sx004098

Generated by Doxygen 1.8.17

Chapter 1

_FrontPage

Module Code: CS1PC20

Assignment report Title: Portfolio

Student Number: 30004098

Date: 31/10/2021

Actual hrs spent on assignment: 15

Assignment evaluation: Good introduction to linux command line and C language

Chapter 2

CS1PC20 Portfolio

Chapter 3

submission_answers

#Question 1 It is important to write code libraries incase you need to call a certain function very often so instead of rewriting code unnecessarily, in the long run, libraries will save time. In smaller projects it may not be as essential, as it may infact impact the time taken to complete the project

#Question 2 Include directives allow for you to call code from another file in to your code, and allow you to use the function of that code. It can save time if you're gonna use the same lines of codes but in different files. These files often end with .h and are referred to as headers or interfaces. An interface is a header file that has a different file extension than the implementation file, e.g using the .h file with a .c

Chapter 4

introductory-bash

1. $ mkdir -p $HOME/portfolio/week1 ; cd $HOME/portfolio/week1
2. $ cd ∼
3. $ rm -r portfolio
4. $ mkdir -p $HOME/portfolio/week1 & cd $HOME/portfolio/week1
5. $ cd ∼
6. $ rm -r portfolio
7. $ mkdir -p $HOME/portfolio/week1 && cd $HOME/portfolio/week1
8. $ echo "Hello World"
9. $ echo Hello, World
10. $ echo Hello, world; Foo bar
11. $ echo Hello, world!
12. $ echo "line one";echo "line two"
13. $ echo "Hello, world > readme"
14. $ echo "Hello, world" > readme
15. $ cat readme
16. $ example="Hello, World"
17. $ echo $example
18. $ echo '$example'
19. $ echo "$example"
20. $ echo "Please enter your name."; read example
21. $ echo "Hello $example"
22. $ three=1+1+1;echo $three
23. $ bc
24. $ echo 1+1+1 | bc
25. $ let three=1+1+1;echo $three CS1PC20, Exercise – Week 1 3/5

26. $ echo date
27. $ cal
28. $ which cal
29. $ /bin/cal
30. $ $(which cal)
31. $ ‘which cal‘
32. $ echo "The date is $(date)"
33. $ seq 0 9
34. $ seq 0 9 | wc -l
35. $ seq 0 9 > sequence
36. $ wc -l < sequence
37. $ for I in $(seq 1 9) ; do echo $I ; done
38. $ (echo -n 0 ; for I in $(seq 1 9) ; do echo -n +$I ; done ; echo) | bc
39. $ echo -e ‘::include <stdio.h>\nint main(void)
    {
    printf(“Hello World\n”);
    return 0;
    }’ > hello.c
40. $ cat hello.c
41. $ gcc hello.c -o hello
42. $ ./hello

Chapter 5

report

1 sudo apt update 2 gcc 3 sudo apt install gcc 4 make 5 sudo apt install make 6 sudo apt install doxygen 7 sudo apt install git 8 sudo apt install textlive-latex-base 9 sudo apt install texlive-latex-base 10 sudo apt install texlive-fonts-recommended texlive-fonts-extra texlive-latex-extra 11 sudo apt install plantuml 12 sudo apt install graphviz 13 shotdown now 14 sudo shutdown now 15 vim 16 nano 17 ls -al 18 history 19 vim .bash_history 20 sudo shutdown now 21 exit 22 sudo shutown now 23 sudo shutdown now 24 ls 25 mkdir -p $HOME/portfolio/week1 26 cd ∼ 27 ls 28 rm -r portfolio 29 ls 30 mkdir -p $HOME/portfolio/week1 31 cd ∼ 32∗ cd $HOME/portfolio/week 33 cd∼ 34 cd ∼ 35 rm -r portfolio/ 36 echo "Hello World" 37 echo Hello World 38 echo Hello, World 39 echo Hello, World; Foo bar 40 echo Hello, World! 41 echo "line one";echo "line two" 42 echo "Hello, world > readme" 43 echo "Hello, world" > readme 44 ls 45 cat readme 46 echo a 47 echo 'a' 48 echo "a" 49 echo "a" 50 echo $example 51 echo '$example' 52 echo "$example" 53 ls 54 bc 55 echo "Please enter your name";read example 56 echo "hello $example" 57 three=1+1+1;echo$three 58 three=1+1+1;echo $three -—> Only prints 1+1+1 59 bc 60 echo 1+1+1 | bc 61 let three=1+1+1;echo $three -—> Prints 3 62 three=1+1+1;echo $three 63 echo date 64 cal 65 /bin/cal 66 $(which cal) 67 'which cal' 68 cd /bin/cal 69 echo "The date is $(date)" 70 sez 0 9 71 sec 0 9 72 seq 0 9 73 seq 0 9 | wc -l 74 sec 0 9 > sequence 75 seq0 9 > sequence 76 seq 0 9 > sequence 77 cat sequence 78 wc -l < sequence 79 for i in $(seq 1 9); do echo $i;done 80 seq 0 9 | wc -l 81 cat sequence 82 for i in $(seq 1 9); do echo $i ; done 83 for i in $(seq 1 9); do echo $l ; done 84 for i in $(seq 1 9); do echo $i ; done 85 for i in $(seq 1 9); do echo $i stop; done 86 echo -e '#include <stdio.h>\nint main(void)

Chapter 6

Week 2 exercise observations

1. $ cd $\sim$ – returned to home directory
2. $ git init portfolio – made a git repository in home/student/portfolio/.git/ idk what git is yet tho
3. $ cd portfolio – changes directory to portfolio
4. $ ls -al – lists all folders within portfolio, including .git
5. $ git status – on branch master, no commits? what is that?
6. $ echo hello $>$ .gitignore – puts hello in to .gitignore
7. $ git add -A – no idea what this does – it added all new files to git i think
8. $ git status – tells us that there is a new file '.gitignore' that has not been commited yet
9. $ git config –global user.email “sx004098@student.reading.ac.uk” – makes the email as the global email for machine
10. $ git config –global user.name “Aadil Sattar” – makes the name as the global name for machine
11. $ git commit -m “first commit, adding week 1 content” – commits the first edit and labels it (this can allow you to go back)
12. $ git status – shows us the status of the git, which has now been committed
13. $ git push – this shows us how to push it to git
14. $ git remote add origin https://csgitlab.reading.ac.uk/sx004098/cs1pc20_↩portfolio.git – connects to csgitlab.reading.ac.uk at the cs1pc20_portfolio.git file
15. $ git push -–set-upstream origin master – pushs it to the git file, while also asking you to log in
16. $ git status – no commits
17. $ echo “# CS1PC20 Portfolio” $>$ readme.md
18. $ git add readme.md
19. $ git commit -m “added readme file” – connotes the readme.md change
20. $ git push – pushes it to csgitlab
21. $ git config —global credential.helper cache – ???
22. $ git branch week2 – creates a git branch named week2
23. $ git checkout week2 – switches to week2 branch

24. $ mkdir week2 – creates week2 directory
25. $ echo “# Week 2 exercise observations” > week2/report.md – adds the words to report.md in week2
26. $ git status – report.md not added in week2
27. $ git add week2 – adds in report.md to week2
28. $ git commit -m “added week 2 folder and report.md” – commits the new change
29. $ git push – pushes this new change to to csgitlab
30. $ git checkout master – switches to master branch
31. $ ls -al – lists everything in the master branch
32. $ git checkout week2 – switches to week2 branch
33. $ ls -al – lists everything in the week2 branch
34. $ git checkout master – switches back to master branch
35. $ git merge week2 – already up to date/merges master and week2 to master
36. $ ls -al – week2 stuff now in master
37. $ git push – already up to date
38. $ rm -r week2 – removes week2 branch
39. $ rm -r week1 – removes week1 branch
40. $ ls -al – lists master branch with no other merged
41. $ git status – tells us we haven't updated on deleting the branch
42. $ git stash – displays state
43. $ git stash drop – brings the state back
44. $ ls -al – week2 is back!!

1. $ cd ~ – goes back to home directory
2. $ cp -r portfolio portfolio_backup – creates backup of portfolio
3. $ rm -rf portfolio – removes original portfolio
4. $ ls -al – only portfolio backup remains
5. $ git clone https://csgitlab.reading.ac.uk/sx004098/cs1pc20_portfolio portfolio – clones in from portfolio
6. $ ls -al – portfolio is back!!

Chapter 7

Week 2 exercise observations

Chapter 8

Week 3 Observations

1. $ cd ~ – Returns to home directory
2. $ cd portfolio – Goes in to portfolio
3. $ mkdir week3 – Creates directory week3
4. $ mkdir week3/greeting – Creates directory greeting within week3 directory
5. $ cd week3/greeting – Go in to greeting directory
6. $ git branch greeting – Create git branch know as greeting
7. $ git switch greeting – Switch to greeting branch greeting
8. $ nano greeting.c – then:
   #include <stdio.h> int greet(void) { printf("Hello world!\n"); return 0; }
9. $ gcc -Wall -pedantic -c greeting.c -o greeting.o – creates output file
10. $ nano test_result.c – then:
    #include <assert.h> #include "greeting.h" int main(void) { assert(0==greet()); return 0; }
11. $ nano greeting.h – then:
    int greet(void);
12. $ echo greeting.o >> ~/portfolio/.gitignore – adds the greeting.o to the end of .gitignore
13. $ echo libgreet.a >> ~/portfolio/.gitignore – adds the greeting.o to the end of .gitignore
14. $ ar rv libgreet.a greeting.o – creates libgreet.a library and adds greeting.o to it
15. $ gcc test_result.c -o test1 -L. -lgreet -I. – compiles a program known as test_result.c and outputing it in test1, and to tell it to look for the library
16. $ ./test1 – shows the output in the test1
17. $ git add -A
18. $ git commit -m “greeting library and greeting test program” – annotates the change
19. $ git push – pushes it to csgitlab
20. $ cd ~/portfolio/week3 – switches to directory week3 in portfolio
21. $ git switch master – switches to master branch
22. $ git branch vectors – creates vectors off master

23. $ git switch vectors – swtiches in to the vectors branch
24. $ mkdir vectors – creates directory "vectors"
25. $ cd vectors – goes in to vectors directory
26. $ nano vectors.h – then:

    #define SIZ 3 int add_vectors(int x[ ], int y[ ], int z[ ]);
27. $ nano test_vector_add.c – then:

    #include <assert.h> #include "vector.h" /** A simple test framework for vector library

    - we will be improving on this later */ int main(void) { /** xvec and yvec will be inputs to our vector arithmetic routines
    - zvec will take the return value */ int xvec[SIZ]={1,2,3}; int yvec[SIZ]={5,0,2}; int zvec[SIZ]; add_vectors(xvec,yvec,zvec); /** We want to check each element of the returned vector */ assert(6==zvec[0]); assert(2==zvec[1]); assert(5==zvec[2]); /** If the asserts worked, there wasn't an error so return 0 */ return 0; }
28. $ nano vector.c – then:

    #include "vector.h" /** A simple fixed size vector addition routine

    - Add each element of x to corresponding element of y, storing answer in z
    - It is the calling codes responsibility to ensure they are the right size
    - and that they have been declared.
    - We return an error code (0 in this case showing no error), but will add the
    - program logic to handle actual errors later */ int add_vectors(int x[ ], int y[ ], int z[ ]) { for (int i=0;i<S↩ IZ;i++) z[i]=x[i]+y[i]; return 0; }
29. $ gcc -Wall -pedantic -c vector.c -o vector.o – creates the output file vector.o and asks to show all warnings
30. $ ar rv libvector.a vector.o – compiles libvector library and adding vector.o
31. $ gcc test_vector_add.c -o test_vector_add1 -L. -lvector -I. – compiling
32. $ ./test_vector_add1 – create directory, test_vector add1
33. $ git add -A
34. $ git commit -m “code to add two vectors of fixed size”
35. $ git push
36. Change the assert(5==zvec[2]); line to be assert(5==zvec[1]); and recompile test to see what happens – It doesnt work and says the assertion failed
37. $ nano vector.h – then append this to the next line:

    int dot_product(int x[ ], int y[ ], int z[ ]);
38. $ nano vector.c – then append this to the next line:

    /** A simple fixed size dot product routine

    - multiply each element of x to corresponding element of y, adding up totals
    - It is the calling codes responsibility to ensure they are the right size
    - and that they have been declared.
    - We return the actual value we have calculated
    - We may need program logic to handle actual errors later */

    int dot_product(int x[ ], int y[ ], int z[ ]) { /** res <- a local variable to hold the result as we calculate it */ int res = 0; for (int i=0;i<SIZ;i++) res=res + x[i]*y[i]; return res; }

39. $ nano test_vector_dot_product.c – then:

    #include <assert.h> #include "vector.h" /** A simple test framework for vector library */ int main(void) { /** xvec and yvec will be inputs to our vector arithmetic routines */ int xvec[SIZ]={1,2,3}; int yvec[SIZ]={5,0,2}; int result; result=dot_product(xvec,yvec); /** We want to check each element of the returned vector */ assert(11==result); return 0; }

40. $ gcc -Wall -pedantic -c vector.c -o vector.o – compiles the file

41. $ ar rv libvector.a vector.o – idk what this does – creates an archive of vector.o in libvector.a

42. $ gcc test_vector_dot_product.c -o test_vector_dot_product1 -L. -lvector - I. – compiles the code

43. $ ./test_vector_dot_product1 – runs the code in test_vector_dot_product1

44. $ git add -A – adds all things that have changed to git

45. $ git commit -m “code to calculate dot product of two vectors of fixed size” – commits the new changes

46. $ git push – pushes it to csgitlab

Chapter 9

my_work_log

1. $ cd ∼;cd portfolio – goes to home directory, then protfolio within that
2. $ git switch master – switches to master branch – already on it
3. $ mkdir -p week4/framework – creates week4 directory, then framework directory within that
4. $ cd week4/framework – goes in to framework directory within week4 directory
5. $ git branch framework – creates framework branch off origin
6. $ git switch framework – switches to framework branch
7. $ nano Makefile – then (indent where there is a <tab>):

feature: <tab>mkdir ;\ <tab>cd && \ <tab>mkdir bin doc src test lib config ;\ <tab>echo “∗” > bin/.gitignore ;\ <tab>echo “∗” > lib/.gitignore

1. $ cat -vTE Makefile – allows us to observe if we have put the tabs(indents) in the right place
2. $ make feature NAME=test_output – idk what this would do – uses the Makefile feature to create a bunch of folders within directory test_output
3. $ ls -al test_output – shows us the files within test_output
4. $ git add Makefile – adds Makefile feature to git
5. $ git commit -m "Setting up Makefile to create feature folders" – commits the new change
6. $ git push – pushes changes to csgitlab
7. $ cd test_output; cd src – goes in to file src within test_output
8. $ nano test_output.c – then:

#include <stdio.h> #include <stdlib.h> #include <string.h> #include <assert.h> /∗∗ 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 ∗/ #define COM_SIZ 60 #define ARG_SIZ 1024 #define RES_SIZ 1024 /∗∗

Chapter 10

bases_and_reports

1. $ cd ∼; cd portfolio – switches to home folder, then goes in to portfolio folder
2. $ git switch master – switches to master branch
3. $ git merge greeting – merges greeting branch on to master
4. $ git merge vectors – merges vectors branch on to master
5. $ git merge framework – merges framework branch on to master
6. $ git branch baseconversion – creates branch baseconversion on origin
7. $ git switch baseconversion – switches to baseconversion branch
8. $ mkdir week5 ; cd week5 – creates week5 directory then goes in to it
9. $ make -f ../week4/framework/Makefile feature NAME=dec2bin – creates feature dec2bin with Makefile from week4 folder
10. $ cd dec2bin – goes in to the directory dec2bin
11. $ nano test/dec2bin_tests – this creates the directory test, then the file dec2bin_tests which includes:

bin/dec2bin 0 0 bin/dec2bin 1 1 bin/dec2bin 8 1000 bin/dec2bin 10 1010

1. $ nano src/conv.h – creates conv.h file in src, then add:

#define STRLEN 20 void dec2r(char in[ ], int r, char out[ ]);

1. $ nano src/conv.c – creates conv.c file in src, then add:

#include "conv.h" #include <stdio.h> /∗∗ 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 the input string is a valid number
- can consider other values later...! ∗/ void dec2r(char in[ ], int r, char out[ ]) { int decval; sscanf(in, "%d",&decval); int pos=STRLEN-1; out[pos-1]='0'; while (decval > 0) { out[–pos]=(decval % r) + '0'; decval /= r; } return; }

1. $ gcc src/conv.c -o lib/conv.o -c – compiles the code, with conv.o containing the output
2. $ ar rv lib/libconv.a lib/conv.o – archives the contents within lib/libconv.a
3. $ nano src/dec2bin.c – then:

#include "conv.h" #include <stdio.h> int main(int argc, char * argv[ ]) { /** requires a decimal value as the single command line argument */ int num; char output[STRLEN]={[0 ... 18] = ' ', [19]='\0'}; dec2r(argv[1],2,output); printf("%s\n", output); return 0; }

1. $ gcc src/dec2bin.c -o bin/dec2bin -Isrc -lconv -Llib – compiles the code, i dont know what the -etc do
2. $ ~/portfolio/week4/framework/test_output/src/test_outputs test/dec2bin_tests – uses the work in the last week folder to validate the code
3. $ cd ~;cd portfolio – goes back to home folder, then portfolio within that
4. $ git switch master – switches to master branch on git
5. $ git merge baseconversion – merges baseconverson branch on to master
6. $ mkdir docs – creates docs directory
7. $ doxygen -g – uses doxygen to compiles a report
8. $ git add Doxyfile – adds Doxygen file to git
9. $ nano submission_answers.md – then:

Answers to courswork questions

1. $ nano _FrontPage.md – then:

Module Code: CS1PC20 Assignment report Title: Portfolio Student Number (e.g. 25098635): Date (when the work completed): Actual hrs spent for the assignment: Assignment evaluation (3 key points):

1. $ git add submission_answers.md – adds submissions_answers.md to git
2. $ git add _FrontPage.md – adds _FrontPage.md to git
3. $ git commit -m “added configured Doxyfile, answers and frontpage” – commits all the new items to git
4. $ git push – pushes everything to csgitlab
5. $ doxygen – idk what the point of this one is – it creates a latex and html folder in docs
6. $ cd docs/latex
7. $ make – idk what this does
8. $ git add refman.pdf – adds refman.pdf to get
9. $ git commit -m “adding documentation” – commits refman.pdf
10. $ git push – pushes new changes to csgitlab

Chapter 11

File Index

11.1 File List

Here is a list of all files with brief descriptions:

Chapter 12

File Documentation

12.1 _FrontPage.md File Reference

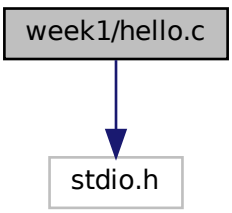
12.2 readme.md File Reference

12.3 submission_answers.md File Reference

12.4 week1/hello.c File Reference

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

Include dependency graph for hello.c:



Functions

- int main (void)

12.4.1 Function Documentation

12.4.1.1 main()

```
int main (
            void  )
```

12.5 week1/introductory-bash.md File Reference

12.6 week1/report.md File Reference

12.7 week2/report.md File Reference

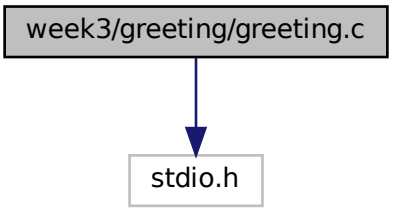
12.8 week2/introductory-git.md File Reference

12.9 week3/c-programs.md File Reference

12.10 week3/greeting/greeting.c File Reference

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

Include dependency graph for greeting.c:



Functions

- int greet (void)

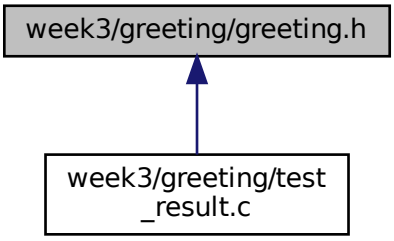
12.10.1 Function Documentation

12.10.1.1 greet()

```
int greet (
            void  )
```

12.11 week3/greeting/greeting.h File Reference

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



Functions

- int greet (void)

12.11.1 Function Documentation

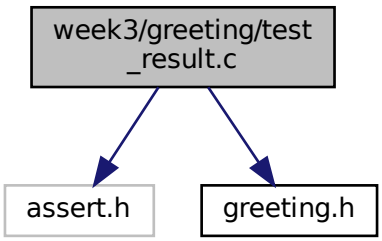
12.11.1.1 greet()

```
int greet (
            void  )
```

12.12 week3/greeting/test_result.c File Reference

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

Include dependency graph for test_result.c:



Functions

- int main (void)

12.12.1 Function Documentation

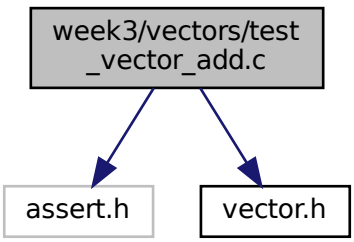
12.12.1.1 main()

```
int main (
            void  )
```

12.13 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)

12.13.1 Function Documentation

12.13.1.1 main()

```
int main (
            void  )
```

A simple test framework for vector library we will be improving on this later xvec and yvec will be inputs to our vector arithmetic routines zvec will take the return value

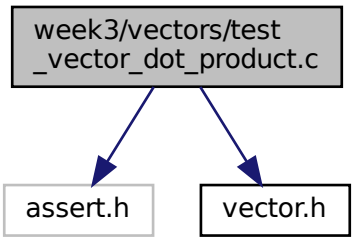
We want to check each element of the returned vector

If the assert worked, there wasn't an error so return 0

12.14 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)

12.14.1 Function Documentation

12.14.1.1 main()

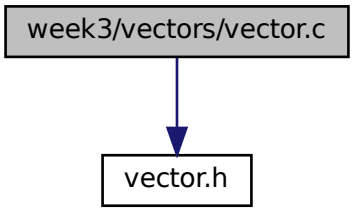
```
int main (
            void  )
```

A simple test framework for vector library xvec and yvec will be input to our vector aritmetic routines

We want to check each element of the returned vector

12.15 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[ ])

12.15.1 Function Documentation

12.15.1.1 add_vectors()

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

A simple fixed size vector addition routine Add each element of x to the corresponding element of y, storing answer in z It is the calling codes responsibility to ensure they are the right size and that they have been declared. We return an error code (0 in this case showing no error), but will add the program logic to handle actual errors later

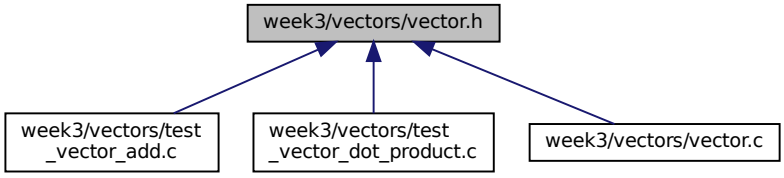
12.15.1.2 dot_product()

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

A simple fixed size dot product routine multiply each element of x to the corresponding element of y, adding up totals It is the calling codes responsibility to ensure they are the right size and that they have been declared. We return the actual value we have calculated We may need program logic to handle actual errors later res <- a local variable to hold the result as we calculate it

12.16 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[ ])

12.16.1 Macro Definition Documentation

12.16.1.1 SIZ

```
#define SIZ 3
```

12.16.2 Function Documentation

12.16.2.1 add_vectors()

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

A simple fixed size vector addition routine Add each element of x to the corresponding element of y, storing answer in z It is the calling codes responsibility to ensure they are the right size and that they have been declared. We return an error code (0 in this case showing no error), but will add the program logic to handle actual errors later

12.16.2.2 dot_product()

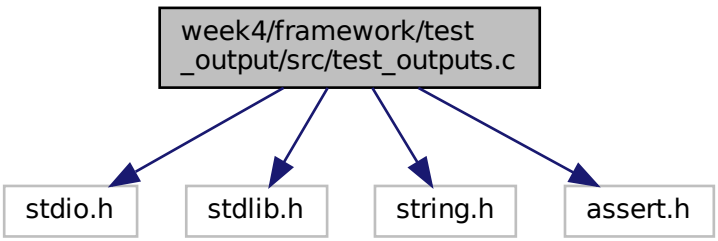
```
int dot_product (
              int x[],
              int y[] )
```

A simple fixed size dot product routine multiply each element of x to the corresponding element of y, adding up totals It is the calling codes responsibility to ensure they are the right size and that they have been declared. We return the actual value we have calculated We may need program logic to handle actual errors later res <- a local variable to hold the result as we calculate it

12.17 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[ ])

12.17.1 Macro Definition Documentation

12.17.1.1 ARG_SIZ

```
#define ARG_SIZ 1024
```

12.17.1.2 COM_SIZ

```
#define COM_SIZ 60
```

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

12.17.1.3 RES_SIZ

```
#define RES_SIZ 1024
```

12.17.2 Function Documentation

12.17.2.1 main()

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

This test program calls an existing executable and checks the outputs to standard output meet the expected value It should be called with: test_outputs <filename which contains test definitions> <fp is a pointer 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 properrly formatted

string handling in C can be cumbersome. typically suggetions online can make use fo "malloc" and "stropy" and "strcat" but these complicate things and are argueably 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" we compare it to the expected value 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 nagate (!) the result to test if they are the same

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

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

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

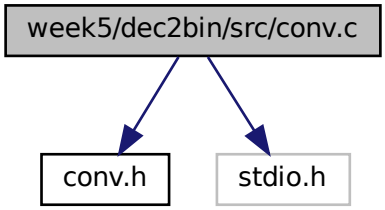
12.18 week4/my_work_log.md File Reference

12.19 week5/dec2bin/bases_and_reports.md File Reference

12.20 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[ ])

12.20.1 Function Documentation

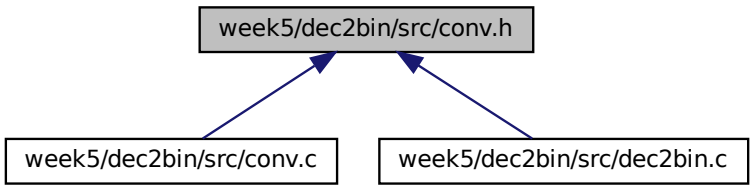
12.20.1.1 dec2r()

```
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 the input string is a valid number can consider other values later...!

12.21 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 out[ ])

12.21.1 Macro Definition Documentation

12.21.1.1 STRLEN

```
#define STRLEN 20
```

12.21.2 Function Documentation

12.21.2.1 dec2r()

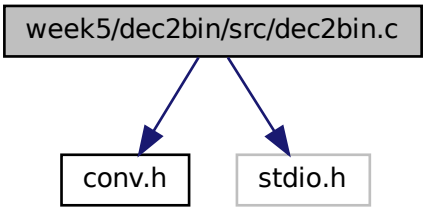
```
void dec2r (
            char in[],
            int r,
            char out[] )
```

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 the input string is a valid number can consider other values later...!

12.22 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[ ])

12.22.1 Function Documentation

12.22.1.1 main()

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

requires a decimal value as the single command line arguement

Index