ez020691

Generated by Doxygen 1.8.17

Chapter 1

_FrontPage

Module Code : CS1PC20

Assignment Report Title: Portfolio

Student Number (ez020691)

Date: 1.11.21

Hours Spent : 19 Hours

Assignment Evaluation: Navigate around the CLI Use and understand the different git commands Write simple programs using C Language

Chapter 2

CS1PC20 Portfolio

Chapter 3

submission_answers

Q1, It is important to use code libraries as it can save time and can also improve the quality of the program. Having code librarys will help you when creating a large program as you can use the same library over and over again for different projects if they meet the needs which can reduce project development time and improve the relibality of the software. Using code libraries in a group project can also be helpful for example if one the team members need a specic code but there is already a library written which does the function they can just use the library instead wrtitiing the whole code again.

Q2, The purpose of #include is to tell the C preprocessor to include all the contents of the file that are specified in the input stream to the compiler and then continue with the rest of the original file. When it comes to .h files its primary roles is to propagate declarations to code files. This allows programmers to import whenever they need them.

Chapter 4

introductory-bash

Report Week1

1. $ mkdir -p $HOME/portfolio/week1 ; cd $HOME/portfolio/week1 Expectation – Will create a folder named portfolio and week1 inside it Results – Creates the folder named portfolio and week1 inside it
2. $ cd ∼ Expectation – Exits all the folders Results – Exits all the folders
3. $ rm -r portfolio Expectation – Removes the folder named portfolio and everything inside Results – Removes folder named portfolio and everything inside
4. $ mkdir -p $HOME/portfolio/week1 & cd $HOME/portfolio/week1 Expectation – Creates the folder named portfolio and week1 inside it Results - Creates the portfolio folder and goes into it but it tries to execute the command same time so it returns an error
5. $ cd ∼ Expectation - Exits all the folders Results - Exits all the folders
6. $ rm -r portfolio Expectation – Removes the folder named portfolio and everything inside Results – Removes the folder named portfolio and everything inside
7. $ mkdir -p $HOME/portfolio/week1 && cd $HOME/portfolio/week1 Expectation – Crates the folder named portfolio and week1 inside it and enters the folders Results - Creates the folder named portfolio and week1 and enters the folder
8. $ echo "Hello World" Expectation – Prints out Hello World Results Prints out Hello World
9. $ echo Hello, World Expectation – Prints out Hello, World but possible error can appear due to no "" Results – Prints out Hello, World
10. $ echo Hello, world; Foo bar Expectation – Prints out Hello, World; Foo bar Results – Prints out Hello, World and then returns an error saying Foo bar is not a command the ; means that the command stops there and it's a new command after.
11. $ echo Hello, world! Expectation – Prints hello, World! Results – Prints Hello, World!
12. $ echo "line one";echo "line two" Expectation – Prints out line one and breaks the command because of the ; and then creates new line to print out line two Results - Prints out line one and breaks the command and creates new line to print out line two
13. $ echo "Hello, world > readme" Expectation – Prints out "Hello, world >readme" Results – Prints out "Hello, world >readme"
14. $ echo "Hello, world" > readme Expectation – Prints out "Hello, world" and returns error Results – Made a folder in week1 named readme

15. $ cat readme Expectation – Reads what is inside the file Results – Reads what is inside the file
16. $ example="Hello, World" Results – Creates variable named example and stores hello, world
17. $ echo $example Expectation – Prints out Hello, World Results – Prints out Hello, World
18. $ echo '$example' Expectation – Prints out Hello, World Results – Prints out $example
19. $ echo "$example" Expectation – Prints out $example Results – Prints out Hello, World
20. $ echo "Please enter your name."; read example Results - Stores an input from the user in example
21. $ echo "Hello $example" Results - Hello (Input)
22. $ three=1+1+1;echo $three Results - Prints out 1+1+1 as it is storede in the variable three
23. $ bc
24. $ echo 1+1+1 | bc
25. $ let three=1+1+1;echo $three Results - Can be used to calculate without using bc
26. $ echo date Results - Prints out date
27. $ cal Results - Shows the callander
28. $ which cal Results - Asks the user which callander to read from
29. $ /bin/cal Results - Reads the callender from a different directory
30. $ $(which cal) Results - Uses what ever inside the brackets and runs it
31. $ `which cal` Results - error
32. $ echo "The date is $(date)" Results - Prints out a string and the the date at the end of it
33. $ seq 0 9 Results - Writes all the numbers from 0 - 9
34. $ seq 0 9 | wc -l Results - Counts how many lines are outputted
35. $ seq 0 9 > sequence Results - Stores the 0 - 9 in sequence file
36. $ wc -l < sequence Results - Shows the length of the data insidebthe file sequence
37. $ for I in $(seq 1 9) ; do echo $I ; done Results - Creates a loop for how many values in sequence 1 - 9 print I
38. $ (echo -n 0 ; for I in $(seq 1 9) ; do echo -n +$I ; done ; echo) | bc
39. $ echo -e '::include \nint main(void)
    {
    printf("Hello World\n");
    return 0;
    }' > hello.c Results - Stores the command for hello world in hello.c file
40. $ cat hello.c Results - Reads the file
41. $ gcc hello.c -o hello Results - Compiles the file
42. $ ./hello Results - Runs the file

Chapter 5

Report Week2

1. $ cd ∼ change directory Expectations- Exits all the folder Results - Exits all the folder
2. $ git init portfolio Results - Creates a git repository called "portfolio"
3. $ cd portfolio Expectations - Changes directory to portfolio Results - Changes directory to portfolio
4. $ ls -al Expectations - Display all the files in the folder Results - Displays all the files that are stored in the directory
5. $ git status Expectations - Shows the current status of the directory Results- Shows current status of the directory to show if any changes has been made
6. $ echo hello > .gitignore – stores “hello” under file name “.gitignore” Expectations - Prints hello and moves it into .gitignore file Results - Stores the word "hello" into .gitignore
7. $ git add -A Expectaion - Adds all updated files into the directory Results - Updates to the main library with all the changes that has happened
8. $ git status Expectations - Shows the current status of the directory Results- Shows current status of the directory to show if any changes has been made
9. $ git config –global user.email “” Results - Allows the user to connect and link there email for commits to csgitlab
10. $ git config –global user.name “” Results - Allows the user to connect and link there username for commits to csgitlab
11. $ git commit -m “first commit, adding week 1 content” Expectaion - Pushesh all the recent chnages that have all been commited into csgitlab Results - Stores and updates csgitlabe with all the changes
12. $ git status Expectations - Shows the current status of the directory Results- Shows current status of the directory to show if any changes has been made
13. $ git push Expectaion - Pushesh all the recent chnages that have all been commited into csgitlab Results - Stores and updates csgitlabe with all the changes
14. $ git remote add origin https://csgitlab.reading.ac.uk//cs1pc20_portfolio.git Results - Connects to the remote csgtitlab repository to allow pushing files to save them
15. $ git push –set-upstream origin master Results - Pushesh all the recent chnages that have all been commited into csgitlab into a new branch that is created
16. $ git status Expectations - Shows the current status of the directory Results- Shows current status of the directory to show if any changes has been made
17. $ echo “# CS1PC20 Portfolio” > readme.md Expectaion - Print "# CS1PC20 Portfolio” Results - Saves the "# CS1PC20 Portfolio” into a readme.md file

18. $ git add readme.md Expectaion - Adds the readme.md file for the next commit Results - Updates the git and adds readme.md file for the commit
19. $ git commit -m “added readme file” Expectaion - Pushesh all the recent chnages that have all been commited into csgitlab Results - Stores and updates csgitlabe with all the changes
20. $ git push Expectaion - Pushesh all the recent chnages that have all been commited into csgitlab Results - Stores and updates csgitlabe with all the changes
21. $ git config –global credential.helper cache Results - Stores the username and password so we dont have to type it everytime
22. $ git branch week2 Expectation Creates new branch called week2 where new files can be created and stored off from the masters branch Results - Creates a new branch from the master branch called week2
23. $ git checkout week2 Results - Used to navigate between the branches created by the git and store all the new commits on that branch
24. $ mkdir week2 Expectation – Will create a folder named week2 Results – Creates the folder named week2
25. $ echo “# Week 2 exercise observations” > week2/report.md Expectaion - Saves "# Week 2 exercise observations” into week2 folder and then into a report.md file Results - Expectaion - Saves "# Week 2 exercise observations” into week2 folder and then into a report.md file
26. $ git status Expectations - Shows the current status of the directory Results- Shows current status of the directory to show if any changes has been made
27. $ git add week2 – adds week 2 to next commit Expectaion - Adds the week2 file for the next commit Results - Updates the git and adds week2 file for the commit
28. $ git commit -m “added week 2 folder and report.md” Expectaion - Pushesh all the recent chnages that have all been commited into csgitlab Results - Stores and updates csgitlabe with all the changes

1. $ git push Expectaion - Pushesh all the recent chnages that have all been commited into csgitlab Results - Stores and updates csgitlabe with all the changes
2. $ git checkout master Results - Switches to master branch and saves all the new commits
3. $ ls -al Expectations - Display all the files in the folder Results - Displays all the files that are stored in the directory
4. $ git checkout week2 Results - Switches to week2 branch and saves all the new commits
5. $ ls -al Expectations - Display all the files in the folder Results - Displays all the files that are stored in the directory
6. $ git checkout master Results - Switches to master branch and saves all the new commits
7. $ git merge week2 Expectaion - Merges week2 into Master branch Results - Megers week2 branch into Master
8. $ ls -al Expectations - Display all the files in the folder Results - Displays all the files that are stored in the directory
9. $ git push Expectaion - Pushesh all the recent chnages that have all been commited into csgitlab Results - Stores and updates csgitlabe with all the changes
10. $ rm -r week2 Expectaion - Removes week2 directory Results - Removes week2 directory
11. $ rm -r week1 Expectaion - Removes week1 directory Results - Removes week1 directory
12. $ ls -al Expectations - Display all the files in the folder Results - Displays all the files that are stored in the directory
13. $ git status Expectations - Shows the current status of the directory Results- Shows current status of the directory to show if any changes has been made

14. $ git stash Expectaion - Saves stash file Results - Temporarily saves any of the uncommited changes
15. $ git stash drop Expectaion - Drops any saved changes temporarily Results - Drops any temporarily saved changes
16. $ ls -al Expectations - Display all the files in the folder Results - Displays all the files that are stored in the directory
17. $ cd ∼ Expectations- Exits all the folder Results - Exits all the folder
18. $ cp -r portfolio portfolio_backup – copies the portfolio file under the name “portfolio_backup” Expectaion - Creates a new backup directory called portfolio_backup Results - Copies portfolio and creates portfolio_↩ backup
19. $ rm -rf portfolio Expectaion - Removes portfolio directory and everything stored in it Results - Force removes portfolio directory
20. $ ls -al Expectations - Display all the files in the folder Results - Displays all the files that are stored in the directory
21. $ git clone https://csgitlab.reading.ac.uk//cs1pc20_portfolio portfolio Results - Clones all the data from csgitlab
22. $ ls -al Expectations - Display all the files in the folder Results - Displays all the files that are stored in the directory

Chapter 6

Week 2 Exercise Observations

Chapter 7

c-programs

Week3 Report

1. $ cd ∼ Expectation - Exits all the folders Results - Exits all the folders
2. $ cd portfolio Expectation - Changes the directory to portfolio folders Results - Changes the directory to portfolio folders
3. $ mkdir week3 Expectation - Creates a folder named week3 in portfolio Results - Creates a folder named week3 in portfolio
4. $ mkdir week3/greeting Expectation - Creates a greeting folder inside week3 folder Results - Creates a greeting folder inside week3 folder
5. $ cd week3/greeting Expectation - Enters week3/greeting directory Results - Enters week3/greeting directory
6. $ git branch greeting Expectation - Creates new branch called greetings where new files can be created and stored off from the masters branch Results - Creates a new branch from the master branch called greeting
7. $ git switch greeting Expectaion - Switches over to the newly created branch called greeting Results - Switches to greeting branch
8. Create a file called greeting.c with the following contents: Create a nano fille called greeting.c and write a simple hellow world code in c which will print "Hello world!" when it has been executed.
9. $ gcc -Wall -pedantic -c greeting.c -o greeting.o I am not to sure what this line actually means but it is something to do with compiling the greeting.c file and checking for any bugs and printing out the "Hello world!"
10. Create a file called test_result.c with the following contents: Create a nano file called test_result.c which will include the greeting.h file which will be created after wards in a new nano file to check if the code returns 0
11. And create a file called greeting.h with the following contents: Create a nano file called greeting.h which will have void command value stored which will be used for test_result.c file to return the value
12. $ echo greeting.o >> ∼/portfolio/.gitignore Expectaion - Moves the greeting.o into different location - portfolio folder and then moves it into .gitignore file Results - Moves the greeting.o into portfolio/.gitignore file
13. $ echo libgreet.a >> ∼/portfolio/.gitignore Expectaion - Moves the greeting.o into different location - portfolio folder and then moves it into .gitignore file Results - Moves the greeting.o into portfolio/.gitignore file
14. $ ar rv libgreet.a greeting.o Results - Archives the libgreet.o file into libgreet.a file
15. $ gcc test_result.c -o test1 -L. -lgreet -I. Expectation - Compiles test_result.c Results - Compiles the files
16. $ ./test1 Expectaion - Run the file Results - Runs the file

17. $ git add -A Expectaion - Adds all updated files into the directory Results - Updates to the main library with all the changes that has happened
18. $ git commit -m “greeting library and greeting test program” Expectaions - Saves/Commits the files changes with the message of what is being saved Results - Updates the directory with all the changes
19. $ git push Expectaion - Pushesh all the recent chnages that have all been commited into csgitlab Results - Stores and updates csgitlabe with all the changes
20. $ cd ∼/portfolio/week3 Expectaion - Changes the directory to week3 Results - Changes the directory to week3
21. $ git switch master Expectaion - Switches back to the master branch Results - Switches back to the master branch
22. $ git branch vectors Expectation - Creates new branch called vectors where new files can be created and stored off from the masters branch Results - Creates a new branch from the master branch called vectors
23. $ git switch vectors Expectaion - Switches over to the newly created branch called vectors Results - Switches to greeting vectors
24. $ mkdir vectors Expectaion - Makes a new folder called vectors Results - Makes a new folder called vectors
25. $ cd vectors Expectaion - Changes the directory to vectors folder Results - Changes the directory to vectors folder
26. Create a file called vector.h with the following contents: Create a new nano file called vector.h and defines SIZ with three different variables x,y and z with int
27. Create a file called test_vector_add.c with the following contents: Creates a new nano file called test_vector_add.c which will have all the inputs for the different variables x,y and z
28. And now create the code to actually “do the math” – vector.c Creates a new nano file called vector.c and will contain the code to see if they are the right size and that they have been declared
29. $ gcc -Wall -pedantic -c vector.c -o vector.o Results - Compiles the files
30. $ ar rv libvector.a vector.o Results - Archives the libvector.o file into vector.a file
31. $ gcc test_vector_add.c -o test_vector_add1 -L. -lvector -I. compiles teh test_vector_add.c and test_vector↩ _add1
32. $ ./test_vector_add1 Expectaion - Run the file Results - Runs the file
33. $ git add -A Expectaion - Adds all updated files into the directory Results - Updates to the main library with all the changes that has happened
34. $ git commit -m “code to add two vectors of fixed size” Expectaions - Saves/Commits the files changes with the message of what is being saved Results - Updates the directory with all the changes
35. $ git push Expectaion - Pushesh all the recent chnages that have all been commited into csgitlab Results - Stores and updates csgitlabe with all the changes
36. Edit vector.h so it contains this: Edits the vector.h file and add new variables for x and y that are also int
37. Edit vector.c so it contains this: Edit the vector.c file to add the new declared variables x and y for dot_product and to see if they have been declared
38. And create test_vector_dot_product.c so it contains: Create a nano filed called test_vector_dot_product.c containing that code which will be a test framework for vector library and cheks each elememnts of the return vector
39. $ gcc -Wall -pedantic -c vector.c -o vector.o Results - Compiles the files
40. $ ar rv libvector.a vector.o Results - Archives the libvector.a file into vector.o file
41. $ gcc test_vector_dot_product.c -o test_vector_dot_product1 -L. -lvector -I. Results - Compiles the files

42. $ ./test_vector_dot_product1 Expectaion - Run the file Results - Run the file
43. $ git add -A Expectaion - Adds all updated files into the directory Results - Updates to the main library with all the changes that has happened
44. $ git commit -m “code to calculate dot product of two vectors of fixed size” Expectaions - Saves/Commits the files changes with the message of what is being saved Results - Updates the directory with all the changes
45. $ git push Expectaion - Pushesh all the recent chnages that have all been commited into csgitlab Results - Stores and updates csgitlabe with all the changes

Chapter 8

make_and_test

Report Week4

1. $ cd portfolio Expectations - Changes the directory to portfolio Results- Changes the directory to portfolio
2. $ git switch master Expectations - Switches to master branch Results - Switches to master branch
3. $ mkdir -p week4/framework Expectations - Creates a week4 folder and then folder inside that named framework Results- Creates a week4 folder and then folder inside that named framework
4. $ cd week4/framework Expectations - Changes directory into week4/framework folder Results - Enters the newly created directory called framework
5. $ git branch framework Expectation Creates new branch called framework where new files can be created and stored off from the masters branch Results - Creates a new branch from the master branch called framework
6. $ git switch framework Expectation - Switches to framework branch Expectation - Switches to framework branch
7. $ nano Makefile Expectation - Creates a Makefile with the following contents Results - Creates a Makefile
8. $ cat -vTE Makefile used to check the preview of the Makefile Results - Shows Tabs and End-ofline and shouls show that "<tab>" is shown as ^I instead actaul tab
9. $ make feature NAME=test_output substitutes “test_output” Results- Created "NAME" feature to store test↩ _output
10. $ ls -al test_output Results - Displays all files in that directory
11. $ git add Makefile Results - Adds the Makefile for the next commit
12. $ git commit -m "Setting up Makefile to create feature folders" Results - Commits all the changes in the directory
13. $ git push expectaion - Pushesh all the recent chnages that have all been commited into csgitlab Results - Stores and updates csgitlabe with all the changes
14. $ cd test_output; cd src Results - Goes into the src folder
15. $ nano test_outputs.c Expectation - Create a .c file to write c program Results - Creates a .c file
16. $ gcc -Wall -pedantic test_outputs.c -o test_outputs Expectation - Compiles and checks for error Results - Compiles the file
17. $ ./test_outputs file_does_not_exist Expectation - Gives an error as files does not exist and can not be found Results - Gives error

18. $ ./test_outputs Results - Gives error
19. $ nano op_test Expectation - Creates a new file to test the code Results - Creates a new file to write a new code to test the output
20. $ ./test_outputs op_test Results - Runs the code and prints out the line spacing for test_output to match it exactly for 108 lines
21. $ git add test_outputs.c Results - Adds the test_output.c file to the next commit
22. $ git add op_test Results - Adds it to the next commit
23. $ git commit -m "test framework and sample test suite" Saves any changes made with the following message Results - Commits all the changes that has happend in the directory
24. $ git push expectaion - Pushesh all the recent chnages that have all been commited into csgitlab Results - Stores and updates csgitlabe with all the changes

Chapter 9

bases_and_reports

Week5 Report

1. $ cd portfolio Expectations - Changes directory to portfolio Results - Changes directory to portfolio
2. $ git switch master Results - Switches to master branch
3. $ git merge greeting Expectaion - Merges greeting into Master branch Results - Megers greeting branch into Master
4. $ git merge vectors Expectaion - Merges vectors into Master branch Results - Megers vectors branch into Master
5. $ git merge framework Expectaion - Merges framework into Master branch Results - Megers framework branch into Master
6. $ git branch baseconversion Expectation Creates new branch called baseconversion where new files can be created and stored off from the masters branch Results - Creates a new branch from the master branch called baseconversion
7. $ git switch baseconversion Results- Switches to baseconversion branch
8. $ mkdir week5 ; cd week5 Expectaion - Creates a Week5 filder and enters the directory week5
9. $ make -f ../week4/framework/Makefile feature NAME=dec2bin Results - Creates multiple executables files one of them is the dec2bin
10. $ cd dec2bin Results - Changes directory to dec2bin
11. $ gcc src/conv.c -o lib/conv.o -c Results - Compiles the file
12. $ ar rv lib/libconv.a lib/conv.o Results - Archives the folder
13. $ gcc src/dec2bin.c -o bin/dec2bin -Isrc -lconv -Llib Results - Compiles the file
14. $ ∼/portfolio/week4/framework/test_output/src/test_outputs test/dec2bin_tests Results - Goes to the specific files directory and executes the test file that is stored there
15. $ cd ∼ Results - Exits all the folders
16. $ git switch master
    Results - Switches to master branch
17. $ git merge baseconversion Expectaion - Merges baseconversion into Master branch Results - Megers baseconversion branch into Master
18. $ mkdir docs Expectaion - Creates a docs folder Results - Creates a docs folder

19. $ doxygen -g Expectatons - Creates the doxyfile Results - Creates the doxyfile
20. $ git add Doxyfile Expectaion - Adds the Doxyfile files into the directory changes for next commit Results - Updates to the main library with all the changes that has happened
21. $ git add submission_answers.md Expectaion - Adds submission_answers.md files into the directory changes for next commit Results - Updates to the main library with all the changes that has happened
22. $ git add _FrontPage.md Expectaion - Adds _FrontPage.md files into the directory changes for next commit Results - Updates to the main library with all the changes that has happened
23. $ git commit -m “added configured Doxyfile, answers and frontpage” Expectaion - Pushesh all the recent chnages that have all been commited into csgitlab Results - Stores and updates csgitlabe with all the changes
24. $ git push Expectaion - Pushesh all the recent chnages that have all been commited into csgitlab Results - Stores and updates csgitlabe with all the changes
25. $ doxygen Results - Creates a HTML and Latex folders
26. $ cd docs/latex Results - Changes directory into latex folder
27. $ make Results - Creates executables
28. $ git add refman.pdf Expectaion - Adds refman.pdf files into the directory changes for next commit Results - Updates to the main library with all the changes that has happened
29. $ git commit -m “adding documentation” Expectaion - Pushesh all the recent chnages that have all been commited into csgitlab Results - Stores and updates csgitlabe with all the changes
30. $ git push Expectaion - Pushesh all the recent chnages that have all been commited into csgitlab Results - Stores and updates csgitlabe with all the changes

Chapter 10

File Index

10.1 File List

Here is a list of all files with brief descriptions:

Chapter 11

File Documentation

11.1 _FrontPage.md File Reference

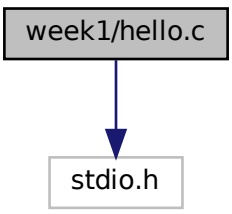
11.2 readme.md File Reference

11.3 submission_answers.md File Reference

11.4 week1/hello.c File Reference

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

Include dependency graph for hello.c:



Functions

- int main (void)

11.4.1 Function Documentation

11.4.1.1 main()

```
int main (
            void  )
```

11.5 week1/introductory-bash.md File Reference

11.6 week2/introductory-git.md File Reference

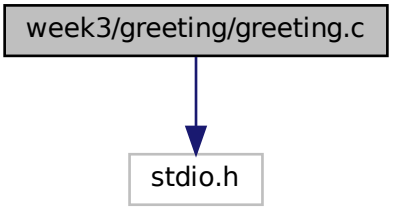
11.7 week2/report.md File Reference

11.8 week3/c-programs.md File Reference

11.9 week3/greeting/greeting.c File Reference

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

Include dependency graph for greeting.c:



Functions

- int greet (void)

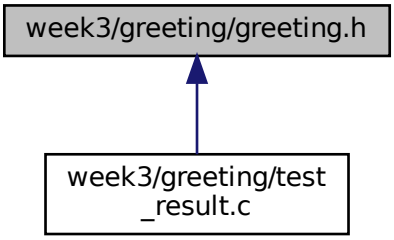
11.9.1 Function Documentation

11.9.1.1 greet()

```
int greet (
            void  )
```

11.10 week3/greeting/greeting.h File Reference

This graph shows which files directly or indirectly include this file:



Functions

- int greet (void)

11.10.1 Function Documentation

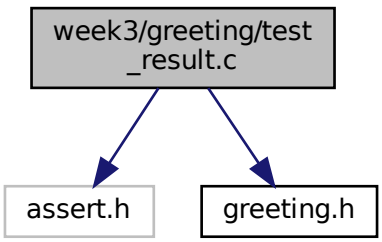
11.10.1.1 greet()

```
int greet (
            void  )
```

11.11 week3/greeting/test_result.c File Reference

```
#include <assert.h>
#include "greeting.h"
```

Include dependency graph for test_result.c:



Functions

- int main (void)

11.11.1 Function Documentation

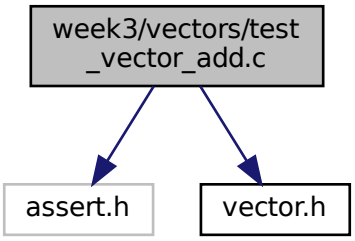
11.11.1.1 main()

```
int main (
            void  )
```

11.12 week3/vectors/test_vector_add.c File Reference

```
#include <assert.h>
#include "vector.h"
```

Include dependency graph for test_vector_add.c:



Functions

- int main (void)

11.12.1 Function Documentation

11.12.1.1 main()

```
int main (
            void  )
```

This is a simple test framework for vector library xvec and yvec will be the inputs to the vector arithmetics and zvec will take the return value

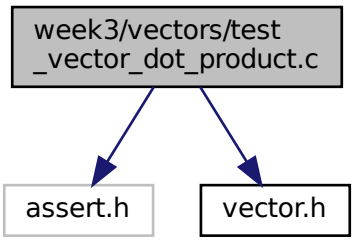
Checks each of the returned vector

If the asserts works and there wasn't any errors returns 0

11.13 week3/vectors/test_vector_dot_product.c File Reference

```
#include <assert.h>
#include "vector.h"
```

Include dependency graph for test_vector_dot_product.c:



Functions

- int main (void)

11.13.1 Function Documentation

11.13.1.1 main()

```
int main (
            void  )
```

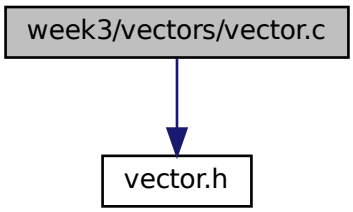
Simple test framework for vector library xvec and yvec will be the input values for the vector arithmetic routine

check each element value of the returned vector

11.14 week3/vectors/vector.c File Reference

```
#include "vector.h"
```

Include dependency graph for vector.c:



Functions

- int add_vectors (int x[ ], int y[ ], int z[ ])
- int dot_product (int x[ ], int y[ ])

11.14.1 Function Documentation

11.14.1.1 add_vectors()

```
int add_vectors (
            int x[],
            int y[],
            int z[] )
```

It is a simple fixed size vector for addition routine, adds each elements x to corresponding elements of y, storing the answer in z.

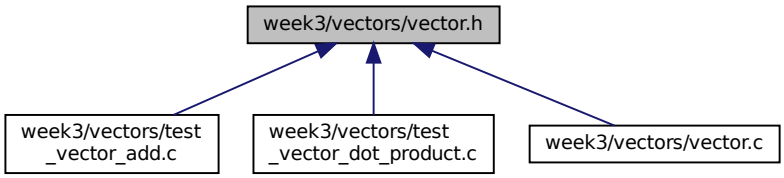
11.14.1.2 dot_product()

```
int dot_product (
            int x[],
            int y[] )
```

Fixed size dot product routine, multiply element x to corresponding element of y, adding up totals. Return the actual value that has been calculated. res is a local variable to hold the result that is calculated

11.15 week3/vectors/vector.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define SIZ 3

Functions

- int add_vectors (int x[ ], int y[ ], int z[ ])
- int dot_product (int x[ ], int y[ ])

11.15.1 Macro Definition Documentation

11.15.1.1 SIZ

```
#define SIZ 3
```

11.15.2 Function Documentation

11.15.2.1 add_vectors()

```
int add_vectors (
            int x[],
            int y[],
            int z[] )
```

It is a simple fixed size vector for addition routine, adds each elements x to corresponding elements of y, storing the answer in z.

11.15.2.2 dot_product()

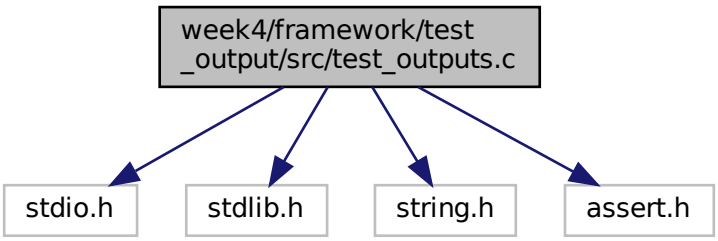
```
int dot_product (
            int x[],
            int y[] )
```

Fixed size dot product routine, multiply element x to corresponding element of y, adding up totals. Return the actual value that has been calculated. res is a local variable to hold the result that is calculated

11.16 week4/framework/test_output/src/test_outputs.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
```

Include dependency graph for test_outputs.c:



Macros

- #define COM_SIZ 60
- #define ARG_SIZ 1024
- #define RES_SIZ 1024

Functions

- int main (int argc, char ∗argv[ ])

11.16.1 Macro Definition Documentation

11.16.1.1 ARG_SIZ

```
#define ARG_SIZ 1024
```

11.16.1.2 COM_SIZ

```
#define COM_SIZ 60
```

define some constant values for size of data noting of course that if your data needs bigger values, you have to edit the source code and change the constants defined here

11.16.1.3 RES_SIZ

```
#define RES_SIZ 1024
```

11.16.2 Function Documentation

11.16.2.1 main()

```
int main (
            int argc,
            char * argv[] )
```

This test program calls an existing executable and checks the outputs to standard output meet the expected values. It should be called with: test_outputs <filename which contains test definitions> < fp is a pointer used to give access to the file descriptor of the pipe

try to open the file named on the command line

we will read each line from the file. These should be structured as: command to run inputs expected output

Note: this could go horribly wrong if the input file is not properly formatted

string handling in C can be cumbersome. typically suggestions online make use of "malloc" and "strcpy" and "strcat" but these complicate things and are arguably not good practice strtok gives us a useful shortcut to remove newlines (the way it is used here)

Now we call the command, with the arguments and capture the result so we can compare it to the expected result. the "popen" command opens a special type of 'file' called a 'pipe'

This is how we get the result back out of the pipe we opened after reading the result in to "actual" strcmp is slightly unusual - it returns 0 if the strings are the same, >0 if string1 is bigger than string2, and <0 if string1 is less than string2 because 0 is 'false', we negate (!) the they are the same.

we create a message to let us know what was expected and what we got note that we have split the line in the next statement – that is fine, we can!

Because we want the test suite to keep running, we use an if statement rather than the assert function

if we don't close file handles, we risk using up the machines resources

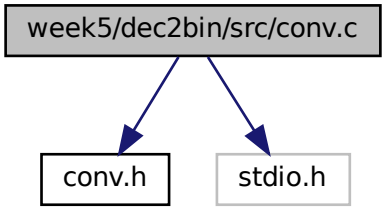
11.17 week4/make_and_test.md File Reference

11.18 week5/dec2bin/bases_and_reports.md File Reference

11.19 week5/dec2bin/src/conv.c File Reference

```
#include "conv.h"
#include <stdio.h>
```

Include dependency graph for conv.c:



Functions

- void dec2r (char in[ ], int r, char out[ ])

11.19.1 Function Documentation

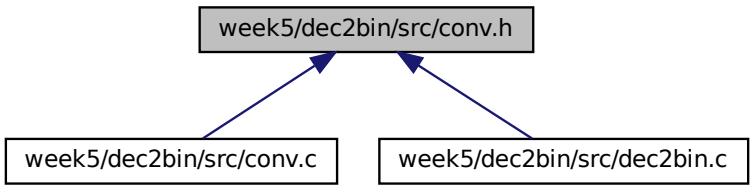
11.19.1.1 dec2r()

```
void dec2r (
            char in[],
            int r,
            char out[] )
```

convert a string from base 10 to another base <=10 and >1 (!) limit inputs to non-negative integers also assume (never a good idea!) that input string is a valid number can consider other values later...!

11.20 week5/dec2bin/src/conv.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define STRLEN 20

Functions

- void dec2r (char in[ ], int r, char[ ])

11.20.1 Macro Definition Documentation

11.20.1.1 STRLEN

```
#define STRLEN 20
```

11.20.2 Function Documentation

11.20.2.1 dec2r()

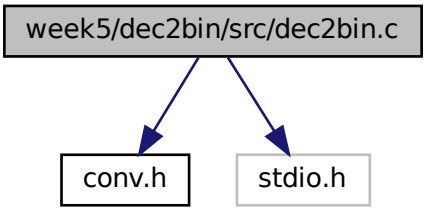
```
void dec2r (
            char in[],
            int r,
            char out[] )
```

convert a string from base 10 to another base <=10 and >1 (!) limit inputs to non-negative integers also assume (never a good idea!) that input string is a valid number can consider other values later...!

11.21 week5/dec2bin/src/dec2bin.c File Reference

```
#include "conv.h"
#include <stdio.h>
```

Include dependency graph for dec2bin.c:



Functions

- int main (int argc, char ∗argv[ ])

11.21.1 Function Documentation

11.21.1.1 main()

```
int main (
            int argc,
            char * argv[] )
```

requires a decimal value as the single command line arguement

Index