Folder src

24 printable files

(file list disabled)

src\Events\KeyEvents\CtrlC\CtrlC.c

```c
//includes
#include "CtrlC.h"
#include "../../SafeExit/SafeExit.h"

//definition
void CtrlC(){
    SafeExit();
}
```

src\Events\KeyEvents\CtrlC\CtrlC.h

```c
#ifndef CTRL_C_H_
#define CTRL_C_H_

//declarations
void CtrlC();

#endif
```

src\Events\SafeExit\SafeExit.c

```c
//include
#include "../../Globals.h"
#include "SafeExit.h"

//definition
void SafeExit(){
    shouldExit = 1;
}
```

src\Events\SafeExit\SafeExit.h

```c
#ifndef EXIT_H_
#define EXIT_H_

//declarations
void SafeExit();

#endif
```

src\Globals.h

```h
#ifndef GLOBALS_H_
#define GLOBALS_H_

//global variable declarations
extern char* fileLocation;
extern char* content;
extern int shouldExit;
extern int size;
extern int jump;
extern int foot;

#endif
```

src\Input\InputHandler\Foot\Foot.c

```c
//includes
#include <stdlib.h>
#include "../../../Math/Min/Min.h"
#include "Foot.h"

//definition
void Foot(char tokenInput[64][16], int* footPointer, int size){
    //get foot location address and set to foot variable
    *footPointer = strtol(tokenInput[1], NULL, 16);

    //ensure jump is less than size + 1
    *footPointer = Min(*footPointer, size+1);
}
```

src\Input\InputHandler\Foot\Foot.h

```h
#ifndef FOOT_H_
#define FOOT_H_

//declaration
void Foot(char tokenInput[64][16], int* footPointer, int size);

#endif
```

src\Input\InputHandler\InputHandler.c

```c
//include
#include <stdio.h>
#include <string.h>
#include "../../Globals.h"
#include "InputHandler.h"
#include "Save/Save.h"
#include "Write/Write.h"
#include "Jump/Jump.h"
#include "Foot/Foot.h"

```

```c
//definition
void InputHandler(char tokenInput[64][16]){
    //command hand selection and hand off
    if(strcmp(tokenInput[0], "save\n") == 0){
        Save(fileLocation, content, size);
    }else if(strcmp(tokenInput[0], "write") == 0){
        Write(content, tokenInput);
    }else if(strcmp(tokenInput[0], "jump") == 0){
        Jump(tokenInput, &jump);
    }else if(strcmp(tokenInput[0], "foot") == 0){
        Foot(tokenInput, &foot, size+1);
    }else{
        printf("Invalid command\n");
    }
}
```

src\Input\InputHandler\InputHandler.h

```c
#ifndef INPUT_HANDLER_H_
#define INPUT_HANDLER_H_

//declarations
void InputHandler(char tokenInput[64][16]);

#endif
```

src\Input\InputHandler\Jump\Jump.c

```c
//includes
#include <stdlib.h>
#include "../../../Math/Max/Max.h"
#include "Jump.h"

//definition
void Jump(char tokenInput[64][16], int* jumpPointer){
    //get jump location address and set to jump variable
    *jumpPointer = strtol(tokenInput[1], NULL, 16);

    //ensure jump is 0 or more
    *jumpPointer = Max(*jumpPointer, 0);
}
```

src\Input\InputHandler\Jump\Jump.h

```c
#ifndef JUMP_H_
#define JUMP_H_

//declaration
void Jump(char tokenInput[64][16], int* jumpPointer);

#endif
```

src\Input\InputHandler\Save\Save.c

```c
//includes
#include <stdio.h>
#include "Save.h"

//definition
void Save(char* location, char* content, int size){
    //open file
    FILE* fp = fopen(location, "wb");

    //write data to file
    fwrite(content, size, 1, fp);

    //close file
    fclose(fp);

    //respond to input
    printf("Saved to file\n");
}
```

src\Input\InputHandler\Save\Save.h

```c
#ifndef SAVE_H_
#define SAVE_H_

//declaration
void Save(char* location, char* content, int size);

#endif
```

src\Input\InputHandler\Write\Write.c

```c
//includes
#include <stdio.h>
#include <stdlib.h>
#include "Write.h"

//definition
void Write(char* content, char chunkedInput[64][16]){
    //get lower address line
    int lowerAddr = strtol(chunkedInput[1], NULL, 16);

    //get upper address line
    int upperAddr = strtol(chunkedInput[2], NULL, 16);

    //get lower byte
    int lowerByte = atoi(chunkedInput[3]);

    //get upper byte
    int upperByte = atoi(chunkedInput[4]);

```

```
    //get line and data size
    int lineSize = upperAddr - lowerAddr;
    int dataSize = upperByte - lowerAddr;

    //get data and store to data array
    int* data = malloc(dataSize);
    for (int i = 0; i < dataSize; i++){
        sscanf(chunkedInput[5+i], "%hhd", &(data[i]));
    }

    //write data
    for (int line = 0; line < lineSize; line++){
        for (int byte = 0; byte < dataSize; byte++){
            int location = (lowerAddr+line)*16+lowerByte+byte;
            content[location] = data[byte];
        }
    }
}
```

src\Input\InputHandler\Write\Write.h

```
#ifndef WRITE_H_
#define WRITE_H_

//declaration
void Write(char* content, char chunkedInput[64][16]);

#endif
```

src\Input\Tokenize\Tokenize.c

```
//include
#include <string.h>
#include "Tokenize.h"

//definition
int Tokenize(char* content, char tokenedOutput[64][16], char* delimiter, int numTokens, int tokenSize){
    int count = 0;
    char* token = strtok(content, delimiter);
    while (token != NULL && count < numTokens){
        strncpy(tokenedOutput[count], token, tokenSize);
        count += 1;
        token = strtok(NULL, delimiter);
    }
    return count;
}
```

src\Input\Tokenize\Tokenize.h

```
#ifndef TOKENIZE_H_
#define TOKENIZE_H_
```

```
3
4  //declarations
5  int Tokenize(char* content, char tokenedOutput[64][16], char* delimiter, int numTokens, int
   tokenSize);
6
7  #endif
```

src\Math\Max\Max.c

```
1  //include
2  #include "Max.h"
3
4  //definitions
5  int Max(int a, int b){
6      if (a > b){
7          return a;
8      }
9      return b;
10 }
```

src\Math\Max\Max.h

```
1  #ifndef MAX_H_
2  #define MAX_H_
3
4  //declarations
5  int Max(int a, int b);
6
7  #endif
```

src\Math\Min\Min.c

```
1  //include
2  #include "Min.h"
3
4  //definitions
5  int Min(int a, int b){
6      if (a < b){
7          return a;
8      }
9      return b;
10 }
```

src\Math\Min\Min.h

```
1  #ifndef MIN_H_
2  #define MIN_H_
3
4  //declarations
5  int Min(int a, int b);
6
```

```
#endif
```

src\Output\DisplayContent\DisplayContent.c

```
//include
#include <stdio.h>
#include "DisplayContent.h"

//definition
void DisplayContent(char* content, int size, int jump, int foot){
    //display header
    printf("      LA_X LA_Y RA_X RA_Y D_UP D_DW D_LF D_RH D_BA D_BB D_BX D_BY D_L1 D_L2 D_R1 D_R2\n");

    //loop through each line of content
    //one line for every 16 bytes
    for (int line = 0; line < (int)((foot-jump)/16); line++){
        //print line address
        printf("$%04x ", line*16+jump);

        //loop through each byte of line
        for (int byte = 0; byte < 16; byte++){
            //get value in int8 form
            __int8 value = (__int8) content[line*16+byte];

            //print value with leading sign(special manipulation for positive)
            if (value >= 0){
                printf("+%03i ", value);
            }else{
                printf("%04i ", value);
            }
        }

        //create new line
        printf("\n");
    }
}
```

src\Output\DisplayContent\DisplayContent.h

```
#ifndef DISPLAY_CONTENT_H_
#define DISPLAY_CONTENT_H_

//declarations
void DisplayContent(char* content, int size, int jump, int foot);

#endif
```

src\main.c

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

```c
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include "Globals.h"
#include "Events/KeyEvents/CtrlC/CtrlC.h"
#include "Events/SafeExit/SafeExit.h"
#include "Output/DisplayContent/DisplayContent.h"
#include "Input/Tokenize/Tokenize.h"
#include "Input/InputHandler/InputHandler.h"

//initialization of globals
char* fileLocation;
char* content;
int shouldExit = 0;
int size = 0;
int jump = 0;
int foot = 0;

//main method
int main(int argc, char** argv){
    //set ctrl c signal
    signal(SIGINT, CtrlC);

    //check if location passed
    if (argc <= 1){
        //location not passed, error out
        printf("Error, no file passed");
        return 1;
    }

    //set file location
    fileLocation = argv[1];

    //load file into file pointer
    FILE* filePointer = fopen(argv[1], "rb");

    //check if file exists
    if (filePointer == NULL){
        printf("Cannot find %s: No such file", argv[1]);
        return 1;
    }

    //seek file size, store to size
    fseek(filePointer, 0 , SEEK_END);
    size = ftell(filePointer);

    //ensure size is a multiple of 16
    if (size % 16 != 0){
        printf("Error, missing data, size is not a multiple of 16");
        return 1;
    }

    //create content buffer with allocation of size + 1 -needed for end char-
    //read file into content buffer
    content = malloc(size+1);
    content[size] = '\0';
```

```
    fseek(filePointer, 0, SEEK_SET);
    fread(content, size, 1, filePointer);

    //close file for safety
    fclose(filePointer);

    //set jump and foot variables
    jump = 0;
    foot = size;

    //output file contents
    DisplayContent(content, size, jump, foot);

    //create input buffer of size 1024(64*16)
    //create token input buffer of size 1024(64*16)
    char inputBuffer[1024];
    char tokenInput[64][16];

    //input loop
    while (shouldExit == 0){
        //print line leader
        printf("Editor ~ ");

        //get input and push into input buffer
        fgets(inputBuffer, sizeof(inputBuffer), stdin);

        //tokenize input and store number of tokens
        int numTokens = Tokenize(inputBuffer, tokenInput, " ", 64, 16);

        //handle input
        InputHandler(tokenInput);

        //re-output file contents
        DisplayContent(content, size, jump, foot);
    }

    //return successful
    return 0;
}
```

Autonomous editor

After we finished working on the autonomous revisions, we decided that we needed to develop an editor to modify and view our new routine files. To accomplish this multiple languages were tried out, and eventually the C programming language was decided on. We would make a terminal based editor that used commands instead of keystrokes to operate. These essential commands would write, jump, foot, and save. Write would use 5+ parameters to make changes to the loaded data in RAM. The first pair of two parameters would be line address range (inclusive of bottom), the second pair of two parameters would be the data range within the lines (inclusive of bottom). The jump command would cut off values beneath the parameter given, thus jumping to a segment of code for easy editing. The foot command would cut off values at or above the parameter given, the setting the bottom/foot of the code. The save command would overwrite the file with the changes in ram. This was accomplished shortly with just a few errors. We had an error where after saving the file (with or without changes) if you pressed Ctrl-C to exit the file would be emptied despite having contents written over when saved. To fix this we had to an interrupt override event to the Ctrl-C command that would perform a safe for saving exit. In the future of development we hope add to serial communication functionality to this program and a second program on the brain so that we can modify the files without having to remove the sd card from the brain. We also hope to then upon completion of that development add a GUI to the editor program so that it is more accessible for others if we choose to make our autonomous system public.

Autonomous changes

These last couple weeks for the code I worked on revising the autonomous routine saving/loading to be not only cleaner, but faster too. To recap, our autonomous system uses signed 8 bit integers to store input values that range from -127 to +127 inclusive in a data type that can store -128 to +127 inclusive. This allows us to use only one byte for each input inside program memory to store inputs. However, when we would save the values we would up to 5 bytes to store these values. This was because we stored the values as numerical text instead of the raw data, had to store the sign as its own text byte, and had to usa a comma to separate each value. To fix this we decided to instead convert the 8 bit integers to raw bytes, convert them to their respective byte characters, and make a file filled with these to cleanly store values. This was a great change and cleaned up / fastened the code at the same time due to the slowness of strings. It also compacted the files and made them "easier" to read. However we encountered the issue of not having a good editor to edit and view these files. So we decided that we would need to solve this problem ourselves later.