TURTLE GRAPHICS REPORT

Unix and C Programming S2 2018

William Payne
18925997@student.curtin.edu.au

Table of Contents

FUNCTIONS:

File: draw.c:

Function: executeGCommand()

Description:

The draw.c function **'void executeGCommandList(Pen *pen, LinkedList *commands)'** iterates through a LinkedList executing each function pointer stored in the GCommand struct resulting in an ascii picture being printed to the terminal. After each GCommand struct has been executed it is freed (The list node is removed upon calling removeLast()).

Declaration:

```
void executeGCommandList(Pen *pen, LinkedList *commands)
```

Parameters:

- pen ~ This struct holds information pertaining to the GCommand function pointers and the effects.c line() function.
- commands ~ This is the LinkedList containing the GCommand structs.

Return Value:

This function has no return value.

Example:

```
    else if(createPen(&pen) && createList(&commands))
    {
        commands->freeFunc = &freeCommand;
        if(readCommands(argv[1], commands) == 0)
        {
            executeGCommandList(pen, commands);
            setColourDefault();
        }

    }
```

File: file_io.c

Function: readCommands()

Description:

The file_io.c function **'int readCommands(char *fileName, LinkedList *commands)'** Creates GCommand structs using data read in from the specified file name. Each GCommand is insertLast() into the imported LinkedList 'commands'.

Declaration:

```
int readCommands(char *fileName, LinkedList *commands)
```

Parameters:

- fileName ~ This is the String containing name of the file to be opened.
- commands ~ This is the LinkedList for storing the new GCommand structs.

Return Value:

This function returns an integer value that represents one of the following error codes.

Error Codes:

0 = No errors occurred.
1 = File doesn't exist.
2 = Read error occurred.
3 = File failed to close.

Example:

```
if(readCommands(filename, commands) == 0)
{
    //read successful - do stuff
}
else
{
    //read failed – handle it
}
```

Function: formatLog()

Description:

The file_io.c function **'int formatLog(char *line, char *type, Coord *startPos, Coord *endPos)'** formats two coordinates to the specifications of the constant 'LOG_FORMAT'. The formatted String is stored in the imported char* 'line' e.g. "DRAW (14.000, 88.000) - (20.000, 20.000)".

Declaration:

```
int formatLog(char *line, char *type, Coord *startPos, Coord *endPos)
```

Parameters:

- line ~ This is the String for storing the formatted log entry.
- type ~ This is the 4-character String containing the type of GCommand e.g. "DRAW"
- startPos ~ This is a pointer to a Coord that contains the start position.
- endPos ~ This is a pointer to a Coord that contains the end position.

Return Value:

0 ~ Temp value(could be replaced with an error code).

Example:

```
    currPos = pen->position;
    distance = *(double*)(command->data);

    calculatePosition(&newPos, &currPos, distance, pen->angle);

    formatLog(logEntry, MOVE, &currPos, &newPos);
    tlog(logEntry);

    pen->position = newPos;
```

Function: tlog()

Description:

This file_io.c function **'int tlog(char *log)'** compiles log entries into a buffer and flushes it to a file specified by the constant string LOG_FILE. The buffer is flushed when the logCount reaches the constant int BUFFER_SIZE or when the imported string 'log' is NULL. The function contains a statically allocated pointer to a string array however the array itself is allocated on the heap. To prevent memory leaks the tlog() function should be passed NULL when the user is finished writing logs, this prints the remaining logs to the LOG_FILE, frees all malloc'd memory and sets the static pointer to NULL.

Declaration:

```
int tlog( char *log)
```

Parameters:

- log ~ This is the String containing the log information to be printed.

Return Values:

0 ~ Temp return

Example:

```
formatLog(logEntry, MOVE, &currPos, &newPos);
tlog(logEntry);
```

Function: emptyBuffer()

Description:

This file_io.c function **'static int emptyBuffer(char ***logBuffer, int *logCount)'** works in tandem with tlog() by handling the freeing of allocated memory, updating the log counter and passing the String array to the appendStrings() function.

Declaration:

```
static int emptyBuffer(char ***logBuffer, int *logCount)
```

Parameters:

- logBuffer ~ This is the pointer to the logBuffer to be printed and freed.
- logCount ~ This is the pointer to the logCount to be reset to 0.

Return Values:
0 ~ Temp return

Example:
IF the imported log is NULL, empty the contents of the logBuffer by passing to emptyBuffer()

```
if(log == NULL)
{
    /*If logCount is 0, logBuffer will be NULL. No further action is
     needed*/
    if(logCount != 0)
    {
        emptyBuffer(&logBuffer, &logCount);
    }
}
```

If logCount reaches BUFFER_SIZE, empty the buffer by calling emptyBuffer()

```
 /*When the log buffer is filled, empty and free all memory used*/

if(logCount == BUFFER_SIZE)
 {
     emptyBuffer(&logBuffer, &logCount);
 }
```

Function: appendStrings()

Description:

This file_io.c function **'int appendStrings(char *fileName, char **Strings, int stringCount)'** appends the imported strings to the end of the file specified by the import 'fileName'.

Declaration:

```
 int appendStrings(char *fileName, char **strings, int stringCount)
```

Parameters:

- fileName ~ This is the String containing the name of the file to be appended to.
- strings ~ This array of strings to be appended to the end of the file.
- stringCount ~ This is the size of array 'strings'

Return Values:

This function returns an integer value that represents one of the following
error codes.

Error Codes:

0 = No errors.
1 = File doesn't exist.
2 = Write error.
3 = Error closing file.

Example:
Parsing a string array to function emptyBuffer()

```
emptyBuffer(&logBuffer, &logCount);
```

emptyBuffer() calls appendStrings with LOG_FILE as the file name and logBuffer as the String array.

```
static int emptyBuffer(char ***logBuffer, int *logCount)
{
    int ii;
    if(appendStrings(LOG_FILE, *logBuffer, *logCount) == 0)
    {
        //Success
    }
```

File: g_command.c:

Function: createGCommand()

Description:

This g_command.c function '**int createGCommand(char *type, char *data, GCommand **newGCommand)'** Creates a new heap GCommand struct and allocates the heap memory needed for the specified 'type'. The GCommand struct is also assigned a function pointer based on its 'type'.

Types:

ROTATE - Assigned the rotate() function and allocated a double.
MOVE - Assigned the move() function and allocated a double.
DRAW - Assigned the draw() function and allocated a double.
FG - Assigned the colourForeground() function and allocated an int.
BG - Assigned the colourBackGround() function and allocated an int.
PATTERN - Assigned the changePattern() function and allocated a char.

Declaration:

```
 int createGCommand(char *type, char *data, GCommand **newGCommand)
```

Parameters:

- type ~ This is the String containing the type of GCommand to create.
- data ~ This is the String containing the data to be passed into the command.
- newGCommand ~ This is the pointer to the new GCommand struct.

Return Values:

1 ~ Temp value.

Example:

```
newGCommand = NULL;
type = strtok(line, delim);
data = strtok(NULL, delim);

/*If a bad line is encountered terminate reading*/
if(type != NULL && data != NULL && strtok(NULL, delim) == NULL)
{
    if(createGCommand(type, data, &newGCommand) == 0)
    {
        //Success
    }
    else{ /*error*/
```

Function: -rotate()

Description:

This g_command.c function **'int rotate(Pen *pen, GCommand *command)'** Calculates an angle from the data provided by the command struct and either adds or subtracts it from the current angle stored in the imported Pen struct.

Declarations:

```
int rotate( Pen *pen, GCommand *command)
```

Parameters:

- pen ~ This is the pointer to the pen whose angle is being modified.
- command ~ This is the GCommand struct containing the angle value.

Return Values:

0 ~ Temp value.

Example:

Finding correct type an assigning rotate() to a temporary function pointer.

```
if(strcmp(type, ROTATE) == 0)
{
    commandFuncPtr = &rotate;
    allocaterPtr = &allocateDouble;
}
```

Assigning the rotate() function to newGCommand 'executer' field.

```
/*Assigning the command function*/
(*newGCommand)->executer = (VoidFunc)commandFuncPtr;
```

Defencing the CommandFunc ptr and passing in the parameters to rotate().

```
while(commands->count > 0)
{
    command = removeFirst(commands);

    /*Running command*/
     (*((CommandFunc)(command->executer)))(pen, command);

    /*Freeing command*/
    freeCommand(command);
}
```

Function: move()

Description:

This g_command.c function **'int move(Pen *pen, GCommand *command)'** function updates the cursors position by passing the coordinate data from calculatePosition() to pen->position. This function is used as a function pointer in GCommand structs. Each call to this function creates a new log entry that is passed to tlog().

Declarations:

```
int move(Pen *pen, GCommand *command)
```

Parameters:

- pen ~ This is the pointer to the pen whose position is being modified.
- command ~ This is the GCommand struct containing the distance value.

Return Values:

0 ~ Temp value.

Example:

Finding correct type an assigning move() to a temporary function pointer.

```
 else if(strcmp(type, MOVE) == 0)
 {
     commandFuncPtr = &move;
     allocaterPtr = &allocateDouble;
 }
```

Assigning the move() function to newGCommand 'executer' field.

```
 /*Assigning the command function*/
 (*newGCommand)->executer = (VoidFunc)commandFuncPtr;
```

Defencing the CommandFunc ptr and passing in the parameters to move().

```
while(commands->count > 0)
 {
     command = removeFirst(commands);

     /*Running command*/
      (*((CommandFunc)(command->executer)))(pen, command);

     /*Freeing command*/
     freeCommand(command);
 }
```

Function: draw()

Description:

This g_command.c function **'int draw(Pen *pen, GCommand *command)'** function calculates the start and end position for the effects.c line() function to plot the pattern stored in the pen struct in a line across the terminal. The new position is calculated by calculatePosition() function. Each call to this function creates a new log entry that is passed to tlog().

Declaration:

```
int draw(Pen *pen, GCommand *command)
```

Parameters:

- pen ~ This is the pointer to the pen whose position is being modified.
- command ~ This is the GCommand struct containing the distance value.

Return Values:

0 ~ Temp value

Example:

Finding correct type an assigning draw() to a temporary function pointer.

```
 else if(strcmp(type, DRAW) == 0)
 {
     commandFuncPtr = &draw;
     allocaterPtr = &allocateDouble;
 }
```

Assigning the draw() function to newGCommand 'executer' field.

```
 /*Assigning the command function*/
 (*newGCommand)->executer = (VoidFunc)commandFuncPtr;
```

Defencing the CommandFunc ptr and passing in the parameters to draw().

```
while(commands->count > 0)
 {
     command = removeFirst(commands);

     /*Running command*/
      (*((CommandFunc)(command->executer)))(pen, command);

     /*Freeing command*/
     freeCommand(command);
 }
```

Function: colourFg(), colourBg()

Description:

The g_command.c functions **'int colourForeground/colourBackground(Pen *pen, GCommand *command)'** Set the terminal colour foreground/background colour to the colour code stored in the GCommand struct. The colour code is also stored in the Pen struct (although not used) just to keep track of current state of everything pertaining to GCommand and line(). These functions are used as function pointers in the GCommand structs. The terminal colour is changed by calling either setBackgroundColour() or setForegroundColour() in effects.c

Declaration:

```
int colourFG(Pen *pen, GCommand *command)
int colourBG(Pen *pen, GCommand *command)
```

Parameters:

> pen ~ This is the pointer to the pen whose colour code is being modified.

> command ~ This is the GCommand struct containing the new colour escape code.

Return Values:

0 ~ Temp value.

Example:

Finding correct type an assigning colourBG to a temporary function pointer.

```
else if(strcmp(type, BG) == 0)
{
    commandFuncPtr = &colourBG;
    allocaterPtr = &allocateInt;
}
```

Assigning the colourBG() function to newGCommand 'executer' field.

```
/*Assigning the command function*/
(*newGCommand)->executer = (VoidFunc)commandFuncPtr;
```

Defencing the CommandFunc ptr and passing in the parameters to colourBG().

```
while(commands->count > 0)
{
    command = removeFirst(commands);

    /*Running command*/
     (*((CommandFunc)(command->executer)))(pen, command);

    /*Freeing command*/
    freeCommand(command);
}
```

Function: changePattern()

Description:

The g_command.c functions **'int changePattern(Pen *pen, GCommand *command)'** sets the pen field 'pattern' to the char stored in the GCommand struct.

Declaration:

```
int changePattern(Pen *pen, GCommand *command)
```

Parameters:

> pen ~ This is the pointer to the pen whose pattern is being modified.

> command ~ This is the GCommand struct containing the new pattern.

Return Values:

0 ~ Temp value.

Example:

Finding correct type an assigning changePattern to a temporary function pointer.

```
else if(strcmp(type, PATTERN) == 0)
{
    commandFuncPtr = &changePattern;
    allocaterPtr = &allocateChar;
}
```

Assigning the changePattern() function to newGCommand 'executer' field.

```
/*Assigning the command function*/
(*newGCommand)->executer = (VoidFunc)commandFuncPtr;
```

Defencing the CommandFunc ptr and passing in the parameters.

```
while(commands->count > 0)
{
    command = removeFirst(commands);

    /*Running command*/
    (*((CommandFunc)(command->executer)))(pen, command);

    /*Freeing command*/
    freeCommand(command);
}
```

Function: plotter()

Description:

The g_command.c function **'void plotter(void *plotData)'** Prints a single character to the terminal and is used as a call back function the effects.c line() function. The imported void* allows any data to be passed through to the PlotFunc.

Declaration:

```
void plotter(void *plotData)
```

Parameters:

> plotData ~ This is the pointer to the character that will be printed.

Return Values:

0 ~ Temp value.

Example:

Call to line();

```
line(x1, y1, x2, y2, &plotter, (void*)&(pen->pattern));
```

In line();

```
for(i = 0; i <= majorDelta; i++)
{
    /* Move to row y + 1, column x + 1 and plot a point. */
    printf("\033[%d;%dH", y + 1, x + 1);
    (*plotter)(plotData);
```

Function: allocateDouble()

Description:

The g_command.c function **'static int allocateDouble(char *data, GCommand *gCommand)'** allocates enough space on the heap for a single double value and points the structs 'data' field at the memory. The function then parses the information in the string 'data' into that allocated memory. This is used as a call back function when creating a GCommand struct.

Declaration:

```
static int allocateDouble(char *data, GCommand *gCommand)
```

Parameters:

- data ~ This is the String that contains the data for the GCommand.
- gCommand ~ This is the pointer to the new GCommand.

Return Values:

- success ~ This is an error code indicating the success of the function

Error Codes:
0 = No errors.
1 = Invalid data.

Example:
Find the correct allocator function.

```
    if(strcmp(type, ROTATE) == 0)
    {
        commandFuncPtr = &rotate;
        allocaterPtr = &allocateDouble;
    }
```

Malloc the GCommand struct.

```
    if(success == 0)
    {
        /*Allocating heap memory for GCommand struct*/
        *newGCommand = (GCommand*)malloc(sizeof(GCommand));
```

Try to parse the 'data' into the GCommand struct using the allocator function.

```
        /*Attempt to allocate and assign data to GCommand data*/
        if((*allocaterPtr)(data,*newGCommand) == 1)
        {
            /*ERROR - Data was invalid*/
```

Function: allocateInt()

Description:

The g_command.c function **'static int allocateInt(char *data, GCommand *gCommand)'** allocates enough space on the heap for a single integer value and points the structs 'data' field at the memory. The function then parses the information in the string 'data' into that allocated memory. This is used as a call back function when creating a GCommand struct.

Declaration:

```
static int allocateInt(char *data, GCommand *gCommand)
```

Parameters:

- data ~ This is the String that contains the data for the GCommand.
- gCommand ~ This is the pointer to the new GCommand.

Return Values:

- success ~ This is an error code indicating the success of the function

Error Codes:

0 = No errors.
1 = Invalid data.

Example:
Find the correct allocator function.

```
    else if(strcmp(type, BG) == 0)
    {
        commandFuncPtr = &rotate;
        allocaterPtr = &allocateInt;
    }
```

Malloc the GCommand struct.

```
    if(success == 0)
    {
        /*Allocating heap memory for GCommand struct*/
        *newGCommand = (GCommand*)malloc(sizeof(GCommand));
```

Try to parse the 'data' into the GCommand struct using the allocator function.

```
        /*Attempt to allocate and assign data to GCommand data*/
        if((*allocaterPtr)(data,*newGCommand) == 1)
        {
            /*ERROR - Data was invalid*/
```

Function: allocateChar()

Description:

The g_command.c function **'static int allocateChar(char *data, GCommand *gCommand)'** allocates enough space on the heap for a single character value and points the structs 'data' field at the memory. The function then parses the information in the string 'data' into that allocated memory. This is used as a call back function when creating a GCommand struct.

Declaration:

```
static int allocateChar(char *data, GCommand *gCommand)
```

Parameters:

- data ~ This is the String that contains the data for the GCommand.
- gCommand ~ This is the pointer to the new GCommand.

Return Values:

This function returns an error code.

- success ~ This is an error code indicating the success of the function

Error Codes:

0 = No errors.
1 = Invalid data.

Example:
Find the correct allocator function.

```
    else if(strcmp(type, PATTERN) == 0)
    {
        commandFuncPtr = &rotate;
        allocaterPtr = &allocateChar;
    }
```

Malloc the GCommand struct.

```
    if(success == 0)
    {
        /*Allocating heap memory for GCommand struct*/
        *newGCommand = (GCommand*)malloc(sizeof(GCommand));
```

Try to parse the 'data' into the GCommand struct using the allocator function.

```
        /*Attempt to allocate and assign data to GCommand data*/
        if((*allocaterPtr)(data,*newGCommand) == 1)
        {
            /*ERROR - Data was invalid*/
```

Function: freeCommand()

Description:

The g_command.c function **'void freeCommand(void *gCommand)'** Frees heap memory from the 'data' field within the struct then frees the GCommand struct. Note this function imports a void* instead of a GCommand* to make it compatible with the FreeFunc function pointer used by the generic linked list struct.

Declaration:

```
void freeCommand(void *gCommand)
```

Parameters:

- gCommand ~ This is a pointer to the GCommand struct to be freed.

Return Values:

None

Example:

Freeing after the command function is run.

```
/*Running command*/
 (*((CommandFunc)(command->executer)))(pen, command);

/*Freeing command*/
freeCommand(command);
```

Setting list freeFunc pointer to freeCommand.

```
list->freeFunc = &freeCommand;
```

Freeing in a linked list using a function pointer.

```
if(currNode != NULL)
{
    completeFreeRec(currNode->next, freeFunc);
    (*freeFunc)(currNode->value);
    free(currNode);
}
```

File: linked_list.c

Function: createList()

Description:

The linked_list.c function **'int createList(LinkedList **list)'** Allocates heap memory for a new LinkedList struct and initializes all fields to their default states.

Declaration:

```
int createList(LinkedList **list)
```

Parameters:

list(LinkedList**) ~ This is the pointer to the new linked list pointer.

Return Value:

This function returns the int '1' as a place holder for an error code.

Example:
Declaring new LinkedList pointer and passing its memory address to createList.

```
 LinkedList *commands = NULL;
 createList(&commands);
```

Function: insertFirst()

Description:

The linked_list.c function **‘int insertFirst(LinkedList *list, void *value)’** assigns the imported void* to a new ListNode that is then inserted at the front of the imported list. The new list node is allocated heap memory, so it must be freed before the program ends.

Declaration:

```
int insertFirst(LinkedList *list, void *value)
```

Parameters:

- list(LinkedList*) ~ This is the pointer to the list that the value is being added too.
- value(void*) ~ This is a pointer to some data being stored in the list.

Example:

Now the list holds a single node that points to ‘a’.

```
int a = 10;
void* value = (void*)&a;

LinkedList *list = NULL;
createList(&list);

insertFirst(list, value)
```

Function: insertLast()

Description:

The linked_list.c function **‘int insertLast(LinkedList *list, void *value)’** assigns the imported void* to a new ListNode that is then inserted at the end of the imported list. The new list node is allocated heap memory, so it must be freed before the program ends.

Declaration:

```
int insertLast(LinkedList *list, void *value)
```

Parameters:

- list(LinkedList*) ~ This is the pointer to the list that the value is being added too.
- value(void*) ~ This is a pointer to some data being stored in the list.

Example:

```
int a = 10;
void* value = (void*)&a;

LinkedList *list = NULL;
createList(&list);

insertLast(list, value);
```

Function: removeFirst()

Description:

The linked_list.c function **'int removeFirst(LinkedList *list)'** Returns the void pointer stored in the first ListNode then removes the node from the list. All pointers to the node from the LinkedList are removed and the List is relinked. The list node finally freed from the heap and the LinkedList count field is decremented.

Declaration:

```
int removeFirst(LinkedList *list, void *value)
```

Parameters:

- list(LinkedList*) ~ This is the pointer to the list that the value is being added too.

Return Value:

This function returns the pointer that the removed node contained.

- value(void*)

Example:

```
int a = 10;
int b = 15;
int *c = NULL;
void *value = (void*)&a;

LinkedList *list = NULL;
createList(&list);

insertFirst(list, value);
insertFirst(list, (void*)&b);

c = (int*)removeFirst(list, value);
//*c = 15;

c = (int*)removeFirst(list, value);
//*c = 10;
```

Function: removeLast()

Description:

The linked_list.c function **‘int removeLast(LinkedList *list)’** Returns the void pointer stored in the last ListNode then removes the node from the list. All pointers to the node from the LinkedList are removed and the List is relinked. The list node finally freed from the heap and the LinkedList count field is decremented.

Declaration:

```
int removeLast(LinkedList *list, void *value)
```

Parameters:

- list(LinkedList*) ~ This is the pointer to the list that the node is being remove from.

Return Value:

This function returns the pointer that the removed node contained.

- value(void*)

Example:

```
int a = 10;
int b = 15;
int *c = NULL;
void *value = (void*)&a;

LinkedList *list = NULL;
createList(&list);

insertFirst(list, value);
insertFirst(list, (void*)&b);

c = (int*)removeLast(list, value);
//*c = 10;

c = (int*)removeLast(list, value);
//*c = 15;
```

Function: get()

Description:

The linked_list.c function **'void *get(LinkedList *list, int index)'** iterates through the imported link list returning a pointer the value located as the specified index. If the index is not within the bounds of the list the function will return NULL. Note this does not remove the node from the linked list.

Declaration:

```
void *get(LinkedList *list, int index)
```

Parameters:

- list(LinkedList*) ~ This is the linked list the value is being retrieved from.
- index(int) ~ This is the place in the list where the value will be retrieved.

Return Value:

This function returns a void* to the data.

- value(void*)

Example:

```
int a = 10;
int b = 15;
int *c = NULL;
void *value = (void*)&a;

LinkedList *list = NULL;
createList(&list);

insertFirst(list, value);
insertFirst(list, (void*)&b);

c = (int*)get(list, 0);
//*c = 15;

c = (int*)get(list, 0);
//*c = 15;
```

Function: freeList()

Description:

The linked_list.c function **‘int freeList(LinkedList *list)’** Is a wrapper method for the recursive function freeListRec(). This function validates that the list is not NULL then passes the list head to the recursive function. If the list pointer is NULL, it returns an error.

Declaration:

```
int freeList(LinkedList *list)
```

Parameters:

- list(LinkedList*) ~ This is the pointer to a linked that will be freed.

Return Value:

This function returns an error code.

Error Codes:

0 = List freed.
1 = List pointer is NULL.

Example:

If the list not null, free.

```
 if(list != NULL)
 {
     a = freeList(list);
     /*a == 0*/
 }
```

Checking if the list was already freed.

```
if(freeList(list) == 1)
 {
     printf(“List is NULL”);
 }
```

Static Function: freeListRec()

Description:

The linked_list.c function **‘static void freeListRec(ListNode *currNode)’** Recursively goes through a linked list till it reaches the end then as it unwinds each node is freed. This function can stack overflow in list greater than 300,000 nodes large, to avoid this an iterative approach could be used. The function is static enforce use of the wrapper function freeList().

Declaration:

```
static void freeListRec(ListNode *currNode)
```

Parameters:

- currNode(ListNode*) ~ The list node to be freed.

Return Value:

No returns

Example:
freeListRec() call in the freeList() wrapper function:

```
    if(list != NULL)
    {
        freeListRec(list->head);
        free(list);
        success = 0;
    }
```

Function: completeFreeList()

Description:

The linked_list.c function **‘int completeFreeList(LinkedList *list)’** Is a wrapper method for the recursive function completeFreeRec(). This function validates that the list is not NULL then passes the list head to the recursive function. If the list pointer is NULL, it returns an error.

Declaration:

```
int completeFreeList(LinkedList *list)
```

Parameters:

- list(LinkedList*) ~ This is the pointer to a linked that will be freed.

Return Value:

This function returns an error code.

Error Codes:

0 = List freed.
1 = List pointer is NULL.

Example:

If the list not null, free.

```
 if(list != NULL)
 {
     a = completeFreeList(list);
     /*a == 0*/
 }
```

Checking if the list was already freed.

```
if(completeFreeList(list) == 1)
 {
     printf(“List is NULL”);
 }
```

Static Function: freeListRec()

Description:

The linked_list.c function **'static void freeListRec(ListNode *currNode, FreeFunc freeFunc)'** Recursively goes through a linked list till it reaches the end then as it unwinds each node and node value is freed. The value stored in each node is freed using the imported FreeFunc function pointer. This function can stack overflow in list greater than 300,000 nodes large, to avoid this an iterative approach could be used. The function is static enforce use of the wrapper function freeList().

Declaration:

```
static void completeFreeRec(ListNode *currNode, FreeFunc freeFunc)
```

Parameters:

- currNode(ListNode*) ~ The list node to be freed.

Return Value:

No returns

Example:
freeListRec() call in the freeList() wrapper function:

```
    if(list != NULL)
    {
        completeFreeRec(list->head, list->freeFunc);
        free(list);
        success = 0;
    }
```

File: main.c:

Function: main()

Description:

The main.c function **'int main(int argc, char **argv)'** is the main function for the Turtle Graphics program.

Declaration:

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

Parameters:

- argc(int) ~ This is the number of command-line arguments.
- argv(char**) ~ This an array of command-line arguments.

Return Value:

'0' Temp value.

File: pen.c:

Function: createPen()

Description:

The pen.c function **‘int createPen(Pen **pen)’** creates a new Pen struct on the heap and initializes all its fields to their default state. The imported Pen** is dereferenced once and assigned the pointer returned by malloc().

Declaration:

```
int createPen(Pen **pen)
```

Parameters:

- pen(Pen**) ~ This the pointer to the heap Pen struct pointer.

Return Value:

This function returns 1 as temp value.

Example:

```
Pen *pen = NULL;
createPen(&pen);
printf("Angle: %f", pen->angle);      /*”Angle: 0.0”
printf("Fore ground: %d", pen->fg);   /*”Fore ground: 7”
printf("Pattern: %c", pen->pattern);  /*”Pattern: #”
```

Function: freePen()

Description:

The pen.c function **'void freePen(Pen *pen)'** Frees all memory allocated to the Pen struct. Currently the function just calls the C library function free() but the abstraction allows for better maintainability e.g. if the pen struct was to have news fields on that exist on the heap. It also removes doubt from using the struct as whats being alloc'd is hidden when calling createPen().

Declaration:

```
void freePen(Pen *pen)
```

Parameters:

- pen(Pen*) ~ This is a pointer to the Pen struct that will be freed.

Return Value:

No returns.

Example:
Simple example.

```
    if(pen != NULL)
    {
        freePen(pen);
    }
```

File: utils.c:

Function: capitalize()

Description:

The utils.c function **'void capitalize(char *text)'** capitalizes all alphabetic characters in a string ignoring all other symbols. The function overwrites the original String thus a copy should be made if the original String needs to be preserved.

Declaration:

```
void capitalize( char *text)
```

Parameters:

- text(char*) ~ This is the String to be capitalized.

Return Value:

No returns.

Example:

```
 {

    char text[] = "miXEd cAse seNtANCe";
    capitalize(text);
    printf("%s\n", text); /*Result: "MIXED CASE SENTANCE"*/

}
```

```
 {

    char text[] = "%^ignores 123 non-alphabetical”{}characters";
    capitalize(text);
    printf("%s\n", text); /*Result: "%^IGNORES 123 NON-ALPHABETICAL”{}…*/

}
```

Function: round()

Description:

The utils.c function **'int rounds(double num)'** rounds a double value using ceil and floor and returns the value. The function rounds down on < 0.5. The original number is preserved, and the rounded value is returned as an integer.

Declaration:

```
int rounds(double num)
```

Parameters:

- num(double) ~ This is double to be rounded.

Return Value:

This function returns an integer value of the rounded double 'num'.

- rounded(int)

Example:

Rounding up.

```
double a = 1.5;
int b = rounds(a);
printf("%d\n", b); /*Result: "2"*/
```

Rounding down.

```
double a = 1.49;
int b = rounds(a);
printf("%d\n", b); /*Result: "1"*/
```

Function: convertToRadians()

Description:

The utlis.c function **‘void convertToRadians(double *angle)’** converts the imported double value to from degrees to radians. This function overwrites the original value so If it needs to be preserved a copy should be created.

Declaration:

```
void convertToRadians( double *angle)
```

Parameters:

- angle(double*) ~ This is a pointer to the double that will be converted to radians.

Return Value:

This function has no returns.

Example:

This is a snippet from the calculatePosition() in utils.c

```
    convertToRadians(&angle);
    newPos->pos[0] = currPos->pos[0] + distance * cos(angle);
    newPos->pos[1] = currPos->pos[1] + distance * sin(angle);
```

Function: calculatePosition()

Description:

The utils.c function '**void calculatePosition(*newPos, *currPos , distance, angle)'** Calculates the position using a starting position, a distance and an angle writing the new position to the imported newPos Coord struct. This function uses c's inbuilt cos() and sin() functions requiring the imported angle to be converted to radians from degrees.

Declaration:

```
void calculatePosition( Coord *newPos, Coord *currPos
                      , double distance, double angle)
```

Parameters:

- newPos(Coord) ~ This is the new position of the cursor.
- currPos(Coord) ~ This is the current position of the cursor.
- distance(double) ~ This is the distance to move the cursor.
- angle(double) ~ This is the angle to move the cursor on.

Return Value:

This function has no returns.

Example:

This is a modified snippet from the draw() function in g_command.c

```
    int x1, x2, y1, y2;
    Coord currPos = NEW_COORD;
    Coord newPos = NEW_COORD;

    currPos = pen->position;
    distance = *(double*)(command->data);

    calculatePosition(&newPos, &currPos, distance, pen->angle);

    x1 = rounds(currPos.pos[0]);
    y1 = rounds(currPos.pos[1]);
    x2 = rounds(newPos.pos[0]);
    y2 = rounds(newPos.pos[1]);

    line(x1, y1, x2, y2, &plotter, (void*)&(pen->pattern));
```

Function: realign()

Description:

The utils.c function '**realign(Coord *realPos, Coord *currPos)'** Checks to see whether the imported rounded coordinate is further enough out of its alignment with its real position to justify resetting it.

Declaration:

```
void realign( Coord *realPos, Coord *currPos)
```

Parameters:

- realPos(Coord) ~ *The real position a coordinate should be in.*
- currPos(Coord) ~ *This is the current position of the cursor.*

Return Value:

This function has no returns.

INPUT FILE TO COORDINATE SYSTEM:

Implementation:

Reading from the file:

Each line in the file is read in one at time using the c library function fgets() until the EOF is reached. The new line character is removed from each line and then tokenized using a single space as the delimiter. If the input file is correct there should only be 2 tokens, the first be the type of command and the second being the data needed for the command. If the command and data are valid a new GCommand struct containing the command and data is added to a list of commands.

Data to coordinates:

ROTATE, DRAW and MOVE graphic commands are used when calculating the cursors coordinates. Each of these commands store a double value representing either an angle in degrees (ROTATE) or a distance in characters to move (DRAW, MOVE). The Pen struct is used to store the current angle and the current position of the cursor in a Coord struct. The Coord struct holds a double array of size 2 corresponding to a x and y coordinate. When the ROTATE command is called the Pen structs angle field is updated by adding the value stored in the command struct after it is inverted. In order to use the c library function sin() and cos() the stored angle must be converted to radians. When the MOVE command is called the current angle position along with the distance from the GCommand struct is passed to the utils.c function calculatePosition(). The Pen structs position is then updated with new position after it is checked for alignment. The DRAW command works in a similar way but negates 1 from the distance before the coordinates is passed to the effects.c function line() after which the Pen struct is updated using the full distance.

Alternate approach:

Storing of the coordinates:

Instead of using a struct to store the current x and y position, 2 double variables could be used inside of the Pen struct. This would simplify accessing the x and y position individually but increases the number of import statements required when passing the information around. Another possibility would be store the current x and y position with the current angle in a double array of size 3. This would reduce the number import statements required when passing around the coordinate information and possibly reduce the amount of wasted space from poor alignment in the struct.

Calculating the position:

Instead of printing to the terminal after each coordinate is calculated, a list of position could be pre-processed before making any calls to the line() function. This would reduce the amount of calculations required during the printing of the picture which may be useful when viewing an animated turtle graphic file as it may increase the frame rate it can play at.

DEMONSTRATION AND TESTING, RESULTS AND MATERIALS

TurtleGraphics

Command-lines

Standard execution:

```
./TurtleGraphics <filename>
```

Standard execution using make: *(opens default file specified in Makefile)*

```
make run
```

Valgrind -verbose:

```
valgrind -v ./TurtleGraphics <filename>
```

Valgrind -verbose -leak checking:

```
valgrind -v --leak-check=full --show-reachable=yes --track-origin=yes
./TurtleGraphics <filename>
```

Valgrind -verbose using make: *(opens default file specified in Makefile)*

```
 make valError
```

Valgrind -leak checking using make: *(opens default file specified in Makefile)*

```
 make valLeak
```

Valid files:
Each line must start with one of the following arguments:
{ROTATE, MOVE, DRAW, FG, BG, PATTERN}
The argument can be in any case including mixed case e.g. “roTATe 5”
Each line with an argument has exactly one trailing value separated by a space and no values beforehand.
Empty lines are valid and are skipped.

Invalid files:
A line has more than two tokens separated by a space.
A line has only a single token.
The argument given is not one of the 6 described above.
The value after the argument is of an incompatible type.

Valid input 1:

Test file: angles.txt

File contents:

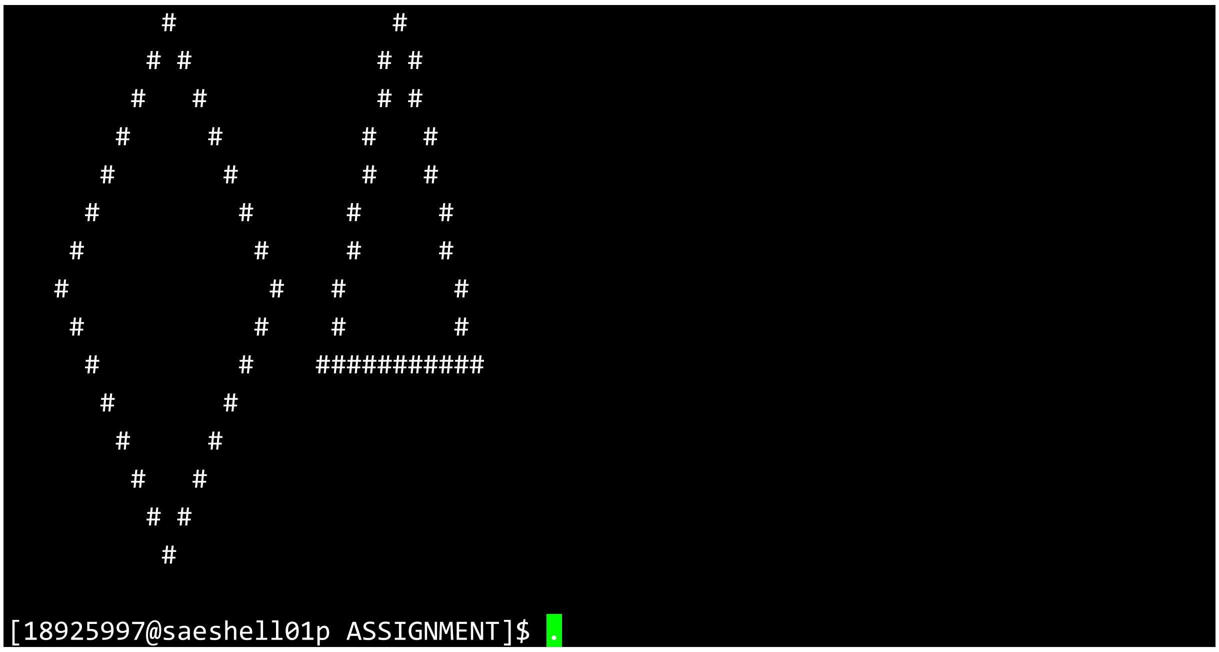
```

move 10
rotate -45
draw 10
rotate -90
draw 10
rotate -90
draw 10
rotate -90
draw 10
```

```
rotate -45
move 15
rotate -60
draw 10
rotate -120
draw
10ssh://%3Cconfig%3E/saeshell02p.sshfs
.jsonc
rotate -120
draw 10
```

Command-line:

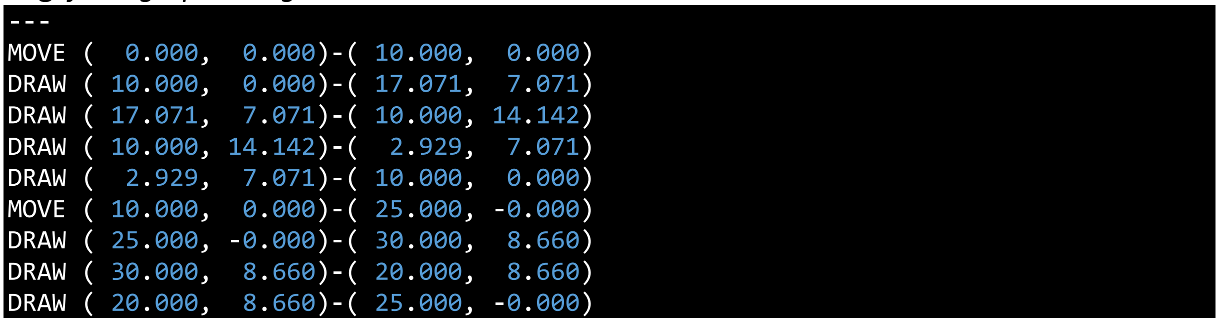
```
[18925997@saeshell02p ASSIGNMENT]$ ./TurtleGraphics angles
```

Terminal Output:



```
          #              #
         # #            # #
        #   #           # #
       #     #         #   #
      #       #        #   #
     #         #      #     #
    #           #     #     #
   #             #   #       #
    #           #    #       #
     #         #    ###########
      #       #
       #     #
        #   #
         # #
          #

[18925997@saeshell01p ASSIGNMENT]$ 
```

Log: *from graphics.log*



```
---
MOVE (  0.000,  0.000)-( 10.000,  0.000)
DRAW ( 10.000,  0.000)-( 17.071,  7.071)
DRAW ( 17.071,  7.071)-( 10.000, 14.142)
DRAW ( 10.000, 14.142)-(  2.929,  7.071)
DRAW (  2.929,  7.071)-( 10.000,  0.000)
MOVE ( 10.000,  0.000)-( 25.000, -0.000)
DRAW ( 25.000, -0.000)-( 30.000,  8.660)
DRAW ( 30.000,  8.660)-( 20.000,  8.660)
DRAW ( 20.000,  8.660)-( 25.000, -0.000)
```

Valid input 2:

Test file: ColoursAndPatterns.txt

File contents:

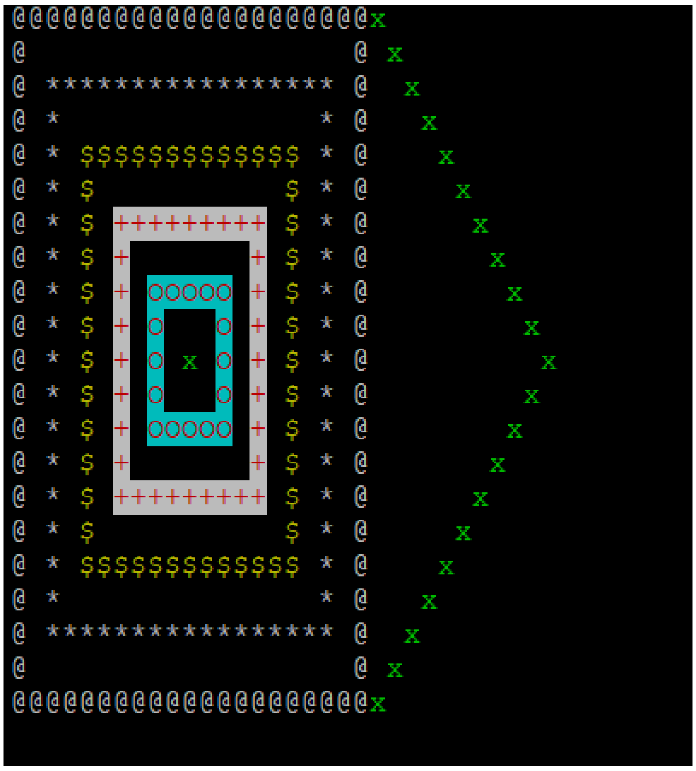
```
rotate -90
move 1
rotate 90
pattern @
draw 20
rotate -90
draw 20
rotate -90
draw 20
rotate -90
draw 20
rotate -90
move 2
rotate -90
move 2
rotate 90
pattern *
draw 16
rotate -90
draw 16
rotate -90
draw 16
rotate -90
draw 16
rotate -90
move 2
rotate -90
move 2
rotate 90
pattern $
BG 0
FG 3
draw 12
rotate -90
draw 12
rotate -90
draw 12
rotate -90
draw 12
rotate -90
move 2
rotate -90
move 2
```

```
rotate 90
BG 7
FG 1
pattern +
draw 8
rotate -90
draw 8
rotate -90
draw 8
rotate -90
draw 8
rotate -90
move 2
rotate -90
move 2
rotate 90
BG 6
pattern O
draw 4
rotate -90
draw 4
rotate -90
draw 4
rotate -90
draw 4
rotate -90
BG 1
FG 6
move 2
rotate -90
move 2
rotate 90
fg 2
bg 0
pattern x
draw 1
move 10
rotate 90
move 10
rotate -135
draw 14
rotate -90
draw 15
```

Command-line:

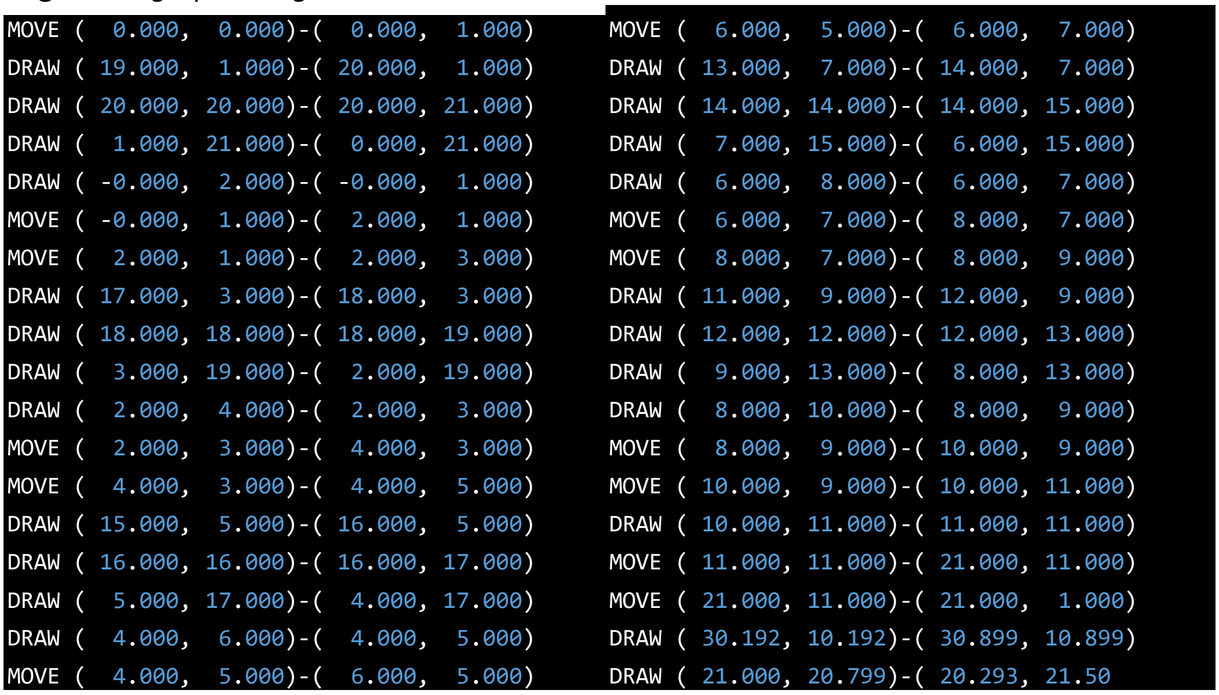
```
[18925997@saeshell02p ASSIGNMENT]$ ./TurtleGraphics ColourAndPatterns.txt
```

Terminal Output:



```
@@@@@@@@@@@@@@@@@@@@@x
@                   @ x
@ ***************** @  x
@ *               * @   x
@ * $$$$$$$$$$$$$ * @    x
@ * $           $ * @     x
@ * $ +++++++++ $ * @      x
@ * $ +       + $ * @       x
@ * $ + OOOOO + $ * @        x
@ * $ + O   O + $ * @         x
@ * $ + O x O + $ * @          x
@ * $ + O   O + $ * @         x
@ * $ + OOOOO + $ * @        x
@ * $ +       + $ * @       x
@ * $ +++++++++ $ * @      x
@ * $           $ * @     x
@ * $$$$$$$$$$$$$ * @    x
@ *               * @   x
@ ***************** @  x
@                   @ x
@@@@@@@@@@@@@@@@@@@@@x
```

Log: *From graphics.log*



```
MOVE (  0.000,   0.000)-(  0.000,   1.000)     MOVE (  6.000,   5.000)-(  6.000,   7.000)
DRAW ( 19.000,   1.000)-( 20.000,   1.000)     DRAW ( 13.000,   7.000)-( 14.000,   7.000)
DRAW ( 20.000,  20.000)-( 20.000,  21.000)     DRAW ( 14.000,  14.000)-( 14.000,  15.000)
DRAW (  1.000,  21.000)-(  0.000,  21.000)     DRAW (  7.000,  15.000)-(  6.000,  15.000)
DRAW ( -0.000,   2.000)-( -0.000,   1.000)     DRAW (  6.000,   8.000)-(  6.000,   7.000)
MOVE ( -0.000,   1.000)-(  2.000,   1.000)     MOVE (  6.000,   7.000)-(  8.000,   7.000)
MOVE (  2.000,   1.000)-(  2.000,   3.000)     MOVE (  8.000,   7.000)-(  8.000,   9.000)
DRAW ( 17.000,   3.000)-( 18.000,   3.000)     DRAW ( 11.000,   9.000)-( 12.000,   9.000)
DRAW ( 18.000,  18.000)-( 18.000,  19.000)     DRAW ( 12.000,  12.000)-( 12.000,  13.000)
DRAW (  3.000,  19.000)-(  2.000,  19.000)     DRAW (  9.000,  13.000)-(  8.000,  13.000)
DRAW (  2.000,   4.000)-(  2.000,   3.000)     DRAW (  8.000,  10.000)-(  8.000,   9.000)
MOVE (  2.000,   3.000)-(  4.000,   3.000)     MOVE (  8.000,   9.000)-( 10.000,   9.000)
MOVE (  4.000,   3.000)-(  4.000,   5.000)     MOVE ( 10.000,   9.000)-( 10.000,  11.000)
DRAW ( 15.000,   5.000)-( 16.000,   5.000)     DRAW ( 10.000,  11.000)-( 11.000,  11.000)
DRAW ( 16.000,  16.000)-( 16.000,  17.000)     MOVE ( 11.000,  11.000)-( 21.000,  11.000)
DRAW (  5.000,  17.000)-(  4.000,  17.000)     MOVE ( 21.000,  11.000)-( 21.000,   1.000)
DRAW (  4.000,   6.000)-(  4.000,   5.000)     DRAW ( 30.192,  10.192)-( 30.899,  10.899)
MOVE (  4.000,   5.000)-(  6.000,   5.000)     DRAW ( 21.000,  20.799)-( 20.293,  21.50
```

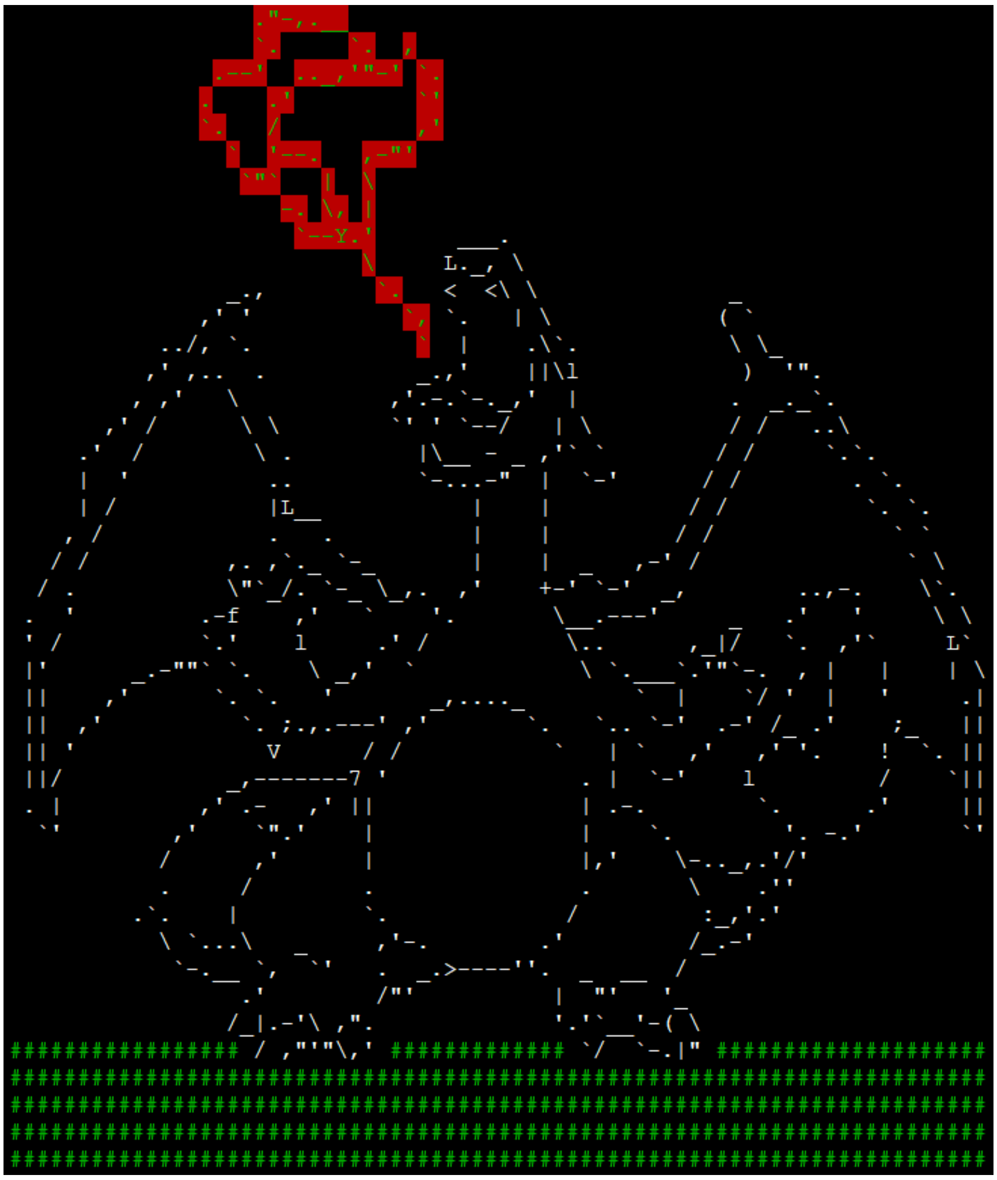
Valid input 3:

Test file: Charizard.txt

File contents: *File is 7000 too many lines long to display.*

```
[18925997@saeshell02p ASSIGNMENT]$ ./TurtleGraphicsSimple Charizard.txt
```

Terminal Output:



Log: *First six and last six log entries from graphics.log*

```
MOVE (  0.000,  0.000)-( 18.000,  0.000) MOVE ( 72.000, 40.000)-( -0.000, 40.000)
DRAW ( 18.000,  0.000)-( 19.000,  0.000) MOVE ( -0.000, 40.000)-( -0.000, 41.000)
DRAW ( 19.000,  0.000)-( 20.000,  0.000) DRAW ( 71.000, 41.000)-( 72.000, 41.000)
DRAW ( 20.000,  0.000)-( 21.000,  0.000) MOVE ( 72.000, 41.000)-( -0.000, 41.000)
DRAW ( 21.000,  0.000)-( 22.000,  0.000) MOVE ( -0.000, 41.000)-( -0.000, 42.000)
DRAW ( 22.000,  0.000)-( 23.000,  0.000) DRAW ( 71.000, 42.000)-( 72.000, 42.000)
```

…

Valid input with Valgrind 1:

Test file: angles.txt

Command-line:

```
[18925997@saeshell02p ASSIGNMENT]$ valgrind -v --leak-check=full --show-
reachable=yes --track-origin=yes ./TurtleGraphics angle.txt
```

Terminal Output:

```
==22413== Memcheck, a memory error detector
==22413== Copyright (C) 2002-2012, and GNU GPL'd, by Julian Seward et al.
==22413== Using Valgrind-3.8.1 and LibVEX; rerun with -h for copyright info
==22413== Command: ./TurtleGraphics angles
==22413==
          #              #
         # #            # #
        #   #           # #
       #     #         #   #
      #       #        #   #
     #         #      #     #
    #           #     #     #
   #             #   #       #
    #           #    #       #
     #         #    ###########
      #       #
       #     #
        #   #
         # #
          #








==22413==
==22413== HEAP SUMMARY:
==22413==     in use at exit: 0 bytes in 0 blocks
==22413==   total heap usage: 66 allocs, 66 frees, 3,207 bytes allocated
==22413==
==22413== All heap blocks were freed -- no leaks are possible
==22413==
==22413== For counts of detected and suppressed errors, rerun with: -v
==22413== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 8 from 6)
```

Valid input with Valgrind 2:

Test file: ColourAndPatterns.txt

Command-line:

```
[18925997@saeshell02p ASSIGNMENT]$ valgrind -v --leak-check=full --show-reachable=yes --track-origin=yes ./TurtleGraphics angle.txt
```

Terminal Output: *Empty space removed between ascii picture and valgrind output. Excluding colour.*

```
==22496== Memcheck, a memory error detector
==22496== Copyright (C) 2002-2012, and GNU GPL'd, by Julian Seward et al.
==22496== Using Valgrind-3.8.1 and LibVEX; rerun with -h for copyright info
==22496== Command: ./TurtleGraphics ColourAndPatterns
==22496==

@@@@@@@@@@@@@@@@@@@@@x
@                   @ x
@ ***************** @  x
@ *               * @   x
@ * $$$$$$$$$$$$$ * @    x
@ * $           $ * @     x
@ * $ +++++++++ $ * @      x
@ * $ +       + $ * @       x
@ * $ + OOOOO + $ * @        x
@ * $ + O   O + $ * @         x
@ * $ + O x O + $ * @          x
@ * $ + O   O + $ * @         x
@ * $ + OOOOO + $ * @        x
@ * $ +       + $ * @       x
@ * $ +++++++++ $ * @      x
@ * $           $ * @     x
@ * $$$$$$$$$$$$$ * @    x
@ *               * @   x
@ ***************** @  x
@                   @ x
@@@@@@@@@@@@@@@@@@@@@x


==22496==
==22496== HEAP SUMMARY:
==22496==     in use at exit: 0 bytes in 0 blocks
==22496==   total heap usage: 300 allocs, 300 frees, 7,575 bytes allocated
==22496==
==22496== All heap blocks were freed -- no leaks are possible
==22496==
==22496== For counts of detected and suppressed errors, rerun with: -v
==22496== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 8 from 6)
```

Invalid Input 1:

Test file: invalid1.txt

Test case:
Valid argument has an incompatible data type.

File contents:

```
MOVE 1
DRAW 1
MOVE egg
```

Command-line:

```
[18925997@saeshell02p ASSIGNMENT]$ ./TurtleGraphics invalid1.txt
```

Terminal Output:

```
Invalid file format: invalid1.txt
"move egg" HERE--->Line:3
```

Command-line: -*with valgrind*

```
[18925997@saeshell02p ASSIGNMENT]$ valgrind ./TurtleGraphics invalid1.txt
```

Terminal Output:

```
==22957== Memcheck, a memory error detector
==22957== Copyright (C) 2002-2012, and GNU GPL'd, by Julian Seward et al.
==22957== Using Valgrind-3.8.1 and LibVEX; rerun with -h for copyright info
==22957== Command: ./TurtleGraphics invalid1.txt
==22957==
Invalid file format: invalid1.txt
"move egg" HERE--->Line:3
==22957==
==22957== HEAP SUMMARY:
==22957==     in use at exit: 0 bytes in 0 blocks
==22957==   total heap usage: 11 allocs, 11 frees, 760 bytes allocated
==22957==
==22957== All heap blocks were freed -- no leaks are possible
==22957==
==22957== For counts of detected and suppressed errors, rerun with: -v
==22957== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 8 from 6)
```

Invalid Input 2:

Test file: invalid2.txt

Test case:
Valid argument has no accompanying value.

File contents:

```
draw 1
move 3
rotate 2
draw
Move 3
Draw 5
```

Command-line:

```
[18925997@saeshell02p ASSIGNMENT]$ ./TurtleGraphics invalid2.txt
```

Terminal Output:

```
Invalid file format: invalid2.txt
"draw" HERE--->Line:4
```

Command-line: -*with valgrind*

```
[18925997@saeshell02p ASSIGNMENT]$ valgrind ./TurtleGraphics invalid2.txt
```

Terminal Output:

```
==22997== Memcheck, a memory error detector
==22997== Copyright (C) 2002-2012, and GNU GPL'd, by Julian Seward et al.
==22997== Using Valgrind-3.8.1 and LibVEX; rerun with -h for copyright info
==22997== Command: ./TurtleGraphics invalid2.txt
==22997==
Invalid file format: invalid2.txt
"draw" HERE--->Line:4
==22997==
==22997== HEAP SUMMARY:
==22997==     in use at exit: 0 bytes in 0 blocks
==22997==   total heap usage: 12 allocs, 12 frees, 784 bytes allocated
==22997==
==22997== All heap blocks were freed -- no leaks are possible
==22997==
==22997== For counts of detected and suppressed errors, rerun with: -v
==22997== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 8 from 6)
```

Invalid Input 3:

Test file: invalid3.txt

Test case:
Too many values after a valid argument.

File contents:

```
DRAW 1
MOVE 3 3
PATTERN X
```

Command-line:

```
[18925997@saeshell02p ASSIGNMENT]$ ./TurtleGraphics invalid3.txt
```

Terminal Output:

```
Invalid file format: invalid3.txt
"MOVE 3 3" HERE--->Line:2
```

Command-line: -*with valgrind*

```
[18925997@saeshell02p ASSIGNMENT]$ valgrind ./TurtleGraphics invalid3.txt
```

Terminal Output:

```
==23078== Memcheck, a memory error detector
==23078== Copyright (C) 2002-2012, and GNU GPL'd, by Julian Seward et al.
==23078== Using Valgrind-3.8.1 and LibVEX; rerun with -h for copyright info
==23078== Command: ./TurtleGraphics invalid3.txt
==23078==
Invalid file format: invalid3.txt
"MOVE 3 3" HERE--->Line:2
==23078==
==23078== HEAP SUMMARY:
==23078==     in use at exit: 0 bytes in 0 blocks
==23078==   total heap usage: 6 allocs, 6 frees, 688 bytes allocated
==23078==
==23078== All heap blocks were freed -- no leaks are possible
==23078==
==23078== For counts of detected and suppressed errors, rerun with: -v
==23078== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 8 from 6)
```

Invalid Input 4:

Test file: invalid4.txt

Test case:

Invalid argument.

File contents:

```
DRAW 5
MOVE 5
BOP-IT 10
```

Command-line:

```
[18925997@saeshell02p ASSIGNMENT]$ ./TurtleGraphics invalid4.txt
```

Terminal Output:

```
Invalid file format: invalid4.txt
"BOP-IT 10 " HERE--->Line:3
```

Command-line: -*with valgrind*

```
[18925997@saeshell02p ASSIGNMENT]$ valgrind ./TurtleGraphics invalid4.txt
```

Terminal Output:

```
==23126== Memcheck, a memory error detector
==23126== Copyright (C) 2002-2012, and GNU GPL'd, by Julian Seward et al.
==23126== Using Valgrind-3.8.1 and LibVEX; rerun with -h for copyright info
==23126== Command: ./TurtleGraphics invalid4.txt
==23126==
Invalid file format: invalid4.txt
"BOP-IT 10 " HERE--->Line:3
==23126==
==23126== HEAP SUMMARY:
==23126==     in use at exit: 0 bytes in 0 blocks
==23126==   total heap usage: 9 allocs, 9 frees, 736 bytes allocated
==23126==
==23126== All heap blocks were freed -- no leaks are possible
==23126==
==23126== For counts of detected and suppressed errors, rerun with: -v
```

Invalid Input 5:

Test file: none

Test case:
File does not exist.

Command-line:

```
[18925997@saeshell02p ASSIGNMENT]$ ./TurtleGraphics imaginary.txt
```

Terminal Output:

```
Error opening file: imaginary.txt
: No such file or directory
```

Command-line: -*with valgrind*

```
[18925997@saeshell02p ASSIGNMENT]$ valgrind ./TurtleGraphics imaginary.txt
```

Terminal Output:

```
==23259== Memcheck, a memory error detector
==23259== Copyright (C) 2002-2012, and GNU GPL'd, by Julian Seward et al.
==23259== Using Valgrind-3.8.1 and LibVEX; rerun with -h for copyright info
==23259== Command: ./TurtleGraphics imaginary.txt
==23259==
Error opening file: imaginary.txt
: No such file or directory
==23259==
==23259== HEAP SUMMARY:
==23259==     in use at exit: 0 bytes in 0 blocks
==23259==   total heap usage: 4 allocs, 4 frees, 1,208 bytes allocated
==23259==
==23259== All heap blocks were freed -- no leaks are possible
==23259==
==23259== For counts of detected and suppressed errors, rerun with: -v
==23259== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 8 from 6)
```

TurtleGraphicsSimple

This version of the assignment generates line art with a white background and black foreground. That is, any FG or BG commands should be ignored.

Command-lines

Standard execution:

```
./TurtleGraphicsSimple <filename>
```

Valid input 1:

Test file: ColoursAndPatterns.txt

File contents:

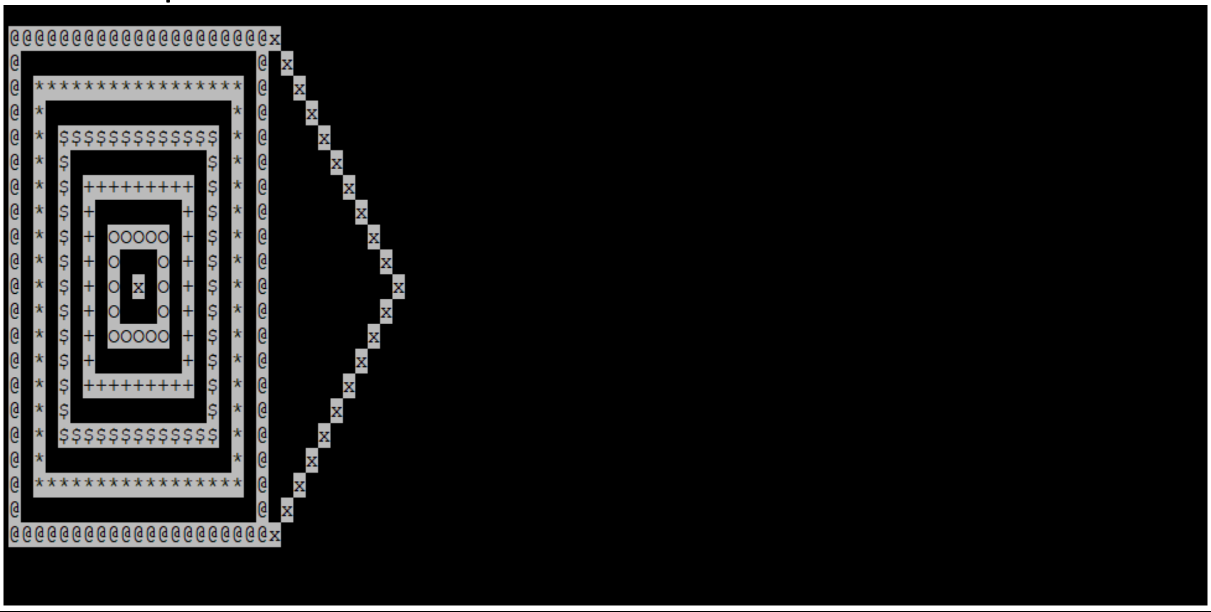
```
rotate -90
move 1
rotate 90
pattern @
draw 20
rotate -90
draw 20
rotate -90
draw 20
rotate -90
draw 20
rotate -90
move 2
rotate -90
move 2
rotate 90
pattern *
draw 16
rotate -90
draw 16
rotate -90
draw 16
rotate -90
draw 16
rotate -90
move 2
rotate -90
move 2
rotate 90
pattern $
BG 0
FG 3
draw 12
rotate -90
draw 12
rotate -90
draw 12
rotate -90
draw 12
rotate -90
move 2
rotate -90
move 2
```

```
rotate 90
BG 7
FG 1
pattern +
draw 8
rotate -90
draw 8
rotate -90
draw 8
rotate -90
draw 8
rotate -90
move 2
rotate -90
move 2
rotate 90
BG 6
pattern O
draw 4
rotate -90
draw 4
rotate -90
draw 4
rotate -90
draw 4
rotate -90
BG 1
FG 6
move 2
rotate -90
move 2
rotate 90
fg 2
bg 0
pattern x
draw 1
move 10
rotate 90
move 10
rotate -135
draw 14
rotate -90
draw 15
```

Command-line:

```
[18925997@saeshell02p ASSIGNMENT]$ ./TurtleGraphicsSimple ColourAndPatterns.txt
```

Terminal Output:



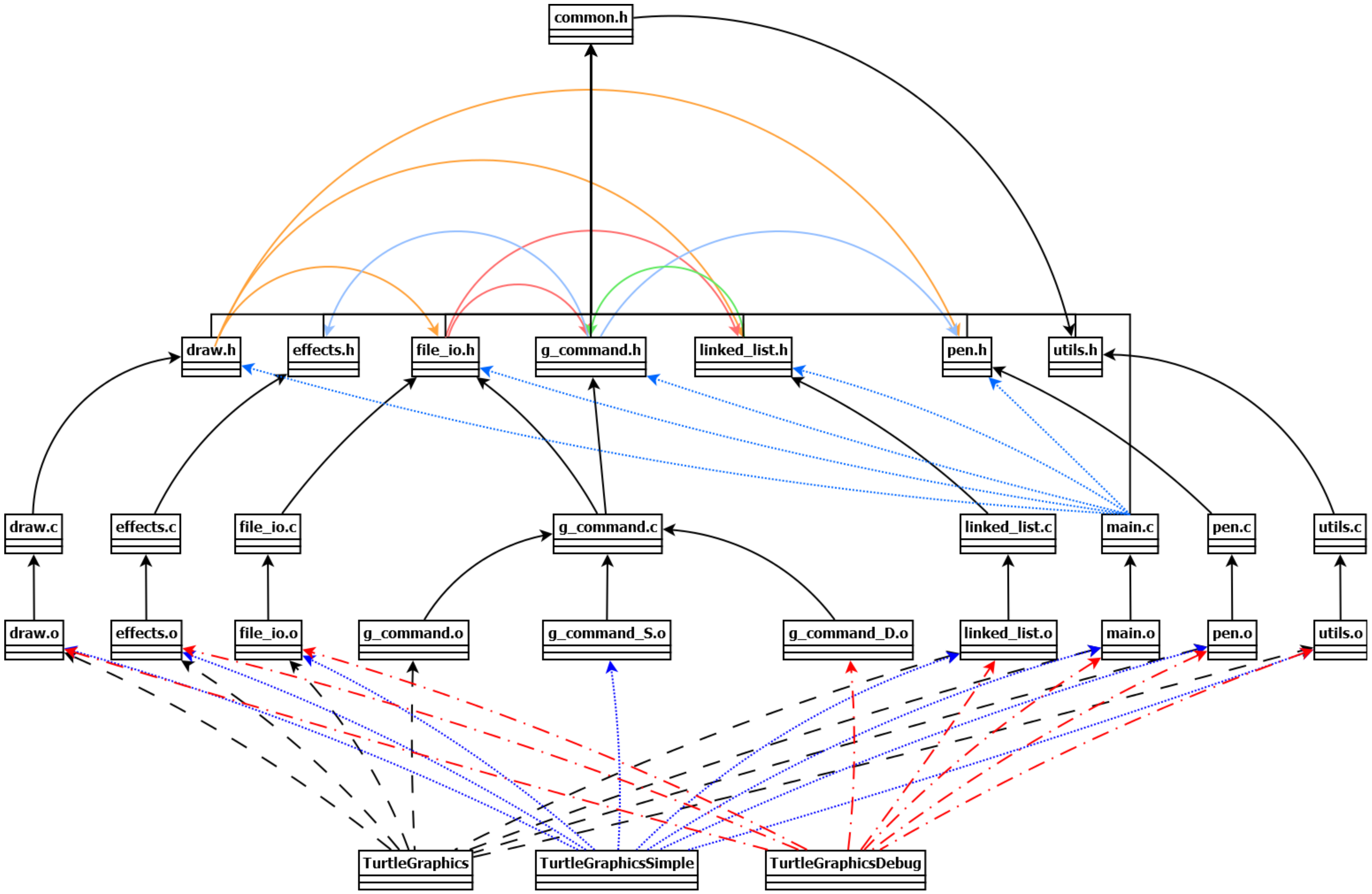
```
[18925997@saeshell02p ASSIGNMENT]$
```

Log: *in graphics.log*

```
---
MOVE (  0.000,  0.000)-(  0.000,  1.000)
DRAW ( 19.000,  1.000)-( 20.000,  1.000)
DRAW ( 20.000, 20.000)-( 20.000, 21.000)
DRAW (  1.000, 21.000)-(  0.000, 21.000)
DRAW ( -0.000,  2.000)-( -0.000,  1.000)
MOVE ( -0.000,  1.000)-(  2.000,  1.000)
MOVE (  2.000,  1.000)-(  2.000,  3.000)
DRAW ( 17.000,  3.000)-( 18.000,  3.000)
DRAW ( 18.000, 18.000)-( 18.000, 19.000)
DRAW (  3.000, 19.000)-(  2.000, 19.000)
DRAW (  2.000,  4.000)-(  2.000,  3.000)
MOVE (  2.000,  3.000)-(  4.000,  3.000)
MOVE (  4.000,  3.000)-(  4.000,  5.000)
DRAW ( 15.000,  5.000)-( 16.000,  5.000)
DRAW ( 16.000, 16.000)-( 16.000, 17.000)
DRAW (  5.000, 17.000)-(  4.000, 17.000)
DRAW (  4.000,  6.000)-(  4.000,  5.000)
MOVE (  4.000,  5.000)-(  6.000,  5.000)
```

```
MOVE (  6.000,  5.000)-(  6.000,  7.000)
DRAW ( 13.000,  7.000)-( 14.000,  7.000)
DRAW ( 14.000, 14.000)-( 14.000, 15.000)
DRAW (  7.000, 15.000)-(  6.000, 15.000)
DRAW (  6.000,  8.000)-(  6.000,  7.000)
MOVE (  6.000,  7.000)-(  8.000,  7.000)
MOVE (  8.000,  7.000)-(  8.000,  9.000)
DRAW ( 11.000,  9.000)-( 12.000,  9.000)
DRAW ( 12.000, 12.000)-( 12.000, 13.000)
DRAW (  9.000, 13.000)-(  8.000, 13.000)
DRAW (  8.000, 10.000)-(  8.000,  9.000)
MOVE (  8.000,  9.000)-( 10.000,  9.000)
MOVE ( 10.000,  9.000)-( 10.000, 11.000)
DRAW ( 10.000, 11.000)-( 11.000, 11.000)
MOVE ( 11.000, 11.000)-( 21.000, 11.000)
MOVE ( 21.000, 11.000)-( 21.000,  1.000)
DRAW ( 30.192, 10.192)-( 30.899, 10.899)
DRAW ( 21.000, 20.799)-( 20.293, 21.50
```

Make File Dependencies:


common.h
draw.h
effects.h
file_io.h
g_command.h
linked_list.h
pen.h
utils.h
draw.c
effects.c
file_io.c
g_command.c
linked_list.c
main.c
pen.c
utils.c
draw.o
effects.o
file_io.o
g_command.o
g_command_S.o
g_command_D.o
linked_list.o
main.o
pen.o
utils.o
TurtleGraphics
TurtleGraphicsSimple
TurtleGraphicsDebug