SIT102 Introduction to Programming

Pass Task 1.1: Hello SIT102 & Hello World

Overview

This task firstly includes a section called “**Hello SIT102**”. It provides guided questions for you to explore the unit site materials and gain understanding of the unit arrangement. It enables you to be familiar with the unit and your learning opportunity, and leads you to success in this unit.

Then, create a classic “**Hello World**” program as an installation tester in Part B of this task is your first step in software creation to ensure your device is programming/coding-ready. As this is **your first task in the unit**, **the guidance in this tasksheet is very detailed** covering *almost* all steps of activities in your week 1 class. It includes information that you can use to understand how to work with the Terminal. It would be good to read over these details as they help extend your understanding of building and executing programs, and give you tools to succeed with later tasks, as listed in Part B – Your Task section which aim to let you familiar with OnTrack submission and completeness processes. **Future tasks will have relevant instructions and requirements, but they won’t have this level of details**.

Submission Details and Your Intended Learning Achievements

For completing this task, you need to showcase the following achievements in your submission:

Section of this task	*Your Intended Learning Achievements*
• **Hello SIT102:**	your understanding of the unit and assessment information
• **Hello World:**	your proper implementation of the classic Hello World program, your executable, and your understanding of the software creation processes

Submit the following files to OnTrack as your task deliverables showcasing the above achievements:

- PDF of your Answers to the questions available in the task resource (Download it from your OnTrack panel - Resources.zip)
- A screenshot of your program running in the Terminal
- Hello World source code (the program.cpp file)

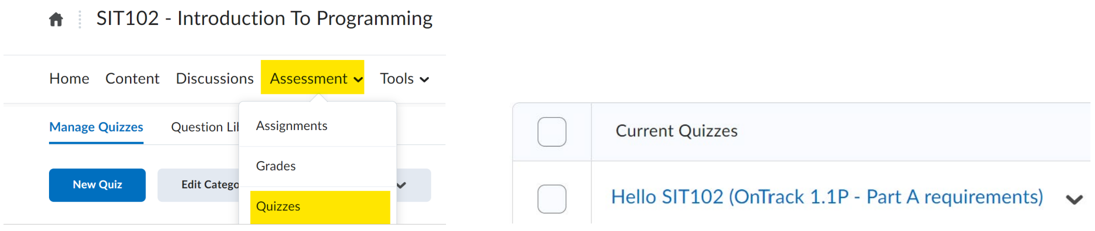
> **Note:**
> OnTrack will ask you to upload the required file **in order**, so please pay attention to the OnTrack upload instructions during submission.

Submission Due

The due for each task has been stated via its OnTrack task information dashboard.

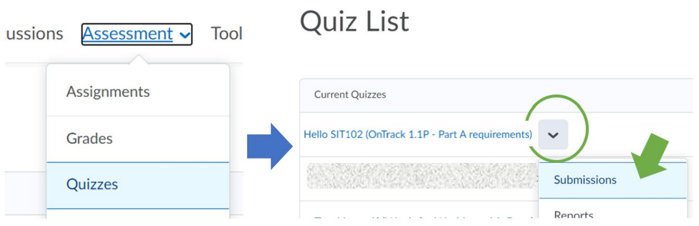
Instructions for Part A: Hello SIT102 section

1. Make sure you would have gone through **SIT102 Unit Site** >> **Announcement**: [Welcome to SIT102 Introduction to Programming]() and its indicated contents, including the Unit Guide, Week 0 - Introduction (and Installation) for the essential details of unit information and assessment criteria.
2. Navigate to **SIT102 Unit Site** >> Assessment >> Quizzes >> Hello SIT102 (OnTrack 1.1P - Part A requirements) (See below figures) *where*
   - you are getting the questions of Hello SIT102,
   - you are submitting your answers to it for system's auto-marking in the unit site quiz tool,
   - you are getting the screenshot of **a full mark** to showcase your understanding of the unit and assessment information.
     - *Note: this Hello SIT102 (OnTrack 1.1P - Part A requirements) quiz has set up unlimited attempt for you to obtain a full mark. If you have gone through the unit information, you would be able to complete this part (all are Matching and MC questions) with a full mark in 15-20 minutes.*

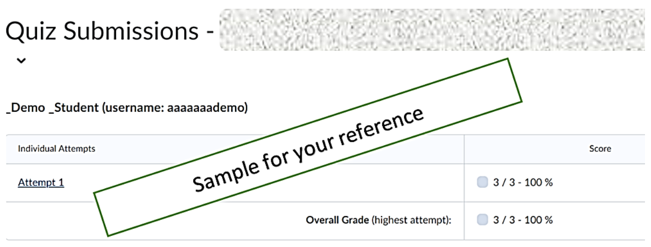


Figures: Parts of layout of **SIT102 Unit Site** >> Assessment >> Quizzes

3. Screenshot your **full mark** result. The screenshot should also include **<u>your name showing on it</u>**
   - The score for auto-mark sections will be available in unit site >> Assessment >> Quizzes in 1 – 2 minutes after your submission. Click the downward arrow next to the corresponding quiz item and select Submissions.



   - Then, you can reach this in the following page with your quiz's auto-marked score



Quiz Submissions -

_Demo _Student (username: aaaaaaademo)

Individual Attempts	Score
Attempt 1	3 / 3 - 100 %
Overall Grade (highest attempt):	3 / 3 - 100 %

4. Paste your screenshot to the given answer sheet WORD doc
   - Download answer sheet WORD doc from your OnTrack panel - Resources.zip).
   - You may also want to jump to the section "Your Task" in this tasksheet to know more about OnTrack panel and resources.zip.

Instructions for Part B: Hello World (Start Programming!)

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.

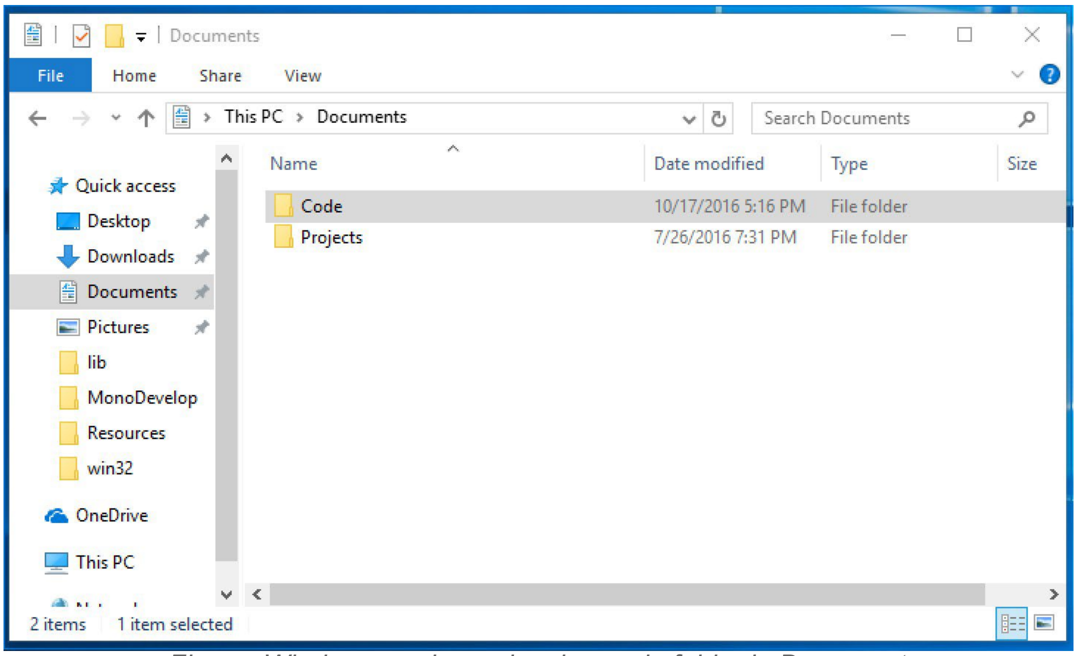
Setting up a folder

1. Install the tools you need to get started and 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 (if applicable) as the software is installed. Remember on Windows that you have to use MSYS2’s `minGW.exe` to access your terminal.

2. **Apart from your Week 1 Class activities**, **for week 1**, here are walk-through videos in the unit site on how to start creating, compiling, and executing a Hello World classic program as the installation tester.
    - (You may have gone through this during Part A of this task) Unit Site > Unit Information > Week 0 - Introduction and Installation
    - (This is a video recap of the following texts in this tasksheet) Unit Site > Online Class > Weekly Pre-Class Learning Resources > Week 1 - Programs and Procedures > Building Programs > Video 1.6 - Installation Tester - Hello World

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**



Name	Date modified	Type	Size
Code	10/17/2016 5:16 PM	File folder	
Projects	7/26/2016 7:31 PM	File folder	

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.

4. Open your **Terminal**

   For windows users, *use your installed* ***MSYS2 minGW x-bit*** *app (in the above step 1), 64 or 32 bits depends on your OS version.*

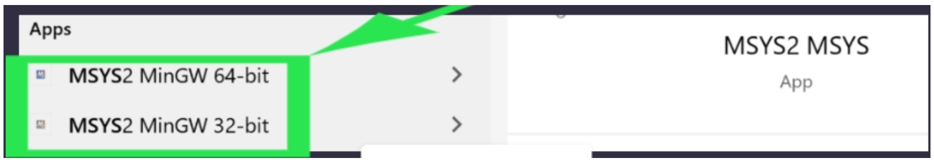


Figure: Illustration of a search for MSYS2 minGW x-bit app via Start Menu

   For Mac users*, use your installed* ***Terminal app*** *(obtained by the above step 1 Xcode installation), drilling down into the* ***Utilities*** *folder within* ***Applications)***

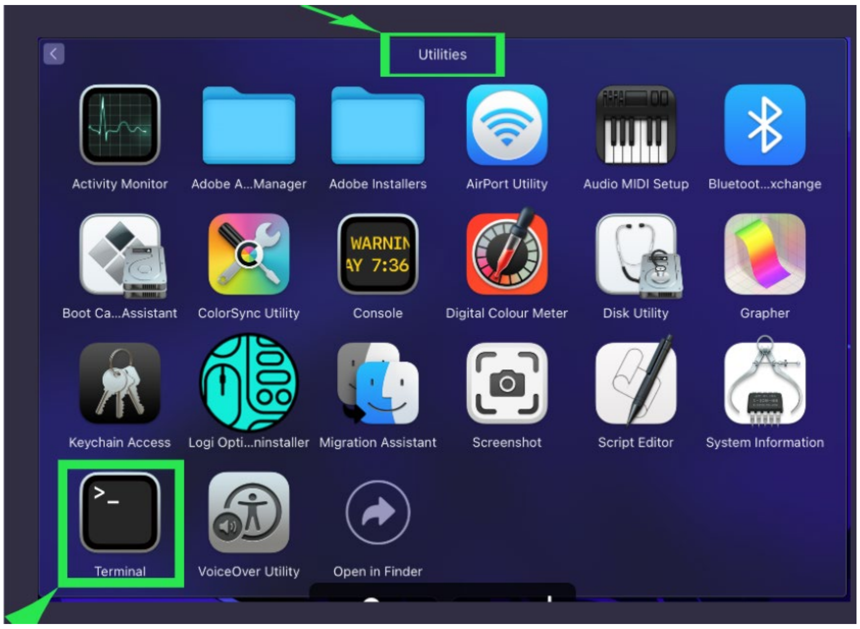
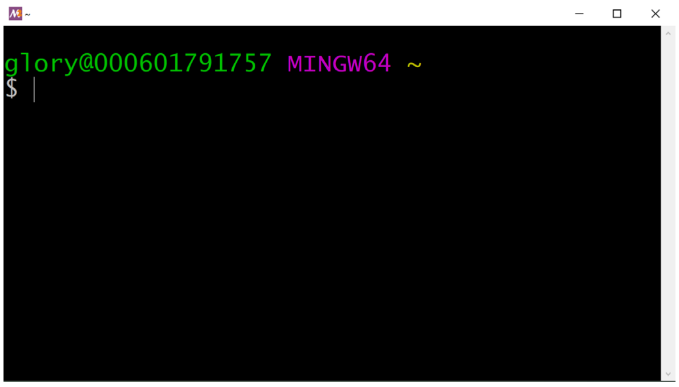


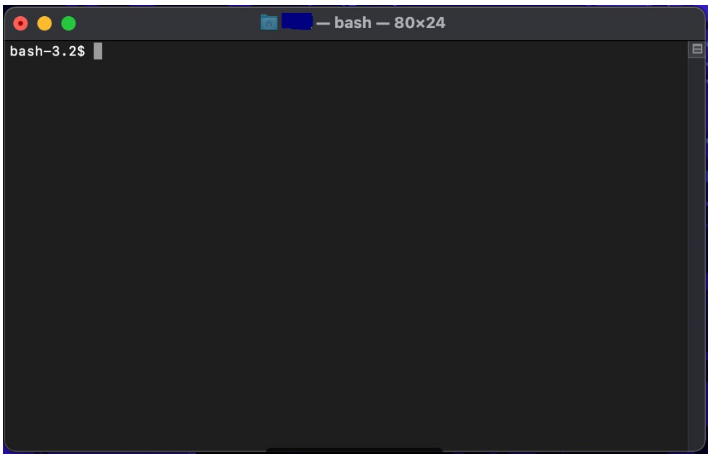
Figure: Illustration of a Utilities folder within Applications in MacOS

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.



```
glory@000601791757 MINGW64 ~
$ |
```

Figure: Example MSYS2 minGW 64-bit terminal window for windows users



```
[redacted] — bash — 80×24
bash-3.2$
```

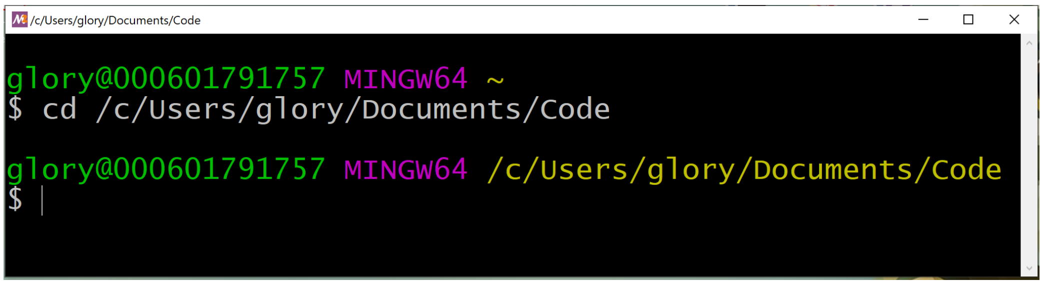
Figure: Example Xcode Terminal window for Mac users

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/glory/Documents` .

   ```
   cd /c/Users/glory/Documents/Code
   ```



```
/c/Users/glory/Documents/Code

glory@000601791757 MINGW64 ~
$ cd /c/Users/glory/Documents/Code

glory@000601791757 MINGW64 /c/Users/glory/Documents/Code
$ 
```

Figure: Example of inputting a cd command in the Terminal

   Give you one more example: If Andrew wants to **cd** into his folder based on the folder path, `Users/andrew/Documents` , he would make use of the following **cd** command:

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

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

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

   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!

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.

6. Create a directory (folder) and initialise your Hello World project folder using **mkdir** command.

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

   ```
   mkdir HelloWorld
   ```

7. Now, navigate to 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 command line in the terminal.

   ```
   skm new c++
   ```

   This gets SplashKit to initialise your project with the necessary dependencies. You should see an **include** folder, a **program.cpp** and some associated project files **under your project folder**.

   You could also run the following command to list files that the above **skm** command line has created:

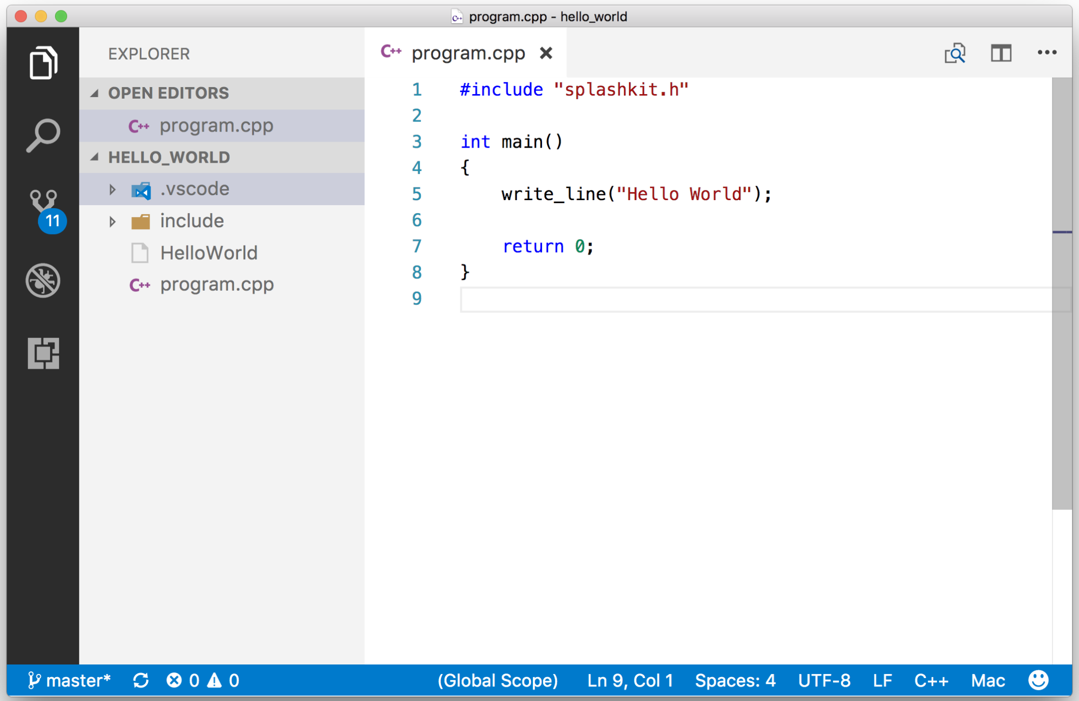
   ```
   ls -lha
   ```

   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**).)

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.



```cpp
#include "splashkit.h"

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

    return 0;
}
```

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 won't 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 won't be processed when you upload to OnTrack. Please type these by 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** (since week 0), join a **Class** session (since week 1), or drop into a **HelpHub** session (since week 1) 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 (the name you have assigned to your program in the compilation process):

```
./a.out
```

```
./HelloWorld
```

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

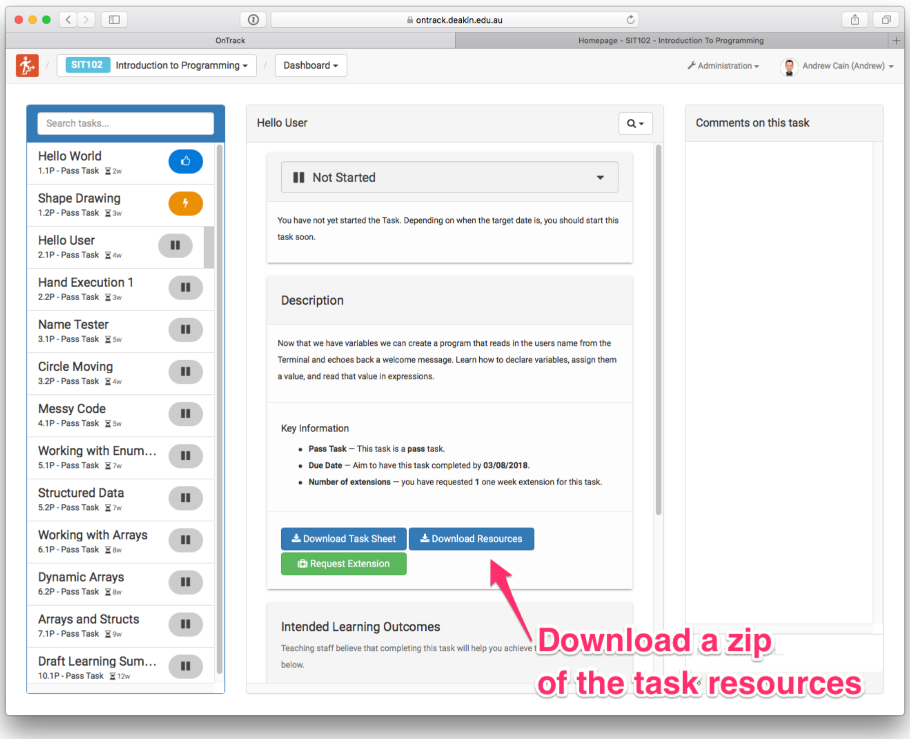
> If this all works you should feel confident that things are setup correctly.
>
> If youhave any issues use the discussion board to get help, as it is important to get these sorted quickly.

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

Your Task

Besides the above program, submit the other required work for this 1.1P (as stated in page 1 of this tasksheet) to OnTrack

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.


ontrack.deakin.edu.au
OnTrack
Homepage - SIT102 - Introduction To Programming
SIT102 Introduction to Programming
Dashboard
Administration
Andrew Cain (Andrew)
Search tasks...
Hello World 1.1P - Pass Task
Shape Drawing 1.2P - Pass Task
Hello User 2.1P - Pass Task
Hand Execution 1 2.2P - Pass Task
Name Tester 3.1P - Pass Task
Circle Moving 3.2P - Pass Task
Messy Code 4.1P - Pass Task
Working with Enum... 5.1P - Pass Task
Structured Data 5.2P - Pass Task
Working with Arrays 6.1P - Pass Task
Dynamic Arrays 6.2P - Pass Task
Arrays and Structs 7.1P - Pass Task
Draft Learning Sum... 10.1P - Pass Task
Hello User
Not Started
Description
Key Information
Download Task Sheet
Download Resources
Request Extension
Intended Learning Outcomes
Comments on this task
Download a zip of the task resources

You have not yet started the Task. Depending on when the target date is, you should start this task soon.

Now that we have variables we can create a program that reads in the users name from the Terminal and echoes back a welcome message. Learn how to declare variables, assign them a value, and read that value in expressions.

- **Pass Task** — This task is a **pass** task.
- **Due Date** — Aim to have this task completed by **03/08/2018**.
- **Number of extensions** — you have requested **1** one week extension for this task.

Teaching staff believe that completing this task will help you achieve t 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 OnTrack, and go back into Task 1.1P.

2. Change the status of the task from ***Not yet Started / 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 Windows logo key + Shift + S to open Snipping Tool.
   - In macOS you can use cmd+shift+4 to select an area for a screenshot.
   - In Ubuntu Linux, you can use the Unity Dash button and type the word screenshot in the search bar, or tools like Flameshot

5. Switch back to OnTrack and upload the required code, answers, and screenshot.

6. On the next screen you will need to align what you have done in the task to achieve **unit learning outcome(s)**. This process helps you to reflect each of your programming work/task of the unit. Add a *short reflective comment (around 50 words)* to describe this, then move on to the next step.

   The purpose of this step is to reflect on **what** you have learnt, and **how** this will help you demonstrate the unit learning outcomes. You can **reflectively comment** on what you got out of finishing/completing this task, and anticipating its overall relevance in your final portfolio based on the unit learning outcomes. So use a rating of 4 or 5 for the work you think will be your best work evidencing in your portfolio to demonstrate your achievements of the corresponding learning outcome(s).

   In the last step you can add a comment to your OnTrack reviewer (tutor), let them know if there are any things youwant them to focus on. Remember **professional communications** could help you exchange ideas of the core knowledge covered in the unit.

7. 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 reviewer who will review it and get back to you shortly (in 5 business days) with some feedback.

Well done! **You have submitted this first task! The last step is to complete your task…**

Task Completeness

An essential part of this unit will be the interactions between **you** and your **OnTrack reviewer (tutor)** on your progress with the tasks and the development of your understanding of the associated concepts and skills. Once your reviewer has checked your task they will indicate you need to ***Discuss*** *or* ***Demonstrate*** 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. Sometimes, you may also need to fix some minor issues in your task based on the feedback given by your reviewer, indicated by an OnTrack status, ***Resubmit***. This is your opportunity to highlight what you have learnt, or to get feedback on aspects you would like some support with. **When your resubmission and answers to the discussion/ demonstration are up to standard**, ***in the next review cycle***, **your reviewer will sign off** your task as ***Complete,*** *as a process of building up your portfolio contents.*

Note 1: These interactions/discussions/demonstrations must occur in order for you to be eligible to pass the unit. Please ensure that you are regularly engaging with your OnTrack reviewer to get your Pass tasks signed off **by the end of a milestone**. **See unit guide for tasks and milestones details.**

Note 2: While your reviewer will help you monitor your progress, in many cases they may need to refer you to other resources such as asking you to join a class/HelpHub session, read posts in discussion board, or revisit class resources for further assistance. **Submissions in OnTrack are generally for work that you consider to be complete (or very close to it) with all requirements have been fulfilled**. Otherwise, your reviewer will directly ask you to **Redo** the task with a resubmission. If you need help with a task, engage with the support services to help you make progress.

Note 3: If your task is given a **Resubmit** status, the task due date will be revised with a 5-day extension **UP TO the task due date** by OnTrack system giving you time to improve your task. Please check the updated due in the task Information panel.

Note 4: You are highly recommended to submit you work as early as possible as **back and forth communications** on your tasks will be needed **after your submission and before the due (Please be noted: OnTrack turn-around time is about 5 business days).**

In the last iteration approaching the due (if your task has not yet been signed off due to some of the task requirements have not yet been fulfilled), your reviewer will do the review and give feedback to your latest on-time submission in 5 business days following the due date. The on-time submission will be signed-off at that time if it is up to the standard. Otherwise you have to make sure your work is then well fixed and get the **task signed off by** the corresponding **milestone as one of the passing requirements for this unit.**