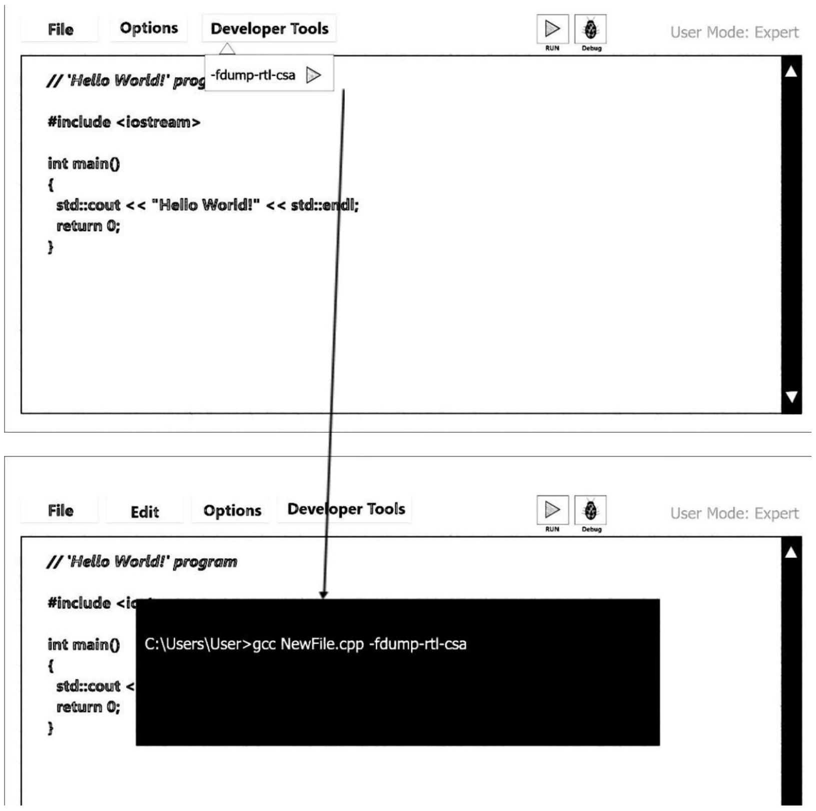
Adding developer options for expert user -



```
// 'Hello
#include
int main
{
  std::co                        ld!" << std::endl;
  return 0;
}
```

-fdump-rtl-csa

Implementing developer options for expert user -



```
// 'Hello World!' program

#include <iostream>

int main()
{
  std::cout << "Hello World!" << std::endl;
  return 0;
}
```

```
C:\Users\User>gcc NewFile.cpp -fdump-rtl-csa
```

1. Design usability

- **Usability goals**

- **Effective** : The application SmartGcc is efficient and effective in its usage, it properly takes the inputs from the user and responses to the requests made by the user and runs the code written by invoking the GCC compiler.
- **Safety** : The application never exits directly when the close button is clicked, if the file is not saved, it prompts the user to confirm exiting without saving or to save the file.
- **Utility** : This application can be used to write code to run in the GCC compiler, this application gives a better graphical user interface to the GCC compiler, where in various user groups are targeted such as Novice Users, Typical Users or the Expert Users. Different user groups have different requirements, while certain functionalities have been assigned to these user groups, if any user wants to access functionalities of another user group, it is possible to change the user mode from within the application. Therefore increasing the usability potential of the application.
- **Learnability**: The application SmartGcc is very easy to use, it includes generic symbols and proper text labels to define each button and also includes a very small amount of learning curve, in case of difficulty in using, help document is provided.
- **Memorability**: The users of the application can choose to save their application and keep a copy with them. Also the expert user can add Developer Options, which can be accessed at a later time as well.

Usability principles

- **Error prevention** : The system is designed to be tolerant towards errors such as users will be prompted to save the code before closing the application.
- **Consistency** : The application is consistent through all interfaces, i.e., it behaves in the same manner for the same input.
- **Visibility of system status** : The application shows the user's mode - Novice, Typical and Expert as a label in the top right corner.
- **Efficiency of use** : The application Smart Gcc, efficiently produces the results and takes less time to show the correct results as per the GCC compiler.

2. Error tolerance

The System is error tolerant, on the 1st Home Screen, it checks whether GCC is installed in the system and then only runs. Secondly, on clicking the close button without saving an edited file, it prompts the user for confirmation whether he wants to exit without saving or save the file.

3. Design efficiency

   The design efficiently performs the tasks that the user instructs it to do, if the user wants to run his code, it runs the code from the editor, if the user wants to close the editor it closes the editor.

Prototype 2

SKETCHING

Sketching is one of the lower fidelity prototyping techniques. It is a paper based prototyping technique which would have lower developmental cost. Sketching was chosen because it is a rapid prototyping tool and communicates the idea very well. As the next designing step is evaluating multiple designs, it will help as a proof of concept.

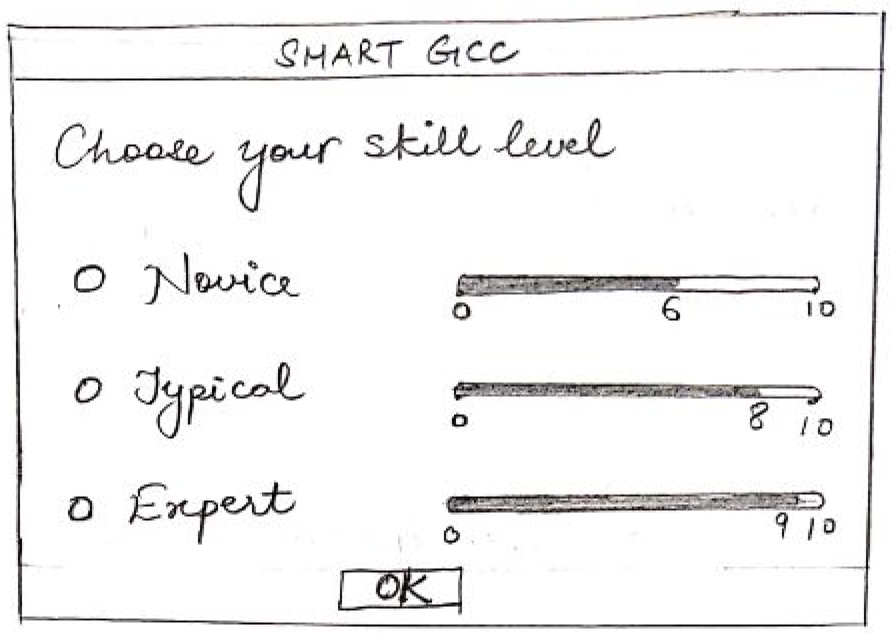


Fig. Choose user type

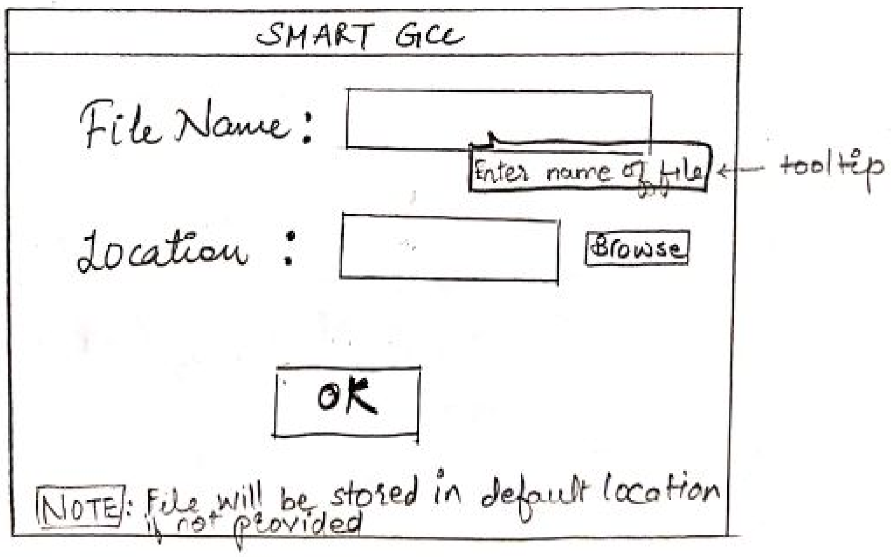
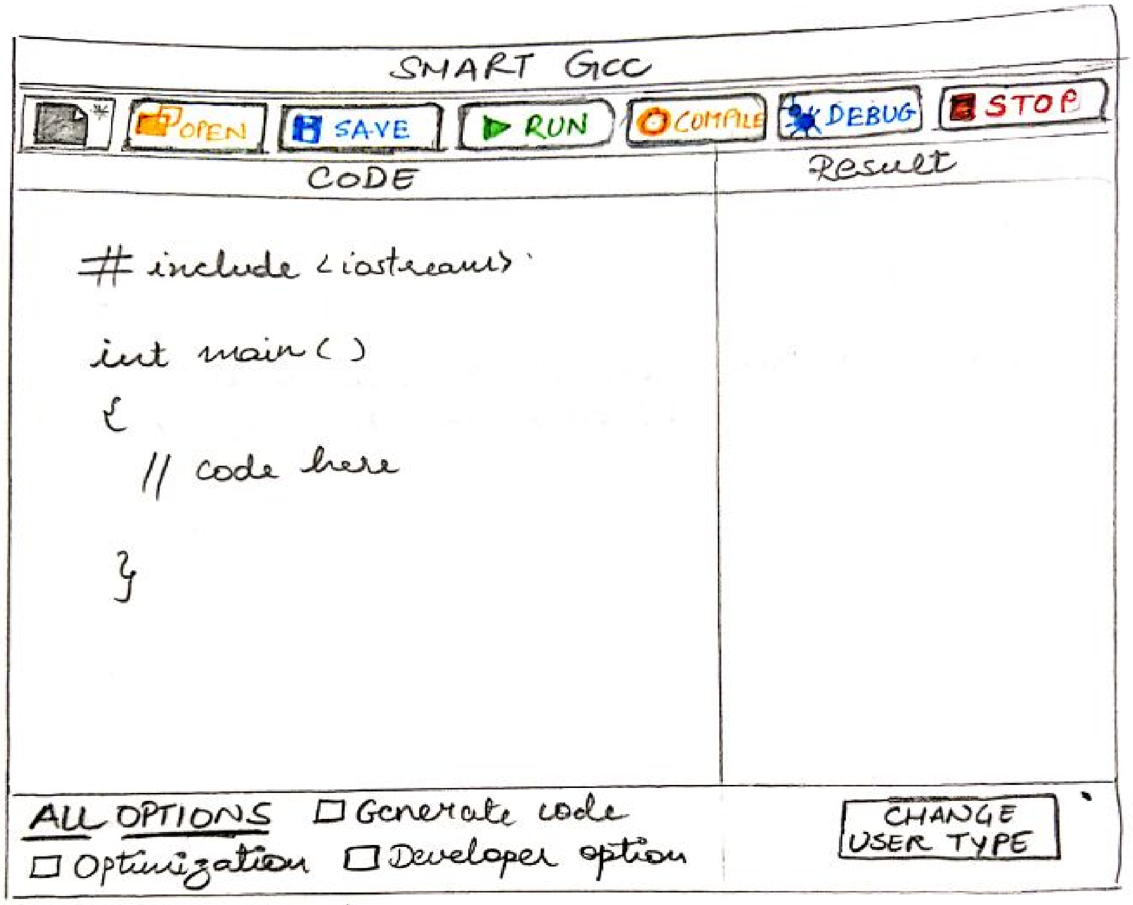


Fig. Create or Load a file



```
#include <iostream>
int main( )
{
  // code here

}
```

Fig. Main Module (Home screen with all functions)

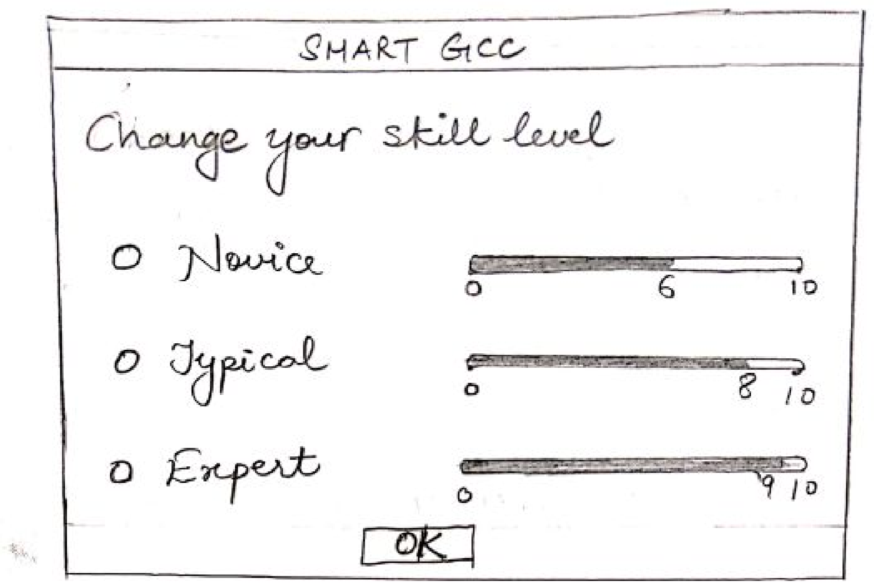


Fig. Change User Type

1. Design usability

- **Usability goals**
  - **Learnability**: This SmartGCC design would be easy to learn as it is designed with various generic symbols that are known to the user. Also, notes and help, through tooltips, are provided to guide the user.
  - **Effectiveness**: The system is designed in the way, the intended tasks of the user will be performed completely or would generate errors and warning messages
  - **Efficiency**: The design aims to provide a definitive output to the user in definite time.
  - **Memorability**: This goal is achieved by providing the option of saving the work done by the user.

- **Usability principles**
  - **Error Prevention**: The user has to confirm before applying changes to this user type or closing the application without saving.
  - **Consistency**: The application is designed to appear the same to all types of users, accept the options that are available for a particular user.
  - **Visibility**: The application will, at all times, show the current skill level of the user and the default options like run, compile, save or create new file.
  - **Efficiency in use**: The application is designed to give an output in finite time.

2. Error tolerance

The error tolerance is achieved in multiple ways in this design. The user will be prompted to save before closing the application. The application will prompt the user to confirm changes the user chose to make, before it is applied to his profile. Moreover, if the user tries to run the program that contains error after compilation, it would not allow the user to take that action as the run option would be disabled.

3. Design efficiency

   The design efficiency is achieved by providing a descriptive output in a time bound, in both cases, success or failure of executing the user's instruction.

Prototype 3

SCENARIO AND STORYBOARD

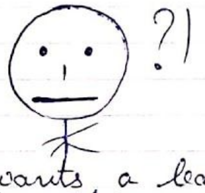
Nick is working professionally in his early thirties and is very hardworking. He is the technical lead of his team and the current need of the project requires him and his team to work on C/C++ to add some modules on their Java project. Nick worked on C/C++ in his college projects and now he needs a little guidance on how to run and compile the C/C++ code. He also needs to teach the same to his teammates. His team includes some members who haven't worked with C/C++ ever, some who used to work on C/C++ and some experts also. So Nick goes online to see if there is a tool for different skill levels, that can help his entire team to learn, catch up and brush the skill level according to the expertise of individual team members.

He searched for various tools for C/C++ and found SmartGCC. He downloads the tool and lands on the Welcome Page that explains him about the usage of SmartGCC and has an arrow for signing up and login option to change the user according to the expertise level of the user. Nick clicks the arrow and sees a signup page with a very beautiful and understanding sign up page. He signs himself as an intermediate level programmer for C/C++. Successful completion of the sign up process takes him to the home page, where Nick sees all the features and options as per his expertise level. He can also see an option to add or remove the features for his convenience. He sees a question mark option for "WHAT", that is a video recording on how an intermediate C/C++ programmer can use SmartGCC for his benefit. It explains all the options that the user can use for efficient learning and programming.

In the top header Nick sees the Developer option that allows him to switch to expert level and use all the options that are not very necessary for a beginner and an intermediate programmer. Nick explores all the options and realises this tool is appropriate for all the team members and can be very useful for the current need of the project. He then advises all his team members to download SmartGCC and start learning and using it according to their respective expertise levels.

Nick at his office ...

1.

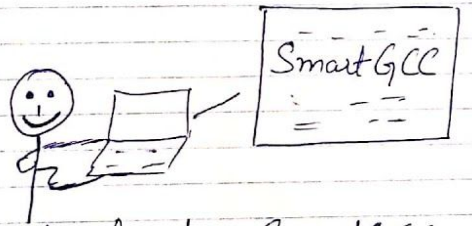


Nick wants a learning tool for C/C++ so that he and his team can learn for the current need of the project.

2.

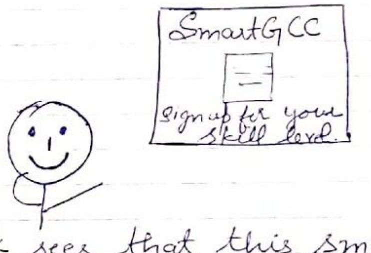
Nick opens his laptop and searches for a promising tool for learning.

3.



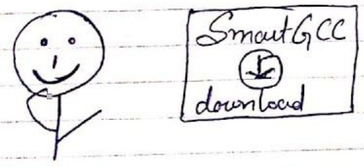
Nick finds SmartGCC online!.

4.



Nick sees that this smart GCC allows users to choose features as per their skill level.

5.



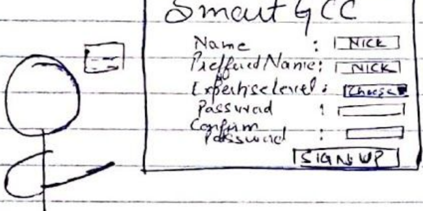
Nick downloads SmartGCC to explore all features and see its performance.

6.

Nick reads the welcome page instructions and gets ready to sign up

SmartGCC
Welcome

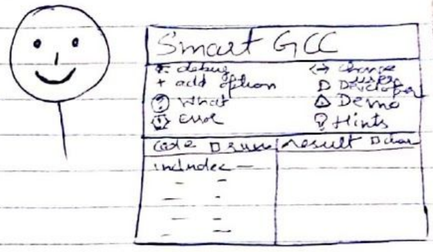
7.


Smart GCC

Name : NICK
Prefferd Name : NICK
Expertise level : Choose
Password :
Confirm Password :
SIGN UP

Nick fills in the basic information and signs up with skill level as intermediate.

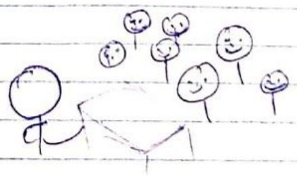
8.



Nick goes to homepage of SmartGCC and sees all the features for intermediate level programmer. He also sees that it gives you an option to add features that are not a part as of given skill level.

9. Nick is really satisfied with smartGCC and decides to use SmartGCC for his current project

10. Nick explains about SmartGCC to his team members, so that they can learn, refresh or brush up their skills



Design usability

- Usability Goals
    - **Effective**: It is how the application performs actions to use, efficiency: the way it performs tasks. For example it remembers the skill level of the user and does not ask it every time the user logs in.
    - **Safety**: Don't put delete with save or exit, showing dialog boxes,
    - **Utility**: The extent to which users can do things in the software. For example, accounting software has a lot of options, paint restricts users to use tools rather than drawing with hands.
    - **Learnability**: Users can learn from tutorials, reducing the time that users may spend learning.
    - **Memorability**: Remembering how to use the product. Categorizing options, making interface minimalistic which is more memorable.
- Usability principles
    - **Error Prevention:** Red tags tells the user when the code is syntactically incorrect, which prevents the user from committing errors.
    - **Consistency**: The app is consistent throughout as the interface is familiar to the user as it is similar to most of the apps.
    - **Visibility of System status**: showing the skill level and options explored out of the skill level.
    - **Efficiency of use:** The app takes less time to give results.

Error tolerance

The design is error tolerant as it doesn't allow the user to make mistakes while entering wrong information. For example when the user tries to enter invalid skill level, the application asks relevant questions to confirm the same.

Similarly, when the user tried to quit the app, it asked the user if he/she really wanted to quit the application and to save the code if the user really wanted to exit the application. This prevents accidental quitting of the application and loss of data.

Design efficiency

The design does what it needs to do efficiently and shows the user the result as soon as it is done. For example, when the user compiles the code or changes the skill level, the interface shows the result of the compiled code or efficiently shifts the skill level with new options on the screen.

Work division in Team

Module	**Team member**
Conceptual Design 1	Subhannita Sarcar
Prototype 1	Subhodip Ray
Conceptual Design 2	Nancy Goyal
Prototype 2	Roohani Naik
Conceptual Design 3	Hesamedin Dadgar
Prototype 3	Bhavpreet Kaur