Testing and Inspection Report

By

Group 2:
Debkanya Mazumder
Sireesha Basamsetty
Onome Ovedje
Diana Carrillo

CS 442

University of Illinois at Chicago

Spring 2016

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. Purpose of the document

This document explains the various Black box and White box testing performed as testing of the MandelBrot application system.

2. Application Overview

The Mandelbrot files contain the following components:

1. Parent Process
2. Child 1 and 2 Processes
3. Pipes & Message queue 1 and 2
4. A shared memory

These components of the program work together to create, display and store Mandelbrot fractal images. Functions of the three major processes is given below:

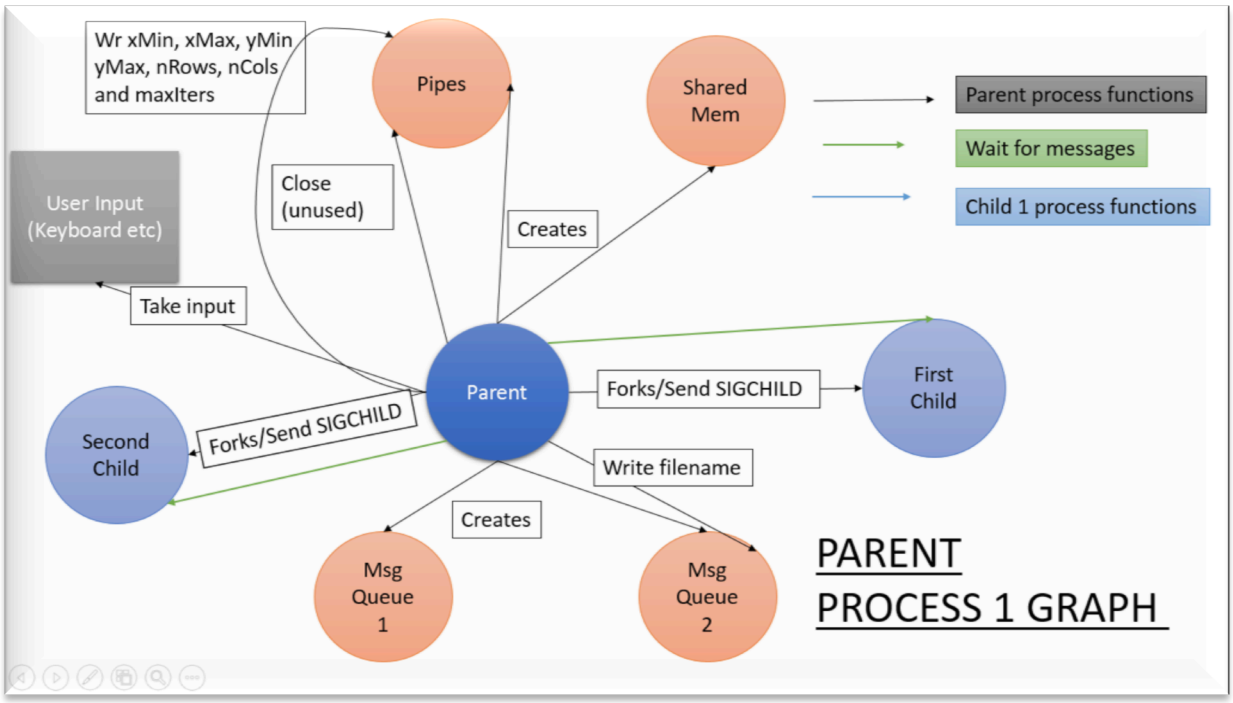
- **Parent process** is responsible for interacting with humans and for creating inter-process communication structure and child processes.
- **First child process** calculates the Mandelbrot calculations and stores in the shared memory
- **Second child** displays the calculations on the screen and saves the data into a file for further processing purposes.

The Software Testing Life Cycle, STLC, consists of multiple testing activities that thoroughly test the software in hand for errors, inconsistencies and other deficiencies. The initial stages of the Software Testing Life Cycle require the gathering of the test cases that should be tested. The test cases can be divided into two separate categories, black box test cases and white box test cases.

3. Graph Implementation of the MandelBrot Application

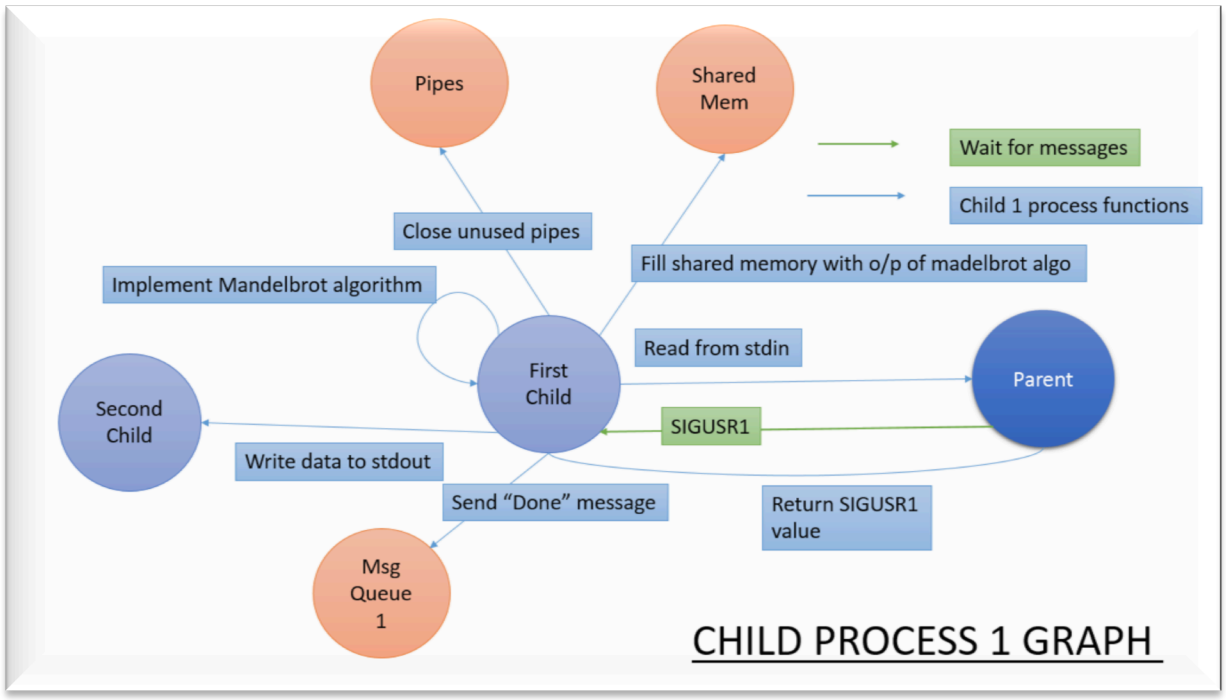
A graph implementation of the MandelBrot application has been done inorder to understand the components of the system and how these components interact with each other. There are three graphs for the parent process and two child processes as shown below.

3(a) Graph for Parent Process



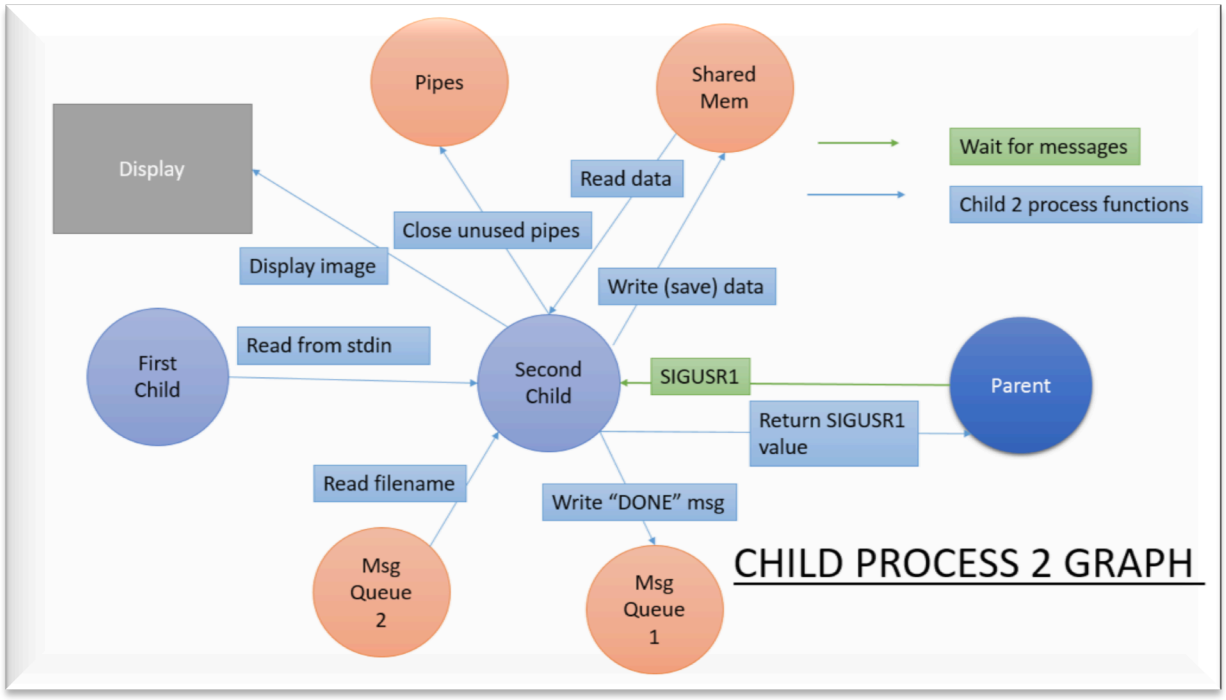
Figure# 1: Graph for Parent Process

3(b) Graph for Child#1 Process



Figure#2: Graph for Child#1 Process

3(c) Graph for Child#2 Process



Figure#3: Graph for Child#2 Process

4. Black box Tests

Black box testing involves testing the components of the MandelBrot files without knowing about the internal functions of the program. Blackbox testing performed on the files are provided in the table below.

Table#1: Blackbox testing

Test ID	File Name	Description	Testing Techniques	Valid Input	Expected Result	Invalid Input	Expected Output
BB-01	mandelbrot-Jbell.cpp	Input: - Valid and Invalid possible File name Inputs are tested against the system.	Techniques used are:- Boundary Value Analysis Error Guessing Equivalence class Partitioning	Valid Input • Abef..z • 0123456 • 023def • XYZS • 1@12 • 1@#$ • Abcdef • ABC012! • Fghw213 • Jen2341! • BCH1@#	Write Filename to message queue 2.	Invalid Input: • !!@#!@ • abc@#@@ • () • ...fg12! • $$#@^ • ^%#@1 • ()*$4d@ • %%#@1 • *&())()() • s@()(*&	Invalid Filename is not accepted by the system.
BB-02	mandelbrot-Jbell.cpp	All possible Inputs for xMin, xMax yMIN yMax to be tested against the system.	Techniques used are:- Boundary Value Analysis Error	Valid Input • -1.9 for xmin, 2 for xmax, -1.5 for ymin, 1.5 for ymax.	The program should be easily able to create Ouput.	Invalid Output: • 5 for xmin, -5 for xmax, -5 for ymin, 5 for ymax. • -5 for	Program will throw an output message saying xmin should be

			Guessing Equivalence class Partitioning	• 1.0 For xmin, 2 for xmax, 0.5 for ymin, 1.5 for ymax. • -2.0 for xmin, -1 for xmax, -1.5 for ymin, -0.5 for ymax.	Program should be easily able to create output. Output will be created by the program.	xmin, 5 for xmax, 5 for ymin, -5 for ymax. • 5 fo xmin, 5 for xmax, -5 for ymin, 5 for ymax. • "xyzabc" for xmin, xmax, ymin, ymax. • Empty line for xmin, xmax, ymin, ymax.	less than xmax, and will ask for input again. Program will throw an output message saying ymin should be less than ymax, and will ask for input again. Program will throw an output message saying xmin should be less than xmax, and will ask for input again. Display Message showing

							that the input should be a number. Display Message showing that the input should be a number.
BB - 03	man delbrot-Jbell.cpp	All possible inputs for nRows, nCols to be tested against the system.	Techniques used are:- Boundary Value Analysis Error Guessing Equivalence class Partitioning	Valid Input: • 50 for nRows, 20 for nCols. • 1 for nRows and 20 for nCols. • 50 for nRows and 1 for nCols.	Output will be created by the program Output will be created by the program Output will be created by the program	Invalid Input: • -10 for nRows. • -10 for nCols. • 0.5 for nRows and nCols.	Output Display Message saying number of rows should be 0 or greater. Output Display Message saying number of rows should be 0 or greater. Output Display Message saying number of rows

							should be 0 or greater.
BB-04	mandelbrot-Jbell.cpp	All possible inputs for maxIter to be tested against the system.	Techniques used are:- Boundary Value Analysis Error Guessing Equivalence class Partitioning	Valid Input: • 100 for maxIter. • 1 and 10 for maxIter. • 200 for maxIter.	Output should be easily created by the program Output should be easily created by the program Output should be easily created by the program	Invalid Input: • 0 for maxIter. • -10 for maxIter.	Output Message saying that number of Iterations should be greater than 0. Output Message saying that number of Iterations should be greater than 0.
BB-05	mandelbrot-Jbell.cpp	Input Done and wait from children to parent.	Techniques used are:- Boundary Value Analysis Error Guessing Equivale	Valid Input: • Done Message. • Wait Message from child to parent.	Program will run correctly.	Invalid Input: • Ok • Complete • Waiting • Running	Program will not run correctly and throw out an error message.

			nce class Partitioning				
BB-06	mandelbrot-Jbell.cpp	Test suites based on program's functional requirements of program responding to SIGCHLD signal.	Techniques used are:- Boundary Value Analysis Error Guessing Equivalence class Partitioning	Valid Input: • 40 for nRows. • 10 for nCols.	Graph contained by file should be 40 by 10.	Invalid Input: • 50 for nRows. • 20 for nCols.	Graph contained by the file should not be 50 by 20 graph.
BB-07	mandelCalc-Jbell.cpp	Test suites based on program's functional requirements of program responding to SIGCHLD signal.	Techniques used are:- Boundary Value Analysis Error Guessing Equivalence class Partitioning	Valid Input: 1. Not killing mandelCalc process through console after starting the program.	Program should not quit, with an error when the program handles SIGCHILD signal.	Invalid Input: Issuing kill mandelCalc process through console after starting the program.	Program should quit, with an error when the program handles SIGCHILD signal.
BB-08	Mandeldisplay-Jbell.c	Test suites based on program's functional requirements of program responding to	Techniques used are:- Error Guessing	Valid Input: 3. Not killing mandelCalc process through console after	Program should not quit, with an error when the program	Invalid Input: Issuing kill mandelCalc process through console after starting the program.	Program should quit, with an error when the program handles

		SIGCHLD signal.	Equivalence class Partitioning	starting the program.	handles SIGCHILD signal.		SIGCHILD signal.
BB-09	Mandeldisplay-Jbell.cpp	Reading valid data from shared memory and saving data to file.	Techniques used are:- Error Guessing Equivalence class Partitioning	Valid Input: 5. Imagedata	Image as output.	Invalid Input: Non imagedata.	Error is thrown and image is not displayed in the output.
BB-10	Mandeldisplay-Jbell.cpp	SIGUSR1 signal from parent is the input.	Techniques used are:- Error Guessing Equivalence class Partitioning	Valid Input: SIGUSR1	Calculated number of Images.	Invalid Input: Other signal from parent.	Exit code is executed and program is halted throwing exit status.
BB-11	MandelCalc-Jbell.cpp	User enters exactly 3 input arguments from command line (mandelCalc file)	Techniques used are:- Error Guessing Equivalence class Partitioning	Valid Input: shmid, msgqid, filename	Program does not crash and next line in the program is executed	Invalid Input: More input arguments or less input arguments.	Program executes the next line
BB-12	Mandlebrot-	User enters values of xMin and	Techniques used are:-	Valid Input: Values of xMin and	Error message to be	Invalid Input: xMin = 2 & xMax = -2	Displays error message

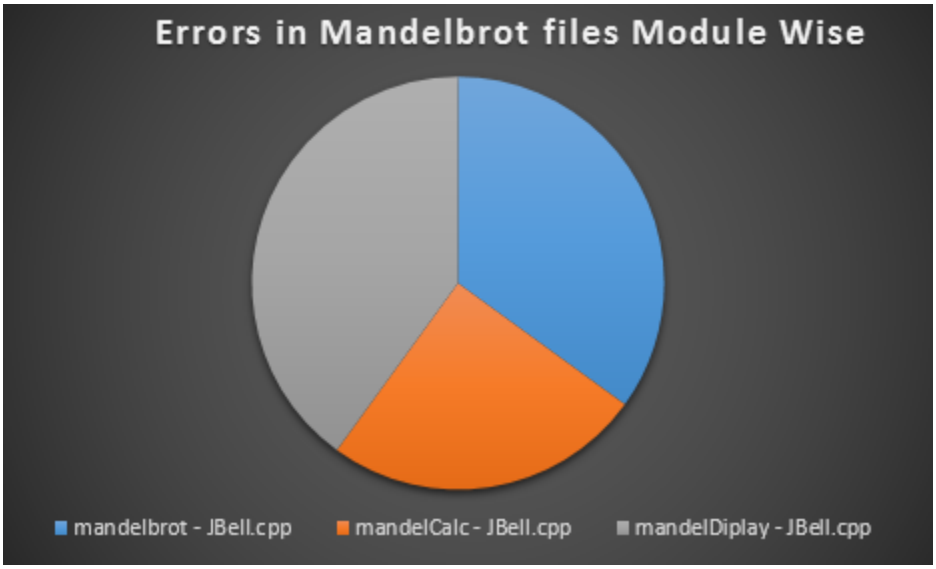
	Jbell.cpp	xMax	Error Guessing Equivalence class Partitioning	xMax are -2 and 2 as xMax > xMin.	displayed "Value of deltaX is negative"		and program exits
BB-13	Mandlebrot-Jbell.cpp	User enters values of yMin and yMax	Techniques used are:- Error Guessing Equivalence class Partitioning	Valid Input: yMin = -2 & yMax = 2	Program should execute normally	Invalid Input: yMin = 2 and yMax = -2	The program executes normally
BB-14	MandelDisplay-Jbell.cpp	User enters value of msgqid	Techniques used are:- Error Guessing Equivalence class Partitioning	Valid Input: msgqid > 1	Error message to be displayed "Value of msgqid is negative"	Invalid Input: msgqid <= 1	Displays error message and program exits
BB-15	MandelDisplay-Jbell.cpp	User enters exactly 4 input arguments from command line	Techniques used are:- Error Guessing Equivalence class Partitioni	Valid Input: Input arguments are shmid, msgqid1, msgqid2, filename	Program should execute normally	Invalid Input: User enters less than or more than 4 input arguments.	The program executes normally

			ng				
BB-16	ManDelDisplay-Jbell.cpp	User enters argument 1 i.e. shmid value as less than or equal to 0	Techniques used are:- Error Guessing Equivalence class Partitioning	Valid Input: argv[1] > 0	Error message to be displayed "Invalid shared memory ID in mandelDisplay"	Invalid Input: argv[1] = -1	Displays error message and program exits
BB-17	ManDelDisplay-Jbell.cpp	User enters argument 2 i.e. msgqid1 less than or equal to 0	Techniques used are:- Error Guessing Equivalence class Partitioning	Valid Input: argv[2] > 0	Error message to be displayed "Invalid shared memory ID in mandelDisplay"	Invalid Input: argv[2] = -1	Displays error message and program exits
BB-18	ManDelDisplay-Jbell.cpp	User enters argument 3 i.e. msgqid2 less than or equal to 0	Techniques used are:- Error Guessing Equivalence class Partitioning	Valid Input: argv[3] > 0	Error message to be displayed **"Invalid message queue ID in mandel Display"**	Invalid Input: argv[3] = -1	Displays error message and program exits
BB-19	ManDelDispCalc-Jbell.	User enters argument 2 i.e. msgqid1 less than or equal to 0	Techniques used are:- Error	Valid Input: argv[2] > 0	Error message to be displayed	Invalid Input: argv[2] = -1	Displays error message and program

	cpp		Guessing Equivalence class Partitioning		**"Invalid message queue ID in mandel Calc"**		exits

5. Code Errors/Flaws:

A distribution of code errors and flaw detected in the Mandelbrot files (module wise) has been indicated by a pie chart given below. The code errors and flaws have been described in a greater detail in the table provided.



Figure#4: A 2D pie chart of distribution of errors in all the Mandelbrot files

Table#2: Errors/Flaws in the code

Flaw ID	File Name	Line No.	Code as written in file	Description
1	mandelbrot - JBell.cpp	166	If(nRows <=0)	Since the user has to press 0 to exit, this should be *if(nRows == 0) {break;}*

2	mandelbrot - JBell.cpp	204	if(!(outfile = fdopen(pipe1[WRITE], "w")))	Fdopen function should have a character array of filename in the first argument. Pipe1 is an int type.
3	mandelbrot - JBell.cpp	244	else if(WIFSIGNALED(status)) cout << "mandelCalc terminated by not catching signal " <<WTERMSIG(status) << endl;	WIFSIGNAL returns true if child process was terminated by a signal. And WTERMSIG returns the number of the signal that caused the child process to terminate. The print statement states that mandelCalc states that the termination occurs by not catching a signal, which is inconsistent.
4	mandelbrot - JBell.cpp	255	else if(WIFSIGNALED(status)) cout << "mandelDisplay terminated by not catching signal " << WTERMSIG(status) << endl << endl;	WIFSIGNAL returns true if child process was terminated by a signal. And WTERMSIG returns the number of the signal that caused the child process to terminate. The print statement states that mandelCalc states that the termination occurs by not catching a signal, which is inconsistent.
5	mandelCalc - JBell.cpp	88,89	deltaX = (xMax - xMin) / (nCols - 1); 89: deltaY = (yMax - yMin) / (nRows - 1);	nCol and nRow are integers, while deltaX and deltaY are double. Requires typecasting.
6	mandelCalc - JBell.	88,89	deltaX = (xMax - xMin) / (nCols - 1); 89: deltaY = (yMax - yMin) / (nRows - 1);	Check if nCols-1 and nRow-1 is not equal to zero to prevent any kind of error before the

	cpp			division is performed.
7	mandelCalc-JBell.cpp	30	int main(int argc, char ** argv, char ** envp) {	Char **envp is an argument which is not used anywhere in the main function.
8	mandelDisplay-JBell.cpp	28	cout << setw(nCols - 12) << right << xMax << endl << endl;	Check if setw(nCols – 12) equals zero or not.
9	mandelCalc-JBell.cpp	-	-	No “Done” message is sent to message queue 1 as mentioned in the functional document.
10	mandelDisplay-JBell.cpp	-	-	No “Done” message is sent to message queue 1 as mentioned in the functional document.
11	mandelBrot-JBell.cpp	164-190	-	Validation of xMin, xMax, yMin, yMax, maxIter and fileSring is missing.
12	mandelBrot-JBell.cpp	41	-	Validation of msgLength is missing.
13	mandelDisplya-JBell.	42	-	Validation of msgLength is missing.

	cpp			
14	mandelDisplya-JBell.cpp	86	-	Validation of xMin, xMax, yMin, yMax, nRows, nCols is missing.
15	mandelDisplya-JBell.cpp	38	-	maxRowsCols is defined but not used anywhere in the program.
16	mandelDisplay-JBell.cpp	36	-	Result is defined but not used anywhere in the program
17	mandelCalc_Jbell.cpp	33	-	int I is defined but not used anywhere in the program.
18	mandelBrot-Jbell.cpp	157	-	Close(pipe1[WRITE]) has not been done. This means that unused pipe1[WRITE] has not been closed.
19	mandelDisplya-JBell.cpp	110,112	-	Align yMax and yMin along the left edge by adding *left* keyword before setw.

20	mandelCalc-JBell.cpp	88,89	deltaX = (xMax - xMin) / (nCols - 1);
			89:
			deltaY = (yMax - yMin) / (nRows - 1);
				No after code condition to check the validity of deltaX and deltaY.

6. Code Review Checklist

6.A Mandelbrot-JBell.cpp

Table#3: CODE REVIEW CHECKLIST: MODULES (mandelbrot-JBell.cpp)

CATEGORY	ITEM	PRESENT? Yes, No, N/A	Comment
Module header	The module must have a module header block containing:		
	File name: mandelbrot-JBell.cpp	Yes	
	Original creator: Prof. John Bell	Yes	
	Date created: Spring 2009	Yes	
	Person who last changed code (if different from creator)	N/A	
	Code revision number and change history (with dates). NOTE: The name of each changed unit should be listed NOTE: If a change is made to correct a defect, the number or ID of the defect corrected should be entered as well.	No	
	High level description: (explain the module's purpose, and the name/purpose	Yes	

	of key data structures, variables, sub-functions used, etc.)		
	Failure modes and effects analysis: List types of failures which could occur in this module and result in a hazard to the patient. List the types of mitigation actions the software takes to prevent hazards from occurring. If these risks are documented in a separate document, reference it. (Editor's note: for non-medical projects, the corollary would simply be an appropriate analysis of possible software failure modes, and what the software should do in each case.)	No	
Module definitions and declarations	**Grouping:** Definitions and declarations should be separated into distinct groups, each with a comment header. For example, #defines, #includes, constant definitions, local function prototypes, etc. would all be grouped separately. If required for greater logical clarity, however, related definitions and declarations may be mixed	Yes	
	Commenting: Each definition or declaration should have an associated descriptive comment unless the declaration is really obvious.	Yes	

Table#4: CODE REVIEW CHECKLIST: UNITS (mandelbrot-JBell.cpp)

CATEGORY	ITEM	Present? Yes, No, N/A	Comments
Code Checks			
	• Descriptive comments are accurate and informative.	Yes	
	• Return values (in particular error returns) are not ignored.	Yes	
	• Constants and literals are not hard coded.	Yes	
	• All variables used have obvious or descriptive names, and correct scope.	Yes	
	• Local functions and non-automatic variables are declared static.	N/A	
	• System global functions have the module name as a prefix to the unit name.	N/A	
	• All functions have prototypes (compiler checks this).	N/A	
	• Data structure fields are described and commented clearly.	No	

	• Code is logically correct (Code performs intended functions, operates correctly)	Yes	
	• Numerical methods are sufficient	Yes	
	• Accuracy of control outputs to external devices are within tolerance	Yes	
	• System I/O mechanisms are consistently used.	Yes	
	• Standard module communication techniques are used (e.g. use of message system)	Yes	
	• Errors are detected and handled, and processing continued	Yes	
	• Error handling conventions are followed (standard use of error handling task, etc.)	Yes	
	• Input values (or other data used) are checked for reasonableness before use	No	No checking for variables like xMin, xMax, yMin, yMax, nRows, nCols, fileString etc.
	• Where necessary, critical	No	Needs

	output parameters or data are checked for reasonableness during processing		checking
	• Code pays attention to recovery from potential hardware faults (e.g. arithmetic faults, power failure, and clock).	N/A	
	• Code pays attention to recovery from device errors.	N/A	
	• There is no redundant code.	Yes	
	• The structure is clean and indentations correct.	Yes	
	• Over complication is avoided.	Yes	
SDS Check	SDS (Software Design Specification) info for this unit is accurate	Yes	

6.B MandelCalc-JBell.cpp

Table#5: CODE REVIEW CHECKLIST: MODULES (mandelCalc-JBell.cpp)

CATEGORY	ITEM	PRESENT? Yes, No, N/A	Comments?
Module header	The module must have a module header block containing:		
	File name: mandelCalc-JBell.cpp	Yes	
	Original creator: Prof. John Bell	Yes	
	Date created: Spring 2014	Yes	
	Person who last changed code (if different from creator)	N/A	
	Code revision number and change history (with dates). NOTE: The name of each changed unit should be listed NOTE: If a change is made to correct a defect, the number or ID of the defect corrected should be entered as well.	N/A	
	High level description: (explain the module's purpose, and the name/purpose of key data structures, variables, sub-functions used, etc.)	No.	Description is needed for this file.

	Failure modes and effects analysis: List types of failures which could occur in this module and result in a hazard to the patient. List the types of mitigation actions the software takes to prevent hazards from occurring. If these risks are documented in a separate document, reference it. (Editor's note: for non-medical projects, the corollary would simply be an appropriate analysis of possible software failure modes, and what the software should do in each case.)	No.	
Module definitions and declarations	**Grouping:** Definitions and declarations should be separated into distinct groups, each with a comment header. For example, #defines, #includes, constant definitions, local function prototypes, etc. would all be grouped separately. If required for greater logical clarity, however, related definitions and declarations may be mixed	Yes	
	Commenting: Each definition or declaration should have an associated descriptive comment	Yes	

	unless the declaration is really obvious.		

Table#6: CODE REVIEW CHECKLIST: UNITS (mandelCalc-JBell.cpp)

CATEGORY	ITEM	PRESENT? Yes, No, N/A	Comments
Code Checks			
	• Descriptive comments are accurate and informative.	Yes	
	• Return values (in particular error returns) are not ignored.	Yes	
	• Constants and literals are not hard coded.	Yes	
	• All variables used have obvious or descriptive names, and correct scope.	No	Role of variable i in line 33 (for instance) is not clear.
	• Local functions and non-automatic variables are declared static.	N/A	

	• System global functions have the module name as a prefix to the unit name.	N/A	
	• All functions have prototypes (compiler checks this).	No	
	• Data structure fields are described and commented clearly.	Yes	
	• Code is logically correct (Code performs intended functions, operates correctly)	Yes	
	• Numerical methods are sufficient	Yes	
	• Accuracy of control outputs to external devices are within tolerance	Yes	
	• System I/O mechanisms are consistently used.	Yes	
	• Standard module communication techniques are used (e.g. use of message system)	Yes	
	• Errors are detected and handled, and processing continued	Yes	
	• Error handling conventions are followed (standard use of error handling task, etc.)	Yes	

	• Input values (or other data used) are checked for reasonableness before use	No	Some inputs need checking.
	• Where necessary, critical output parameters or data are checked for reasonableness during processing	No	Needs validation
	• Code pays attention to recovery from potential hardware faults (e.g. arithmetic faults, power failure, and clock).	N/A	
	• Code pays attention to recovery from device errors.	N/A	
	• There is no redundant code.	Yes	
	• The structure is clean and indentations correct.	Yes	
	• Over complication is avoided.	Yes	
SDS Check	SDS (Software Design Specification) info for this unit is accurate	Yes	

6.C MandelDisplay-JBell.cpp

Table#7: CODE REVIEW CHECKLIST: MODULES (mandelDisplay-JBell.cpp)

CATEGORY	ITEM	PRESENT? Yes, No, N/A	Comments
Module header	The module must have a module header block containing:		
	File name: mandelDisplay-JBell.cpp	Yes	
	Original creator: Prof. John Bell	Yes	
	Date created: Spring 2014	Yes	
	Person who last changed code (if different from creator)	N/A	
	Code revision number and change history (with dates). NOTE: The name of each changed unit should be listed NOTE: If a change is made to correct a defect, the number or ID of the defect corrected should be entered as well.	N/A	

	High level description: (explain the module's purpose, and the name/purpose of key data structures, variables, sub-functions used, etc.)	No	Needs a description
	Failure modes and effects analysis: List types of failures which could occur in this module and result in a hazard to the patient. List the types of mitigation actions the software takes to prevent hazards from occurring. If these risks are documented in a separate document, reference it. (Editor's note: for non-medical projects, the corollary would simply be an appropriate analysis of possible software failure modes, and what the software should do in each case.)	N/A	-
Module definitions and declarations	**Grouping:** Definitions and declarations should be separated into distinct groups, each with a comment header. For example, #defines, #includes, constant definitions, local function prototypes, etc. would all be grouped separately. If required for greater logical	Yes	

	clarity, however, related definitions and declarations may be mixed		
	Commenting: Each definition or declaration should have an associated descriptive comment unless the declaration is really obvious.	Yes	

Table#8: CODE REVIEW CHECKLIST: UNITS (mandelDisplay-JBell.cpp)

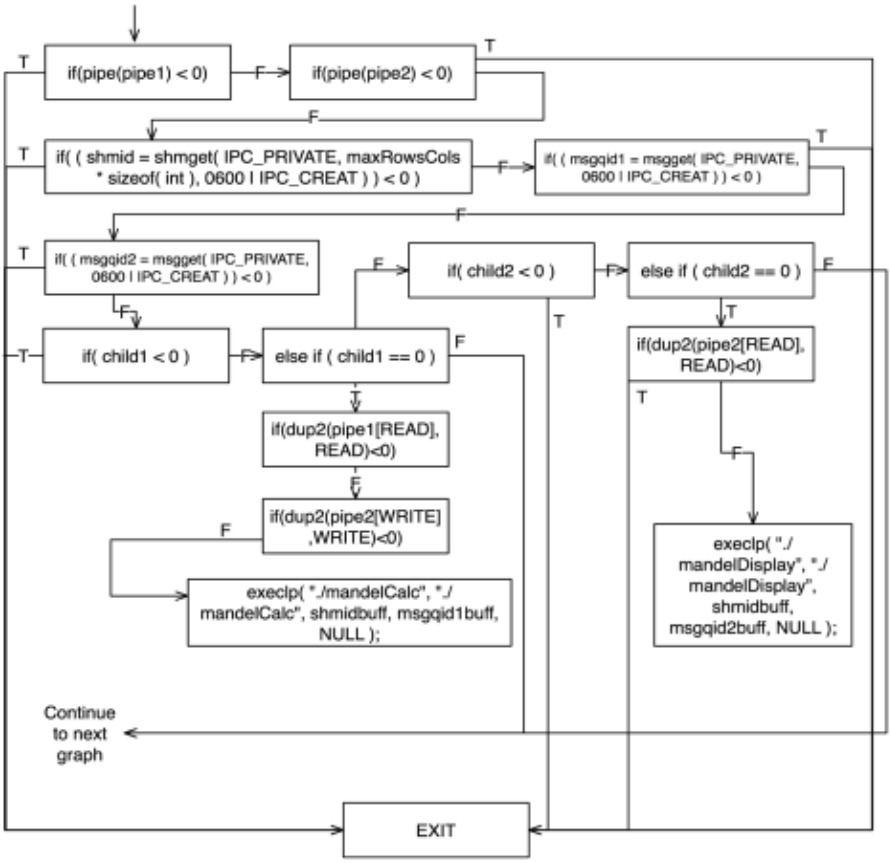
CATEGORY	**ITEM**	**PRESENT? Yes, No, N/A**	**Comments**
Code Checks			
	• Descriptive comments are accurate and informative.	Yes	After reading the document only.
	• Return values (in particular error returns) are not ignored.	Yes	
	• Constants and literals are not hard coded.	Yes	
	• All variables used have obvious or descriptive names, and correct scope.	Yes	

	• Local functions and non-automatic variables are declared static.	No	
	• System global functions have the module name as a prefix to the unit name.	N/A	
	• All functions have prototypes (compiler checks this).	Yes	
	• Data structure fields are described and commented clearly.	No	Needs more description
	• Code is logically correct (Code performs intended functions, operates correctly)	Yes	
	• Numerical methods are sufficient	Yes	
	• Accuracy of control outputs to external devices are within tolerance	Yes	
	• System I/O mechanisms are consistently used.	Yes	
	• Standard module communication techniques are used (e.g. use of message system)	Yes	
	• Errors are detected and handled, and processing continued	Yes	

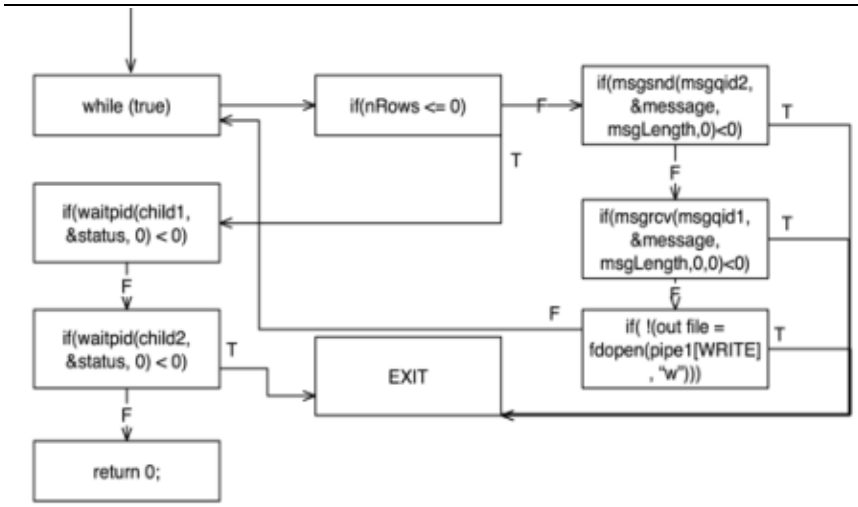
	• Error handling conventions are followed (standard use of error handling task, etc.)	Yes	
	• Input values (or other data used) are checked for reasonableness before use	No	Needs validation
	• Where necessary, critical output parameters or data are checked for reasonableness during processing	No	Needs validation
	• Code pays attention to recovery from potential hardware faults (e.g. arithmetic faults, power failure, and clock).	N/A	
	• Code pays attention to recovery from device errors.	N/A	
	• There is no redundant code.	Yes	
	• The structure is clean and indentations correct.	Yes	
	• Over complication is avoided.	Yes	
SDS Check	SDS (Software Design Specification) info for this unit is accurate	Yes	

7. White box Tests

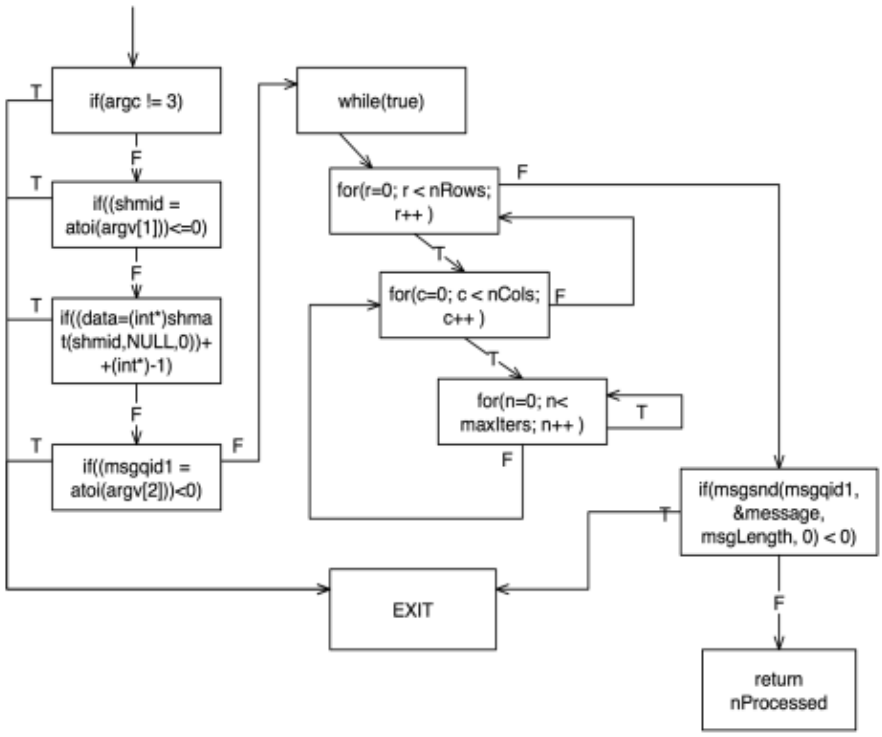
Control Flow Graphs



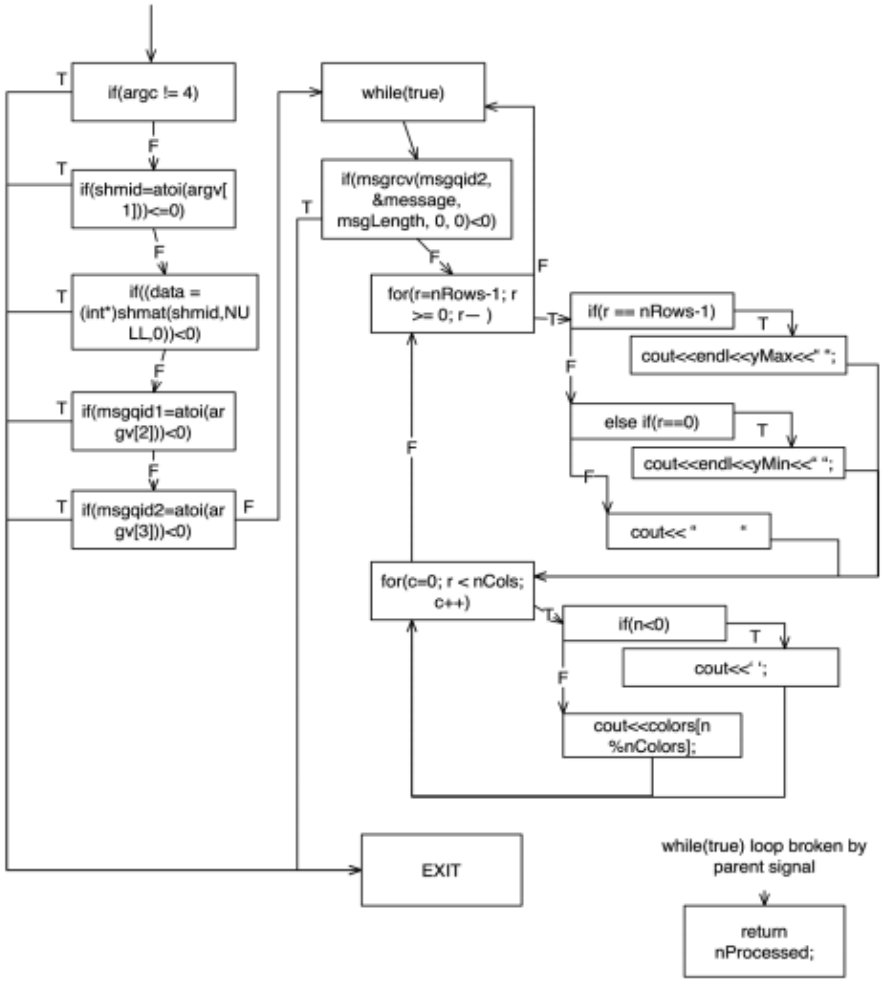
Figure#5: CFG for mandelbrot-JBell.cpp



Figure#6: CFG for mandelbrot-JBell.cpp



Figure#7: CFG for mandelCalc-JBell.cpp



Figure#8: CFG for mandelDisplay-JBell.cpp

7.A- Dirty Tests

Dirty tests are written by examining the code and finding possible lines or pieces of code that can lead to errors under certain circumstances. These tests were written for the purpose of testing those lines of code to see if the expected error will occur.

1. Divide by Zero tests

Table#9: Dirty tests – divide by zero tests

Name	Input	Correct Result	Explanation
DZ-0	1 for nRows, good input otherwise	A graph that covers 1 row showing the Mandelbrot pattern.	Line 88 of Mandelcalc has a possible divide by zero error.
DZ-1	1 for nCols, good input otherwise	A graph that covers 1 column showing the Mandelbrot pattern.	Line 89 of Mandelcalc has a possible divide by zero error.
DZ-2	1 for nRows and 1 for nCols, good input otherwise	A graph that covers 1 column and 1 row showing the Mandelbrot pattern.	Check for both of the above.

2. File name tests

Table#10: Dirty tests – Filename tests

Name	Input	Correct Result	Explanation
FN-0	80 character name for filename, good input otherwise	A graph showing the Mandelbrot calculation and the file containing the numbers of the calculation	Line 195 of Mandelbrot_jbell.cpp has an 80 character limit for the filename. It should be able to accommodate an 80 character filename
FN-1	81 character name for	A graph showing the Mandelbrot	81 characters could/should break the program.

	filename, good input otherwise	calculation and the file containing the numbers of the calculation	
FN-2	File name "./out.txt"	The program shouldn't accept this filename, since Unix/Linux will not accept using "/" in a file.	Line 101 in mandeldisplay writes to the file without previously checking for any incorrect characters, this test will check if the correct input will indeed break the program as a result.

3. Setw tests

Table#11: Dirty tests – SETW tests

Name	Input	Correct Results	Explanation
SW-0	12 for numcolumns	A correct graph and fileoutput.	Line 128 of mandeldisplay has a line that contains "setw(ncols-12)", check the bordercase to see if setw can take 0.
SW-1	11 for numcolumns	A correct graph and fileoutput.	Check line 128 for setw(-1).

Section B : Statement Testing

1. MandelCalc :

Table#12: Statement testing

Name	Input	Correct Result	Explanation
ST-1	Line 52: for argc = 5	Error : Wrong number of args in MandelCalc should be displayed	If argc not equal to 3 , error should be displayed
ST-2	Line 52: For argc = 3	Program Flow should go to next instruction	If argc not equal to 3 , error should be displayed
ST-3	Line 59: For shmid <= 0	Error : Invalid shared memory ID in mandelCalc should be displayed	If shmid <= 0, error should be displayed
ST-4	Line 59: For shmid > 0	Program Flow should go to next instruction	If shmid <= 0, error should be displayed
ST-5	Line 63: Check if shared memory has valid data	If invalid, Error : Shmat in mandelCalc	To check if shared memory has valid data
ST-6	Line 63: Check if shared memory has valid data	If valid, Program flow should go to next instruction	To check if shared memory has valid data
ST-6	Line 70: If msgid1 is less than 0	Error : Invalid message queue ID in mandelCalc	To check if valid message queue id is present
ST-7	Line 70: If msgid1 is greater than or equal to 0	Program flow should go to next instruction	To check if valid message queue id is present
ST-8	Line 93:	For loop is executed as	To check if the row

	For any given nCols value, the for loop must end after finite executions	many times as n rows value if nRows > 0 and shouldn't run if nRows <= 0	wise calculations go through
ST-9	Line 95: For any given nCols value, the for loop must end after finite executions	For loop is executed as many times as n cols value if nRows > 0 and shouldn't run if nRows <= 0	To check if the column wise calculations go through
ST-10	Line 98 : For any given MaxIters value, the for loop must end after finite no of executions	For loop is executed as many times as MaxIters value if MaxIters > 0 and shouldn't run if MaxIters <= 0	To check if the max Iterations wise calculations go through
ST-11	Line 99 : If Zx * Zx + Zy * Zy >= 4,	calculation should stop	To make sure calculations are in mandelBrot range
ST-11	Line 99: If Zx * Zx + Zy * Zy < 4,	Calculation should continue	To make sure calculations are in mandelBrot range
ST-12	Line 106 : If n >= MaxIters, store -1 in data[r][c]	Data[r][c] should be marked as -1	The last array value is marked as -1
ST-13	Line 106: If n < MaxIters, store - 1 in data[r][c]	Data[r][c] should be equal to no of iteration value	The no of iteration should be saved in data[r][c]
ST-14	Line : 124 If msgsnd value is <0	Error : code 1 should come	To check if message is successfully sent to parent
ST-15	Line : 124	Message should be	To check if message

	If msgsnd value is >=0	successfully sent to parent	is successfully sent to parent

2. MandelDisplay

Table#13 : Statement testing in MandelDisplay – JBell.cpp

Name	Input	Correct Output	Reasoning
ST-16	Line : 54 if arg = 4	Program flow should gone fine	To check if correct no of args are passed to child
ST-17	Line : 54 if arg != 4	Error : Wrong number of args in mandelDisplay should come	To check if correct no of args are passed to child
ST-18	Line 61 : If shmid < 0	Error : Invalid Shared memory ID in mandelDisplay should come	To check if shared memory Id is valid
ST-19	Line 61 : If shmid >= 0	Program flow should go fine	To check if shared memory Id is valid
ST-20	Line 65 : If data < 0	Error : shmat in mandelDisplay should come	To check if data is valid
ST-21	Line 65 : If data >= 0	Program flow should go fine	To check if data is valid
ST-22	Line 72 : If msgid1 < 0	Error : Invalid message queue ID 1 in mandelDisplay	To check if message queue ID 1 is valid
ST-23	Line 72 : If msgid1 >= 0	Program flow should continue	To check if message queue ID 1 is valid
ST-24	Line 77 : If msgid2 < 0	Error : Invalid message queue ID 2 in mandelDisplay	To check if message queue ID 2 is valid

ST-25	Line 77 : If msgid2 >= 0	Program flow should continue	To check if message queue ID 2 is valid
ST-26	Line 94 : If message received is < 0	Error : msgrcv(2) in MandelDisplay	To check if MandelDisplay received valid file name
ST-27	Line 94 : If message received is >= 0	Program flow should continue	To check if MandelDisplay received valid file name
ST-28	Line 108 : For any given value of nRows, for loop should be executed finite no of times	For loop is executed as many times as n rows value if nRows > 0 and shouldn't run if nRows <= 0	To check if row wise calculation part goes through
ST-29	Line 114 : For any given value of nCols, for loop should be executed finite no of times	For loop is executed as many times as n rows value if nCols > 0 and shouldn't run if nCols <= 0	To check if column wise calculation part goes through
ST-30	Line 135 : If message send response is < 0	Error : msgsnd (Code 2) should come	To check if parent has successfully received the message
ST-31	Line 135 : If message sent response >= 0	Program execution should go through	To check if parent has successfully received the message

3. MandelBrot :

Table#14: Statement testing in MandelBrot – JBell.cpp

Name	Input	Correct Output	Reasoning
ST-32	Line : 48 if pipe < 0	Error : Pipe (pipe1) failed	To check if pipe1 is created
ST-33	Line : 48 if pipe >= 0	Program flow should continue	To check if pipe1 is created
ST-34	Line : 53 if pipe < 0	Error : Pipe (pipe2) failed	To check if pipe2 is created
ST-35	Line : 53 if pipe >= 0	Program flow should continue	To check if pipe2 is created
ST-36	Line : 60 if shmid < 0	Error : shmget should come	To check if shared memory is successfully created
ST-37	Line : 60 if shmid >= 0	Program flow should continue	To check if shared memory is successfully created
ST-38	Line : 70 if msgid1 < 0	Error : msgget(queue 1) should come	To check if message queue is successfully created
ST-39	Line : 70 if msgid1 >= 0	Program flow should continue	To check if message queue is successfully created
ST-40	Line : 76 if msgid2 < 0	Error : msgget(queue 2) should come	To check if message queue is successfully created
ST-41	Line : 76 if msgid2 >= 0	Program flow should continue	To check if message queue is successfully created

ST-42	Line : 89 If child1 < 0	Error : Forking Child 1 should come	To check is forking is successful
ST-43	Line : 89 If child1 >= 0	Program flow should continue	To check if forking is successful
ST-44	Line : 99 If dup2 < 0	Error : dup2 (stdin) for mandelCalc should come	To duplicate a pipe to stdin and to close the original
ST-45	Line : 99 If dup2 >= 0	Program flow should continue	To duplicate a pipe to stdin and to close the original
ST-46	Line : 106 If dup2 < 0	Error : dup2 (stdout) for mandelCalc should come	To duplicate a pipe to stdout and to close the original
ST-47	Line : 106 If dup2 >= 0	Program flow should continue	To duplicate a pipe to stdout and to close the original
ST-48	Line : 133 If child2 < 0	Error : Forking Child 2 should come	To check is forking is successful
ST-49	Line : 133 If child2 >= 0	Program flow should continue	To check if forking is successful
ST-50	Line : 142 If dup2 < 0	Error : dup2 (stdin) for mandelCalc should come	To duplicate a pipe to stdin and to close the original
ST-51	Line : 142 If dup2 >= 0	Program flow should continue	To duplicate a pipe to stdin and to close the original
ST-52	Line : 197 If message send < 0	Error : msgsnd(filename) should come	To check if filename has been successfully sent
ST-53	Line 198 If msgsnd >= 0	Program flow should continue	To check if filename has been successfully

			sent
ST-54	Line 204 If writing data to pipe is not successful, < 0	Error : fdopen should come	To check if writing to pipe is successful
ST-55	Line 204 : If writing data to pipe is successful, >= 0	Program flow should continue	To check if writing to pipe is successful
ST-56	Line 215 : If message is not received from child 1 successfully < 0	Error : msgrcv(1) should come	To check if parent received message
ST-57	Line 215 : If message is received from child 1 successfully	Program flow should continue	To check if parent received message

Section C- Condition Tests

1. Mandelcalc-Jbell.cpp

```
Line 52-55: if ( argc !=3) {
Cerr<<"Wrong number of args in mandelCalc.\n";
exit(-3);
}
```

Table#15 : Condition testing in MandelCalc – JBell.cpp

Test_Case	Input	Decision	Reasoning
CC-1	True	False	If argc= 1 then it should execute the error message.
CC-2	False	True	if argc=3 then control moves ahead.

```
Line 59-61: if( ( shmid = atoi( argv[ 1 ] ) ) <= 0 ) {
perror( "Invalid shared memory ID in mandelCalc" );
exit( shmid - 1 );
}
```

Table#16 : Condition testing in MandelCalc – JBell.cpp

CC-3	True	False	If the condition is true that control will exit from program displaying an error message.
CC-4	False	True	Program control will moe ahead with next instruction in the program.

```
Line 63-66: if( ( data = ( int * ) shmat( shmid, NULL, 0 ) ) == ( int * ) -1 ) {
perror( "shmat in mandelCalc" );
exit( -6 );
}
```

Table#17: Condition testing in MandelCalc – JBell.cpp

Test_Case	Condition1	Condition2	Decision	Reasoning
CC-5	True	-	True	Program control will exit displaying an error message
CC-6	-	True	True	Program control will exit displaying an error message

Table#18 : Condition testing in MandelCalc – JBell.cpp

CC-7	-	False	False	Program control will go ahead with next instruction
CC-8	True	True	True	Program control will exit displaying an error message

```
Lines 70-73:   if ( ( msgqid1 = atoi( argv[ 2 ] ) ) < 0 ) {
                 perror( "Invalid message queue ID 1 in mandelCalc" );
                 exit( msgqid1 );
               }
```

Table#19 : Condition testing in MandelCalc – JBell.cpp

CC-9	True	False	True	Program control will go ahead with the next instruction
CC-10	False	True	False	Program control will exit displaying error message.
CC-11	False	False	True	Program control will move ahead with next instruction.
CC-12	True	True	False	Program control will exit condition displaying error message.

```
Lines 106-110:  if ( n >= maxIters ) // store -1 in data[ r ][ c ]
                    *( data + r * nCols + c ) = -1;
                else // store n in data[ r ][ c ]
                    *( data + r * nCols + c ) = n;
                }
```

Table#20 : Condition testing in MandelCalc – JBell.cpp

CC-13	True	True	-1 is stored in data[r][c]
CC-14	False	False	Program control jumps to else and n is stored in data[r][c]

```
Lines 124-127: if ( msgsnd ( msgqid1, &message, msgLength, 0 ) < 0 ) {
                    perror( "msgsnd( code 1 )" );
                    exit( -7 );
    }
```

Table#21 : Condition testing in MandelCalc – JBell.cpp

CC-15	True	-	True	True	Program control will exit raising error
CC-16	True	False	-	False	Program control will move ahead with next instruction
CC-17	False	-	False	False	Program control will move ahead with next instruction.

2. mandelDisplay.cpp

```
Lines 61-64: if( ( shmid = atoi( argv[ 1 ] ) ) <= 0 ) {
    perror( "Invalid shared memory ID in mandelDisplay" );
                exit( shmid - 1 );
                }
```

Table#22 : Condition testing in MandelDisplay – JBell.cpp

CC-18	True	False	False	Program control will move ahead with next instruction
CC-19	False	True	False	Program control will move ahead with next instruction
CC-20	True	True	True	Program control will exit showing an error.

```
Lines 77-80: if( ( msgqid2 = atoi( argv[ 3 ] ) ) < 0 ) {
perror( "Invalid message queue ID 2 in mandelDisplay" );
exit( msgqid2 );
}
```

Table#23 : Condition testing in MandelDisplay – JBell.cpp

CC-21	True	-	True	Program control will exit showing error
CC-22	False	True	True	Program control will exit showing error
CC-23	False	False	True	
CC-24	True	True	True	

```
Lines 94-97: if( msgrcv( msgqid2, &message, msgLength, 0, 0 ) < 0 ) {
perror( "msgrcv( 2 ) in mandelDisplay" );
exit( -8 );
}
```

Table#24 : Condition testing in MandelDisplay – JBell.cpp

CC-25	False	False	True	Program control will move ahead with the next instruction
CC-26	True	True	False	Program control will exit showing error

CC-27	False	-	False	Program control will exit showing error

3. Mandelbrot.cpp

*Lines 60-64: if ((shmid = shmget(IPC_PRIVATE, maxRowsCols * sizeof(int), 0600 | IPC_CREAT)) < 0) {*
perror("shmget");
exit(-5);
}

Table#25 : Condition testing in MandelBrot – JBell.cpp

CC-28	True	-	False	True	Program control will exit displaying error.
CC-29	False	True	-	False	Program control will move ahead with next instruction.
CC-30	True	True	True	False	Program control will exit displaying error.

Section D- Path Tests

Table#26: Path Testing

Test ID	Filename and Line Number	Description of Test	Test Input	Expected Output
PT-01	Mandelbrot-JBell.cpp Line number: 48	Testing when pipe(pipe1) < 0	pipe1 = -5	Pipe(pipe1) failed
PT-02	Mandelbrot-JBell.cpp Line number: 48	Testing when pipe(pipe1) > 0	pipe1 = 5	executed

PT-03	Mandelbrot-JBell.cpp Line number: 48	Testing when pipe(pipe1) = 0. This is the boundry case	pipe1 = 0	Pipe(pipe1) failed
PT-04	Mandelbrot-JBell.cpp Line number: 53	Testing with pipe(pipe2) < 0	Pipe2 = -5	Pipe(pipe2) failed
PT-05	Mandelbrot-JBell.cpp Line number: 53	Testing with pipe(pipe2) > 0	Pipe2 = 5	executed
PT-06	Mandelbrot-JBell.cpp Line number: 53	Testing when pipe(pipe2) = 0. This is the boundry case	Pipe2 = 0	Pipe(pipe2) failed
PT-07	Mandelbrot-JBell.cpp Line number: 60	Testing with shmid < 0	shmid = -5	shmget
PT-08	Mandelbrot-JBell.cpp Line number: 60	Testing with shmid > 0	shmid = 5	Executed. And printed: “shmid”
PT-09	Mandelbrot-JBell.cpp Line number: 60	Testing with shmid = 0. Boundry case	shmid = 0	shmget
PT-10	Mandelbrot-JBell.cpp Line number: 70	Testing with msgqid1 < 0.	msgqid1 = -5	Msgget(queue 1)

PT-11	Mandelbrot-JBell.cpp Line number: 70	Testing with msgqid1 > 0.	msgqid1 = 5	Executed and printed: “msgqid1
PT-12	Mandelbrot-JBell.cpp Line number: 70	Testing with msgqid1 = 0. Boundry case	msgqid1 = 0	Msgget(queue 1)
PT-13	Mandelbrot-JBell.cpp Line number: 76	Testing with msgqid2 < 0.	Msgqid2 = -5	Msgget(queue 2)
PT-14	Mandelbrot-JBell.cpp Line number: 76	Testing with msgqid2 > 0.	Msgqid2 = 5	Executed and printed: “msgqid2
PT-15	Mandelbrot-JBell.cpp Line number: 76	Testing with msgqid2 = 0. Boundry case	msgqid1 = 0	Msgget(queue 2)
PT-16	Mandelbrot-JBell.cpp Line number: 89	Testing with child1 < 0.	Child1 = -5	Forking child 1
PT-17	Mandelbrot-JBell.cpp Line number: 89	Testing with child1 > 0.	Child1 = 5	It is a parent and continues to line 119
PT-18	Mandelbrot-JBell.cpp Line number: 92	Testing with child1 = 0.	Child1 = 0	Went inside of the else if block

PT-19	Mandelbrot-JBell.cpp Line number: 99	Testing with dup2(pipe1[READ], READ) < 0.	dup2(pipe1[READ], READ) = -5	Dup2(stdin) for mandelCalc
PT-20	Mandelbrot-JBell.cpp Line number: 99	Testing with dup2(pipe1[READ], READ) > 0.	dup2(pipe1[READ], READ) = 5	Executed. Close(pipe1[READ])
PT-21	Mandelbrot-JBell.cpp Line number: 106	Testing with dup2(pipe2[READ], READ) < 0.	dup2(pipe2[READ], READ) = -5	Dup2(stdout) for mandelCalc
PT-22	Mandelbrot-JBell.cpp Line number: 106	Testing with dup2(pipe2[READ], READ) > 0.	dup2(pipe2[READ], READ) = 5	Executed. Close(pipe2[WRITE])

Section E- Procedure Call Tests

Table#: Procedure Call Test Cases

Test ID	File Name	Description	Input	Expected output
PC-01	Mandelbrot-JBell.cpp	pipe()- Creates pipe	pipe(pipe1) pipe(pipe2)	Returns the status ID
PC-02	Mandelbrot-JBell.cpp	shmget()- creates shared memory	IPC_PRIVATE, maxRowCols*sizeof(int), 0600 | IPC_CREAT	Returns the address of the shared memory
PC-03	Mandelbrot-JBell.cpp	msgget()- creates message queues	IPC_PRIVATE, 0600 | IPC_CREAT	Returns the message queue ID
PC-04	Mandelbrot-JBell.cpp	fork()- creates the child from the parent	-	Returns child object
PC-	MandelCal-	sigHandler() –	Int sig	exit();

05	JBell.cpp / MandelDisplay-JBell.cpp	Handles the signal		
PC-06	MandelCal-JBell.cpp / MandelDisplay-JBell.cpp	signal() – creates a signal	SIGUSR1,sigHandler	-
PC-07	MandelCal-JBell.cpp / MandelDisplay-JBell.cpp	msgsnd()-sends a message back to the parent	Msgqid, &message, msgLength	Returns the status

8. REFERENCES

- http://www.webopedia.com/TERM/B/Black_Box_Testing.html
- http://www.softwaretestinghelp.com/test-summary-report-template-download-sample/

GROUP WORK:

NAME	TEST CASES
Debkanya Mazumder	BB-01 to BB-10
Sireesha Basamsetty	BB-11 to BB-19