SIT102 Introduction to Programming

Pass Task 1.1: Hello World

Overview

As your first step, create the classic "Hello World" program. This will help ensure that you have all of the software installed correctly, and are ready to move on with creating other programs.

There is a walk through video on how to do this in the week 1 material. These videos will show you what you need to do in order to get things working.

As this is your first program, the guidance in this task is very detailed. It includes information that you can use now to understand how to work with the terminal, and you can refer back to this if needed in the future. For this task you can follow along with the videos, as these show you what you need to do. It would be good to read over these details, as they help extend your understanding and give you tools to succeed with later tasks. While future tasks will have relevant details, most future tasks wont have this level of detail.

Submission Details

For this task you need to make the classic Hello World program, and submit its code along with a screen shot showing that you got it working.

Submit the following files to OnTrack:

- Hello World source code (the *Program.cpp* file)
- A screen shot of your program running
- Answers to the questions available in the task resources

Check the following things before submitting:

- Try to make your code match the supplied code in format and layout
- You can demonstrate how to compile and run from the terminal

Instructions

The first task includes the steps needed for you to install the tools you will need in this unit. You will then use these tools to create the classic "Hello World" program.

Setting up a folder

1. Install the tools you need to get started.

   Check the install instructions for your operating system:

   - Install build tools, VS Code for Linux
   - Install xcode tools, VS Code for MacOS
   - Install MSYS2, VS Code for Windows

   > **Note:**
   >
   > You can skip these steps on Deakin lab computers as the software is installed.
   >
   > Remember on Windows that you need to use `mingw64.exe` provided by MSYS2 to access your terminal.

2. Follow along with the video in the weekly content. This will show you how to use the compiler to build and run the hello world program.

   If you would like an additional recap, here is a previous 1.1P Hello World video. This will walk you through the steps that are listed below.

3. If you don't already have one, make a directory (i.e., a *folder*) to store your code (e.g., Documents/Code). On a Deakin computer you may wish to use a directory on your student drive or a USB storage device.

   - Navigate to your Documents directory in *Finder* or *File Explorer*
   - Right click in the Documents directory and select **New Folder**, name it **Code**

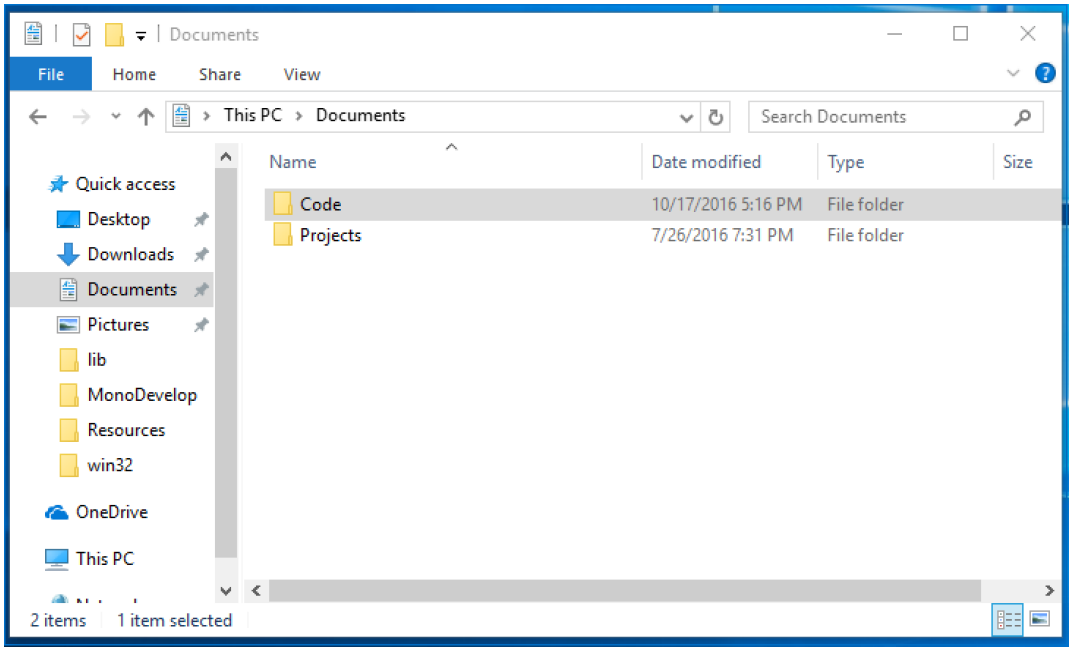

Figure: Windows explorer showing code folder in Documents

Feel free to place this somewhere else on your computer if you want, but please avoid using spaces in the names of any of the folders. Spaces in names will make it hard to interact with from the Terminal as the terminal uses spaces to separate different parts of its commands.

On the lab machines at Deakin you should store this in your user folder located in **/c/SIT/Scratch/**.

4. Open your **Terminal** (Using the Mingw app for Windows)

   Now we have a folder in place, we need to switch to the Terminal to proceed. The Terminal is a program that gives you a command line interface to the computer. You type commands in, press enter, and the Terminal's shell interprets the text you type and performs the actions you requested. Using the terminal and writing programs have many similarities, which makes the Terminal very useful for software developers. As a result there are many advanced programming tools that you can use from the Terminal. For the moment we will stick to the basics, but once you get started there is so much more you can do with this tool.

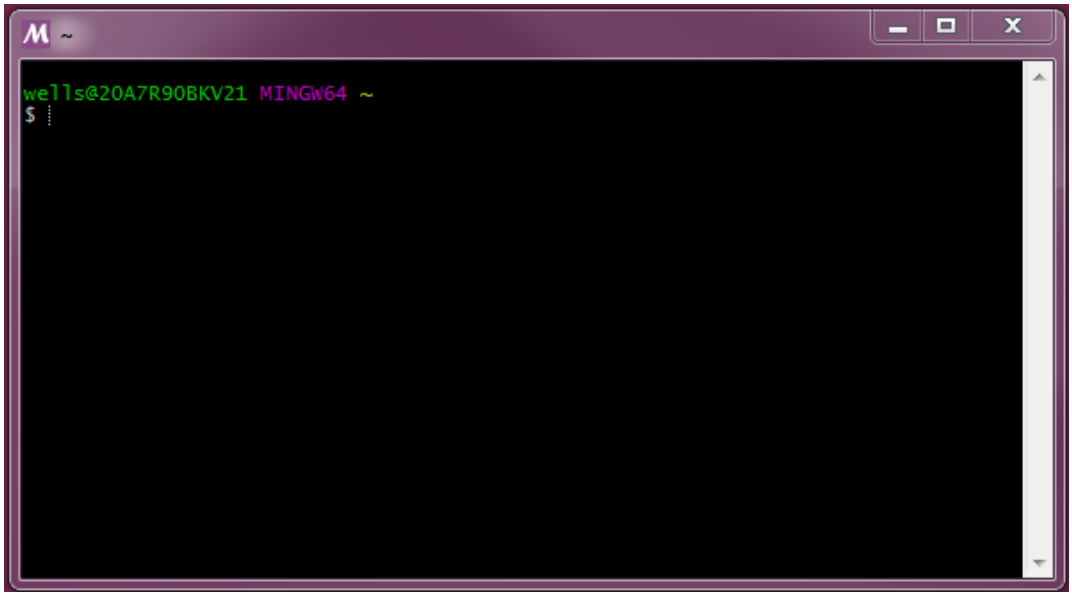

Figure: Example terminal window

5. Navigate to your new folder using the **cd** command.

   Within the terminal, your actions are centred upon a **working directory**. This then gives you easy and ready access to the files and other folders/directories that are located within the *working directory*. So, typically the first task you need to do is change the working directory. In this case we need to change into the *Code* directory you created in your *Documents* folder.

   For example, on Windows this would be something like this, to cd into a folder on my C: drive in `Users/andrew/Documents`.

   ```
   cd /c/Users/andrew/Documents/Code
   ```

   For Mac and Linux, its a little easier as it includes a `~` shortcut to get to your home directory:

   ```
   cd ~/Documents/Code
   ```

   Your terminal is now using this as your working directory. You can check this using the `pwd` command which asks for the **present working directory**.

   ```
   pwd
   ```

   Once you are in the right directory we can create a folder for the project and then initialise this to give us a C++ SplashKit project.

   > On some systems file and folder names are case sensitive, so make sure you type this in carefully: `Code` and `code` are two different names!

6. Create a folder and initialise your Hello World program:

   You can create a directory from the command line using the `mkdir` command.

   To create a *HelloWorld* directory/folder within the *Code* folder you can just run `mkdir HelloWorld` as the terminal is currently in the `Code` folder.

   ```
   mkdir HelloWorld
   ```

7. Now change into that directory. You can use a shortcut by typing `cd He` then hit the *tab* key to auto-complete the folder name. Making use of this awesome feature will help save typing and make you more productive. The command will be `cd HelloWorld`. Hit enter to run the command.

   Like with the `mkdir` command above, this is relative to the current working directory. So this will move into the *HelloWorld* folder in the *Code* folder, etc. File and folder names are relative to the current directory if they do not start with a tilde (~) or a forward slash (/). In these contexts, `~` represents your home directory and `/` represents the root (start) of the file system.

   Both `.` and `..` are also special identifiers, `.` represents the current directory and `..` represents the parent of the current directory. So if you ran `cd ..` when you are in the *Code* directory, it would take you back to the *Documents* directory.

   Run the following command to move into the HelloWorld folder, if you haven't done so already.

   ```
   cd HelloWorld
   ```

8. To setup the project run the following commands in the terminal:

   ```
   skm new c++
   ```

   This gets SplashKit to initialise the project, and sets it up ready for you to work with it from Visual Studio Code.

   Run the following command to see the list of files that these commands created:

   ```
   ls -lha
   ```

   The `-lha` tells the `ls` (list) command to print the names in a list (`-l`) in human readable format (`-h`) and to show all files (`-a`). You should see an `include` folder, a `program.cpp` file as well as the project files.

Writing the Code

1. Open **Visual Studio Code** and within it open your `HelloWorld` folder. The `File` menu should allow you to `Open Folder...` on Windows, or just `Open` on Mac and Linux. You can then use a standard dialog to navigate to and select your `HelloWorld` folder.

   You should see something like the following when this works. You can open the *program.cpp* file by double clicking it in the list on the left.

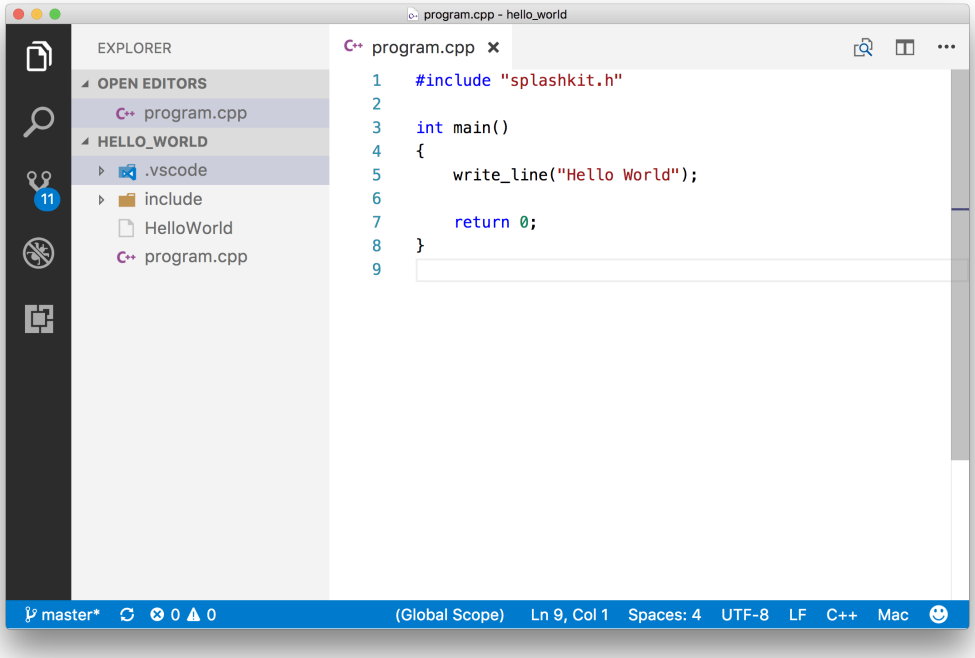

Figure: VS Code showing initial project details

> You should just be able to run `code .` from the Terminal to open VS Code with the current folder (which is `.` in the terminal).

2. Update the code to include the statements that will write hello world to the terminal. The code is shown below (and shown in the previous screenshot).

```
#include "splashkit.h"

int main()
{
    write_line("Hello World");

    return 0;
}
```

C++ is case sensitive, so take care when typing this in. For example, if you call `Main` instead of `main` it wont work! This can be a bit of a pain when you get stared, but with care you will be fine.

3. Save the *program.cpp* file, by selecting Save from the File menu or using the shortcut (ctrl+s or cmd+s for macOS).

That's it. If you have typed this in correctly you have the code for your first program!

Notice the color highlighting in the editor, this is known as *syntax highlighting*. Code editors use knowledge of the rules of the programming language (their syntax) to color different words based on their meaning in the language. These highlights help you visualise the structure of your program, and make sure that you don't have small typos in key words.

> **Beware**
>
> Watch out! Copying code from a PDF and pasting it into your own program may result in invalid characters that wont be processed when you upload to OnTrack. Please type these in yourself.
>
> Typing the code in yourself will help you learn. You want to know how to do this yourself.

Compiling the Program

Now that you have the code, you need to get it into a format that the computer can use. There are many different tools that can be used to achieve this, but they can be generally categorised as either *interpreters* or *compilers*. An **interpreter** will read the program's code and then give the computer the instructions it needs to carry out the requested action as it goes. A **compiler** will read all of the program's code and then save the instructions for the computer into a separate **executable** file. When comparing compilers and interpreters, each has their own advantages and disadvantages. In general interpreters offer greater flexibility and can simplify the programming process, but are slower as they need to interpret the code as the program runs. In contrast, compilers are able to generate efficient code but are generally less dynamic.

The C++ programming language uses a compiler. The C++ compiler reads your code and produces an executable that you can run.

1. Switch back to your Terminal, or open it again and `cd` back into your project's folder.
2. Check you are in the right current directory. You can use the following commands:
    - List the files in this directory using the **ls** command. This will print out the list of files and folders in the directory.

      ```
      ls -lha
      ```

    - Print the working directory using the **pwd** command

      ```
      pwd
      ```

3. Build your program code using the `clang++` (or `g++`) command line tool:

   ```
   skm clang++ program.cpp
   ```

   This compiles your program.cpp code and generates a program called `a.out` by default (`a.exe` on Windows). You can change the name of the program it produces by passing an additional flag to the compiler. By adding `-o HelloWorld` you are telling the compiler to **o**utput a program called `HelloWorld`:

   ```
   skm clang++ program.cpp -o HelloWorld
   ```

   > If this doesn't work then check your installation steps, and ask for help on the discussion board or drop into the **HelpHub** (also know as SIT Learning Support Hub) check the Unit Information page in the unit site for details. You need to get things working as quickly as possible, so get on to this ASAP.

4. Run the program by using its name so either a.out (a.exe on Windows) or HelloWorld:

```
./a.out
./HelloWorld
```

Your program should run, and you will see the message *Hello World* written to the Terminal.

> If this all works you should feel confident that things are setup correctly. If you have any issues use the discussion board to get help, as it is important to get these sorted quickly.

Congratulations! You have written your first program. We will look at what all this code means soon, but for the moment let's take a look at how you can submit your work for feedback.

Your Task

To finish off this task you now need to download the **Task Resources** from OnTrack. You will find the button to download these in the Task Details page, as shown below.

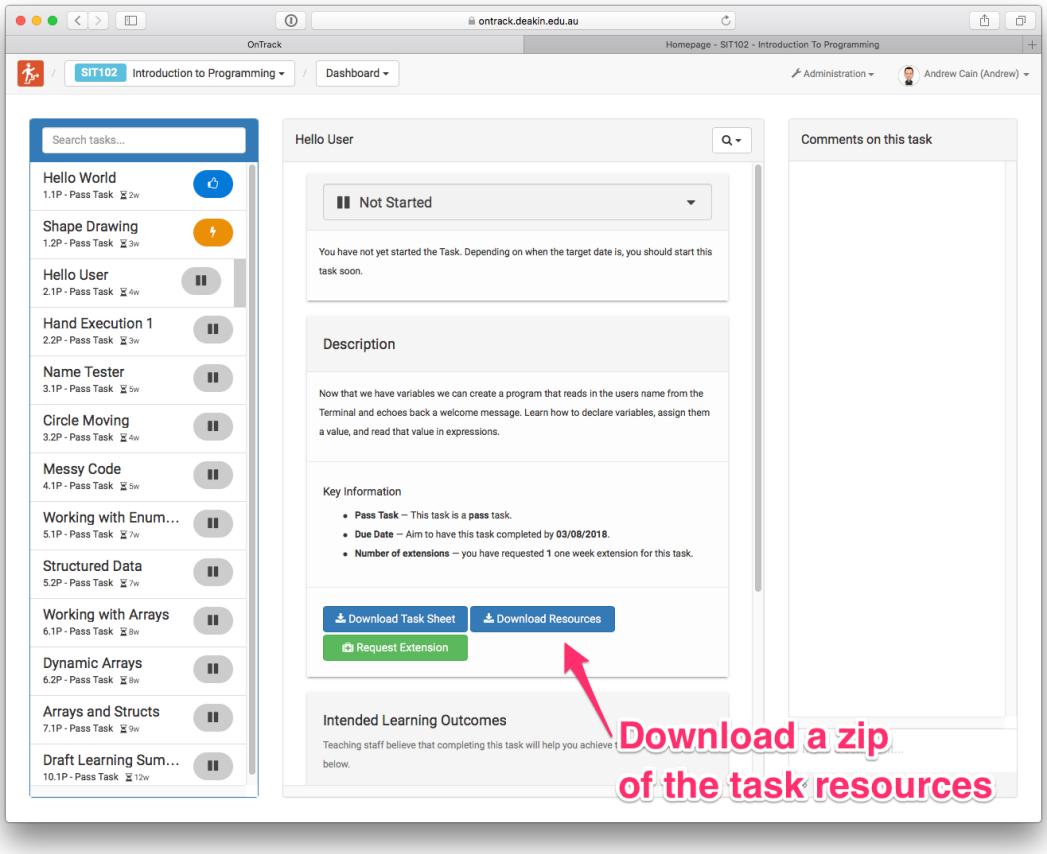

Figure: Download task resources

1. Unzip the file, which in this case will contain a word document template for your answers.

2. Open the word document, and answer the questions within. You want to use these as a chance to demonstrate you have fully understood the actions you performed in the task. So in this case that relates to the use of the tools needed to compile and run a program.

3. When finished, save your work and keep a copy of this. You may need to fix and resubmit your answers if they are not adequate.

4. Next export a copy of the document to PDF. You can use **Save As** to save to PDF. You will need the PDF document to submit to OnTrack.

Submitting your work

Check that you are happy with your code and answers, and that you are ready to submit it to OnTrack for feedback.

1. Login, and go back into Tasks 1.1P.
2. Change the status of the task from **Working on It** to **Ready for Feedback**
3. Check what files you will need to upload. In this case it includes the program's code, answers PDF, as well as a screen shot of the running program.
4. Once ready, run your program and use the appropriate tools to get a screenshot.
    - In Windows you can use the Snipping Tool
    - In macOS you can use cmd+shift+4 to select an area for a screenshot.
    - In Ubuntu Linux, you can use Dash
5. Switch back to OnTrack and upload the code, answers, and screen shot.
6. On the next screen you will need to align what you have done to the unit learning outcomes. For this you could say it is slightly related to programming in general. Add a short comment to describe this, then move on to the next step.

    The purpose of this step is to reflect on what you have leart, and how this will help you demonstrate the unit learning outcomes. You can commenting on what you think you got out of completing this task, and anticipating its overall relevance in your final portfolio. So use a rating of 4 and 5 for the work you think will be your best work in your portfolio.
7. In the last step you can add a comment to your tutor, let them know if there are any things you want them to focus on.
8. Finally, upload...

The files will be uploaded to the server, which will convert them into a single PDF file that can be included in your portfolio. This file will be shown to your tutor who will review it and get back to you shortly with some feedback.

Well done. You have finished this first task!

Task Discussion

A critical part of this unit will be the discussions you have with your tutor on your progress with the tasks and the development of your understanding of the associated concepts and skills. Once your tutor has checked your task they will indicate you need to *Discuss* this with them. This means your work looks good, but we want to discuss this with you to see how you are progressing with the ideas behind the code. This is your opportunity to highlight what you have learnt, or to get feedback on aspects you would like some support with.

Note: these discussions must occur in order for you to be eligible to pass the unit. Please ensure that you are regularly engaging with your tutor to get your tasks signed off. Campus students must do these

discussions in person in their scheduled practical classes.

For this task you need to discuss at least the following with your tutor:

- A little about your background and interests.
- How you went with the task.
- Editor setup - make sure to check you have all the things installed you need. Ask if you need help.
- How does your code become something that can be run on the computer?

Note: While your tutor will help you monitor your progress, in many cases they may need to refer you to other resources such as the help hub, discussion board, or online seminar for further assistance. Submissions in OnTrack are generally for work that you consider to be complete (or very close to it). If you need help with a task, engage with the support services to help you make progress.