EXERCISE 3

s1290251

Tithira Withanaarachchi

Exercise 3

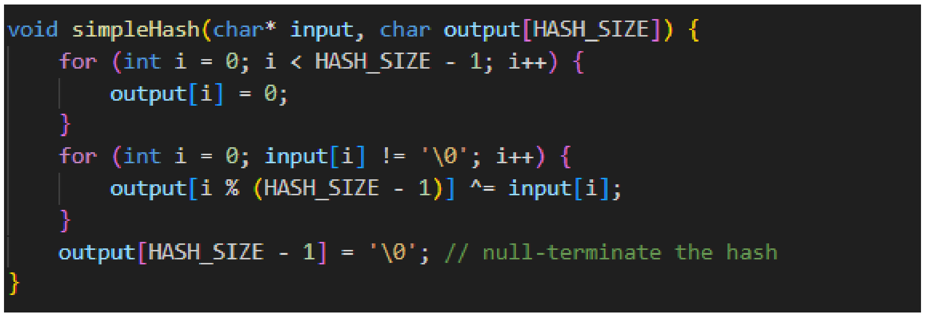
1),2)Code Review and Improvements

Observation:

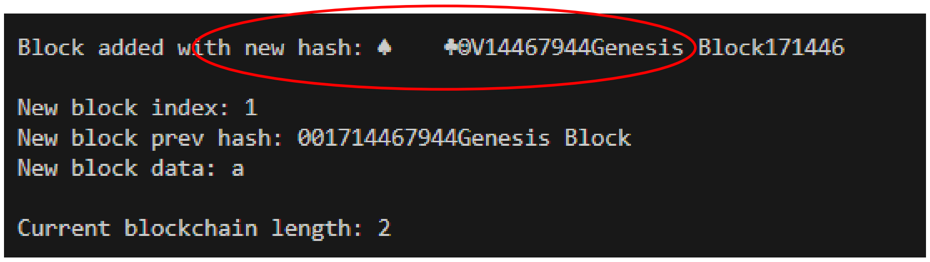
strcpy_s() function is not compatible with Linux and Mac OS.

Main Functionality Fault:

- simpleHash function does not create a valid Hash for the blocks(The Hash contains different signs that should not be included in a hash).



```c
void simpleHash(char* input, char output[HASH_SIZE]) {
    for (int i = 0; i < HASH_SIZE - 1; i++) {
        output[i] = 0;
    }
    for (int i = 0; input[i] != '\0'; i++) {
        output[i % (HASH_SIZE - 1)] ^= input[i];
    }
    output[HASH_SIZE - 1] = '\0'; // null-terminate the hash
}
```

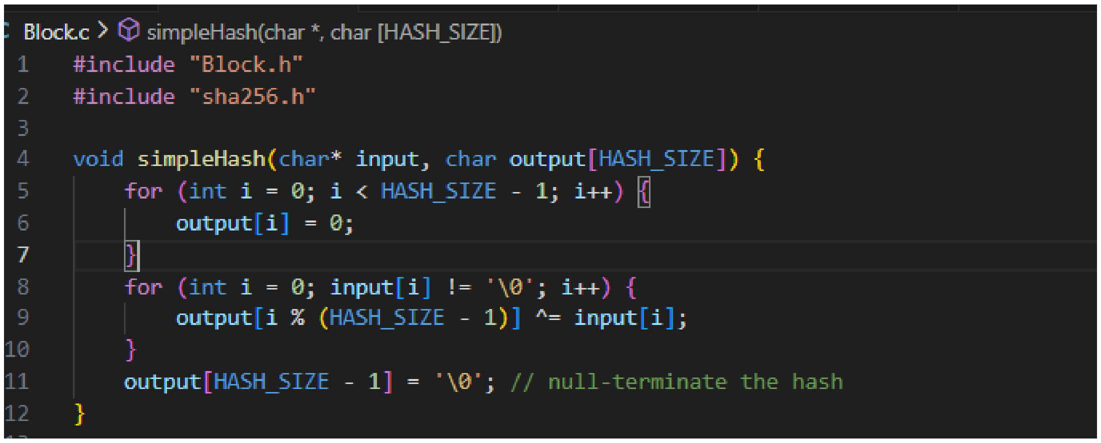


```
Block added with new hash: ♠    ♣0V14467944Genesis Block171446

New block index: 1
New block prev hash: 001714467944Genesis Block
New block data: a

Current blockchain length: 2
```

Improvements related to different drawbacks:



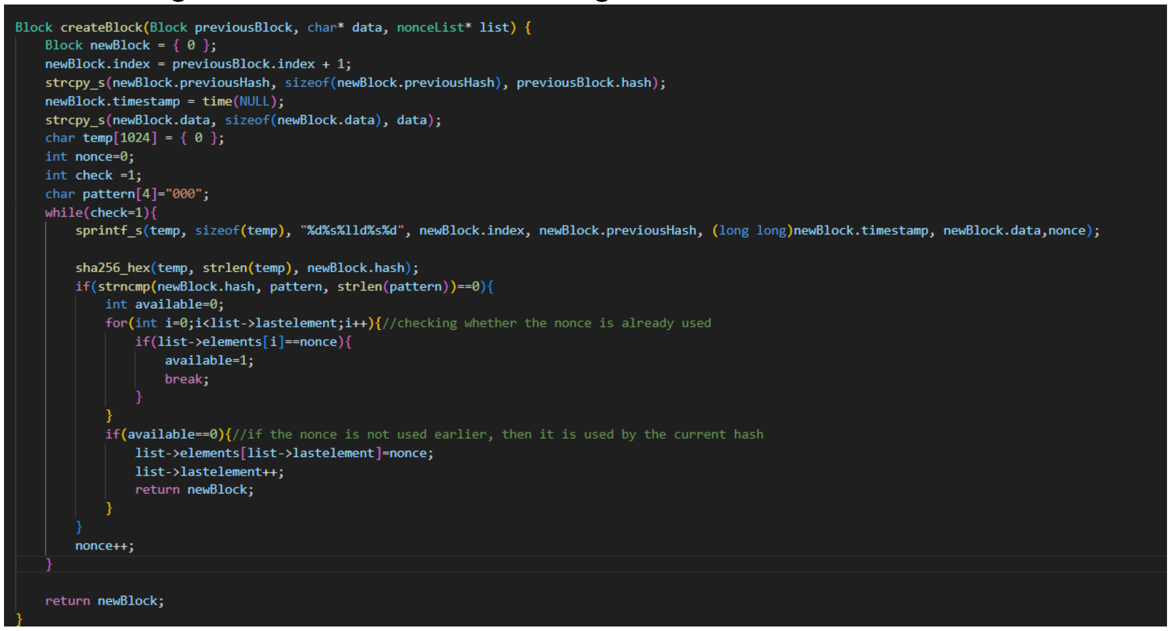
```c
Block.c > simpleHash(char *, char [HASH_SIZE])
#include "Block.h"
#include "sha256.h"

void simpleHash(char* input, char output[HASH_SIZE]) {
    for (int i = 0; i < HASH_SIZE - 1; i++) {
        output[i] = 0;
    }
    for (int i = 0; input[i] != '\0'; i++) {
        output[i % (HASH_SIZE - 1)] ^= input[i];
    }
    output[HASH_SIZE - 1] = '\0'; // null-terminate the hash
}
```

- The above code is replaced with the functionality provided by the below-mentioned git repository to generate a valid hash.

  https://github.com/983/SHA-256.git

- Later as an additional improvement, nonce has been introduced to the code and the hash with 3 leading zeros is considered when adding a new block.

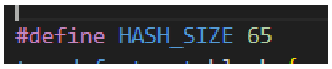


```c
Block createBlock(Block previousBlock, char* data, nonceList* list) {
    Block newBlock = { 0 };
    newBlock.index = previousBlock.index + 1;
    strcpy_s(newBlock.previousHash, sizeof(newBlock.previousHash), previousBlock.hash);
    newBlock.timestamp = time(NULL);
    strcpy_s(newBlock.data, sizeof(newBlock.data), data);
    char temp[1024] = { 0 };
    int nonce=0;
    int check =1;
    char pattern[4]="000";
    while(check=1){
        sprintf_s(temp, sizeof(temp), "%d%s%lld%s%d", newBlock.index, newBlock.previousHash, (long long)newBlock.timestamp, newBlock.data,nonce);

        sha256_hex(temp, strlen(temp), newBlock.hash);
        if(strncmp(newBlock.hash, pattern, strlen(pattern))==0){
            int available=0;
            for(int i=0;i<list->lastelement;i++){//checking whether the nonce is already used
                if(list->elements[i]==nonce){
                    available=1;
                    break;
                }
            }
            if(available==0){//if the nonce is not used earlier, then it is used by the current hash
                list->elements[list->lastelement]=nonce;
                list->lastelement++;
                return newBlock;
            }
        }
        nonce++;
    }

    return newBlock;
}
```

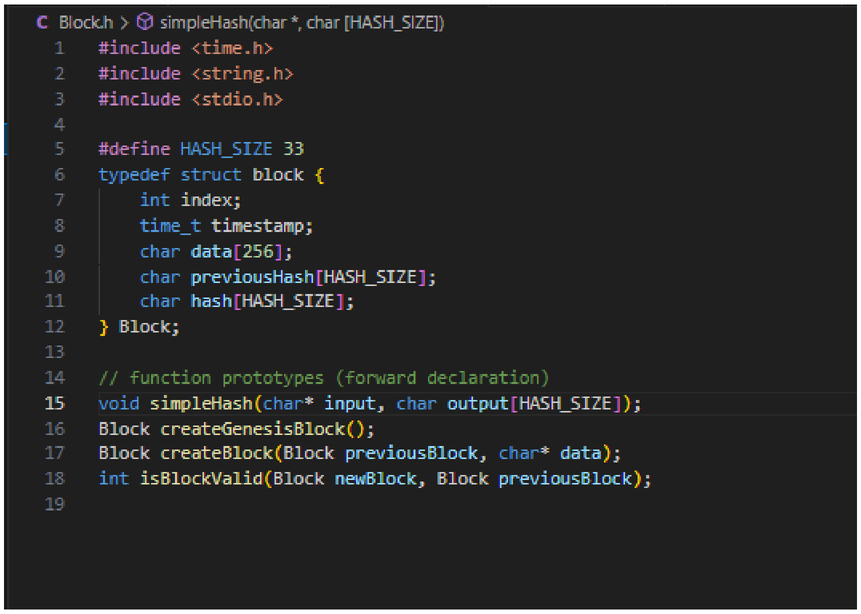
- In SHA256, a hexadecimal number with 64 characters is generated as a hash. Therefore, HASH_SIZE variable of the program should be changed to 65 instead of 33(HASH_SIZE is allocated for 65 elements as the last element is a null(64+1=65)).



```c
#define HASH_SIZE 65
```

Categorizing the code into different files:

Block.h



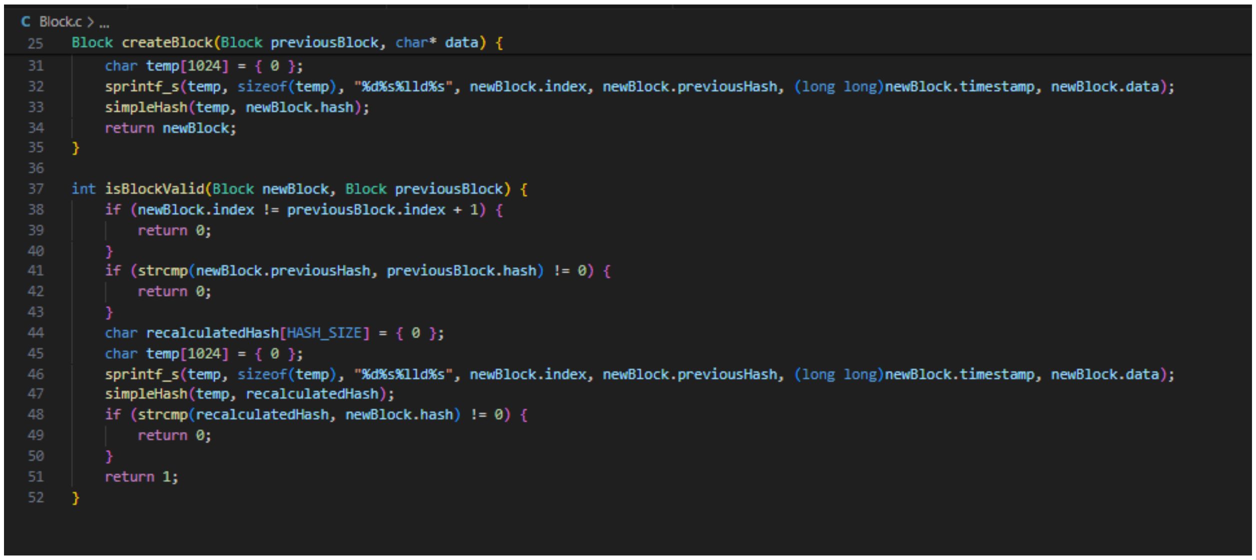
Block.h > simpleHash(char *, char [HASH_SIZE])

```c
#include <time.h>
#include <string.h>
#include <stdio.h>

#define HASH_SIZE 33
typedef struct block {
    int index;
    time_t timestamp;
    char data[256];
    char previousHash[HASH_SIZE];
    char hash[HASH_SIZE];
} Block;

// function prototypes (forward declaration)
void simpleHash(char* input, char output[HASH_SIZE]);
Block createGenesisBlock();
Block createBlock(Block previousBlock, char* data);
int isBlockValid(Block newBlock, Block previousBlock);
```

Block.c

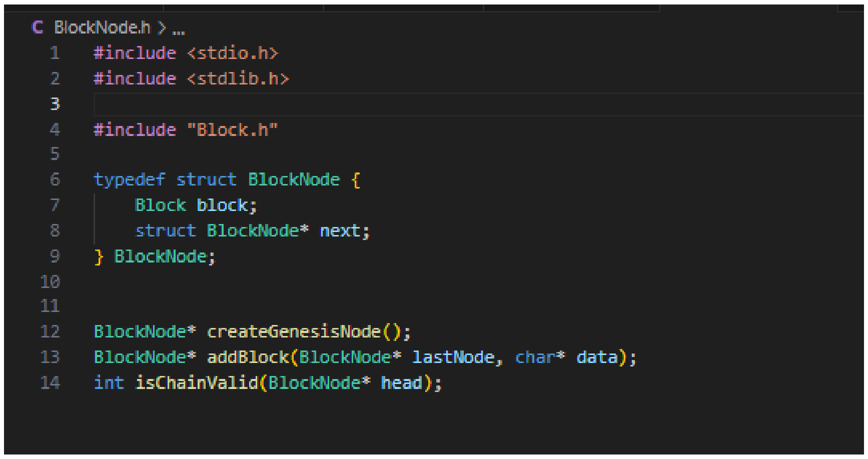


Block.c > ...

```c
Block createBlock(Block previousBlock, char* data) {
    char temp[1024] = { 0 };
    sprintf_s(temp, sizeof(temp), "%d%s%lld%s", newBlock.index, newBlock.previousHash, (long long)newBlock.timestamp, newBlock.data);
    simpleHash(temp, newBlock.hash);
    return newBlock;
}

int isBlockValid(Block newBlock, Block previousBlock) {
    if (newBlock.index != previousBlock.index + 1) {
        return 0;
    }
    if (strcmp(newBlock.previousHash, previousBlock.hash) != 0) {
        return 0;
    }
    char recalculatedHash[HASH_SIZE] = { 0 };
    char temp[1024] = { 0 };
    sprintf_s(temp, sizeof(temp), "%d%s%lld%s", newBlock.index, newBlock.previousHash, (long long)newBlock.timestamp, newBlock.data);
    simpleHash(temp, recalculatedHash);
    if (strcmp(recalculatedHash, newBlock.hash) != 0) {
        return 0;
    }
    return 1;
}
```

BlockNode.h



C BlockNode.h > ...

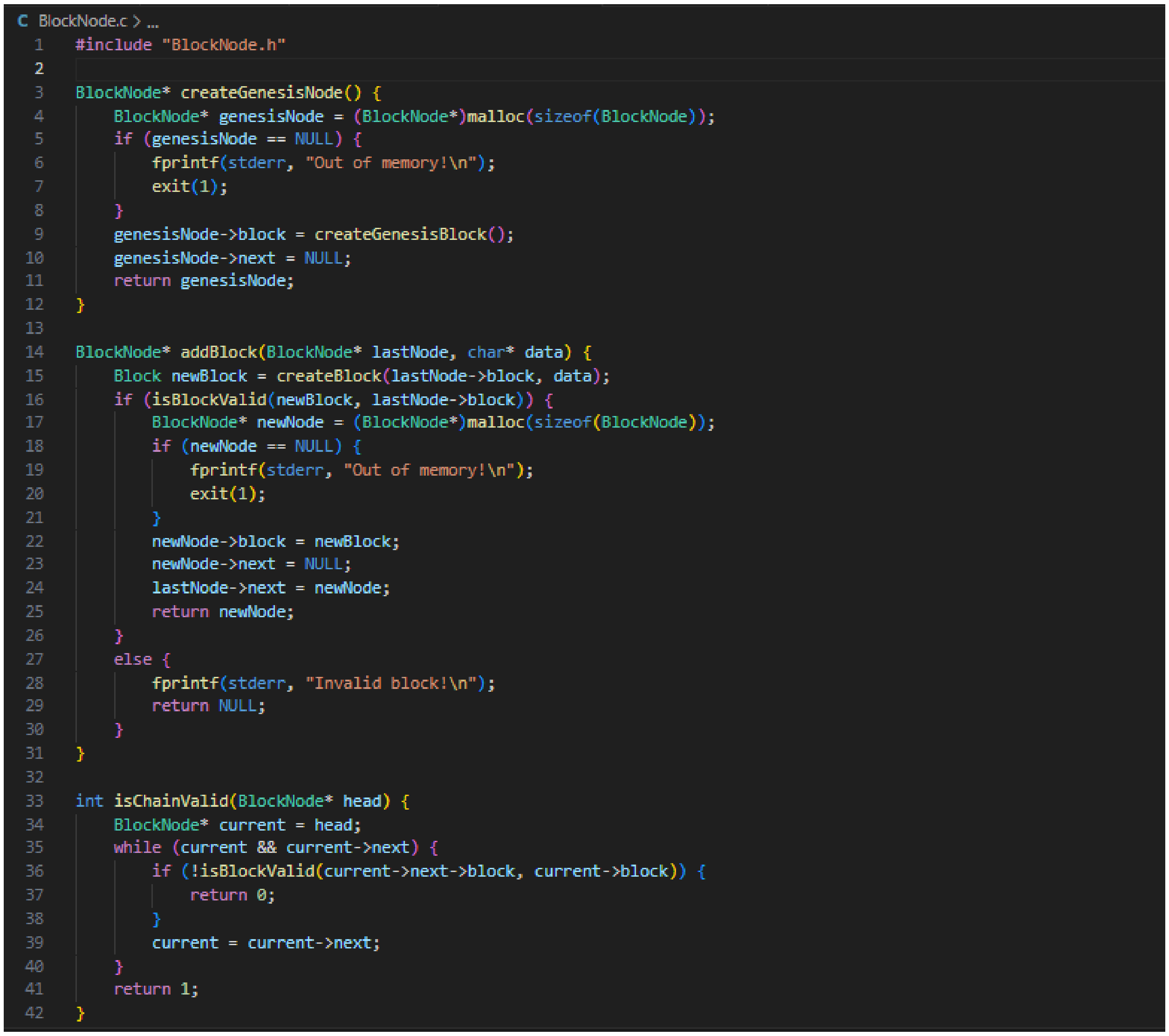
```c
#include <stdio.h>
#include <stdlib.h>

#include "Block.h"

typedef struct BlockNode {
    Block block;
    struct BlockNode* next;
} BlockNode;


BlockNode* createGenesisNode();
BlockNode* addBlock(BlockNode* lastNode, char* data);
int isChainValid(BlockNode* head);
```

BlockNode.c



C BlockNode.c > ...

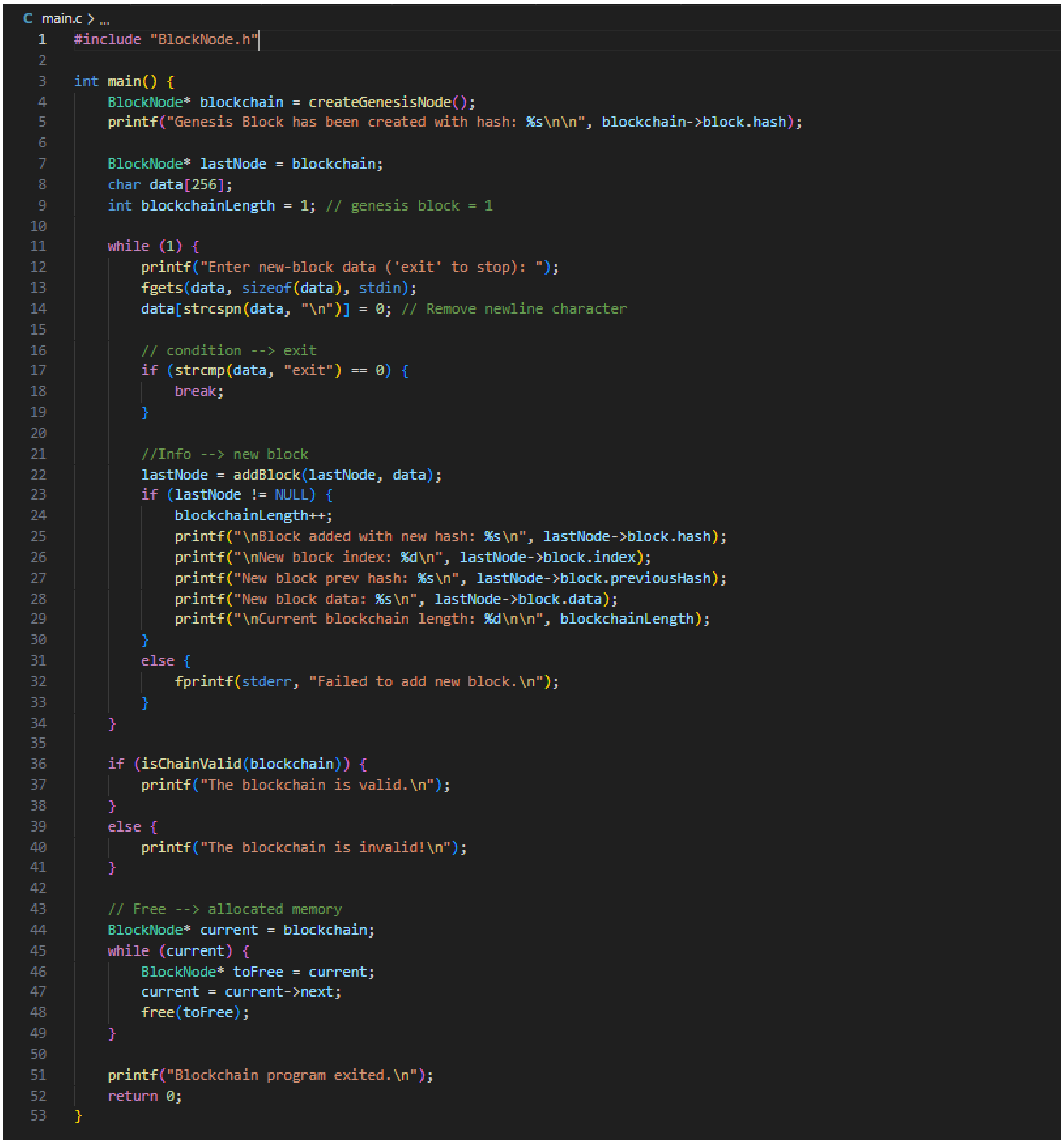
```c
#include "BlockNode.h"

BlockNode* createGenesisNode() {
    BlockNode* genesisNode = (BlockNode*)malloc(sizeof(BlockNode));
    if (genesisNode == NULL) {
        fprintf(stderr, "Out of memory!\n");
        exit(1);
    }
    genesisNode->block = createGenesisBlock();
    genesisNode->next = NULL;
    return genesisNode;
}

BlockNode* addBlock(BlockNode* lastNode, char* data) {
    Block newBlock = createBlock(lastNode->block, data);
    if (isBlockValid(newBlock, lastNode->block)) {
        BlockNode* newNode = (BlockNode*)malloc(sizeof(BlockNode));
        if (newNode == NULL) {
            fprintf(stderr, "Out of memory!\n");
            exit(1);
        }
        newNode->block = newBlock;
        newNode->next = NULL;
        lastNode->next = newNode;
        return newNode;
    }
    else {
        fprintf(stderr, "Invalid block!\n");
        return NULL;
    }
}

int isChainValid(BlockNode* head) {
    BlockNode* current = head;
    while (current && current->next) {
        if (!isBlockValid(current->next->block, current->block)) {
            return 0;
        }
        current = current->next;
    }
    return 1;
}
```

main.c



C main.c > ...

```c
#include "BlockNode.h"

int main() {
    BlockNode* blockchain = createGenesisNode();
    printf("Genesis Block has been created with hash: %s\n\n", blockchain->block.hash);

    BlockNode* lastNode = blockchain;
    char data[256];
    int blockchainLength = 1; // genesis block = 1

    while (1) {
        printf("Enter new-block data ('exit' to stop): ");
        fgets(data, sizeof(data), stdin);
        data[strcspn(data, "\n")] = 0; // Remove newline character

        // condition --> exit
        if (strcmp(data, "exit") == 0) {
            break;
        }

        //Info --> new block
        lastNode = addBlock(lastNode, data);
        if (lastNode != NULL) {
            blockchainLength++;
            printf("\nBlock added with new hash: %s\n", lastNode->block.hash);
            printf("\nNew block index: %d\n", lastNode->block.index);
            printf("New block prev hash: %s\n", lastNode->block.previousHash);
            printf("New block data: %s\n", lastNode->block.data);
            printf("\nCurrent blockchain length: %d\n\n", blockchainLength);
        }
        else {
            fprintf(stderr, "Failed to add new block.\n");
        }
    }

    if (isChainValid(blockchain)) {
        printf("The blockchain is valid.\n");
    }
    else {
        printf("The blockchain is invalid!\n");
    }

    // Free --> allocated memory
    BlockNode* current = blockchain;
    while (current) {
        BlockNode* toFree = current;
        current = current->next;
        free(toFree);
    }

    printf("Blockchain program exited.\n");
    return 0;
}
```

3)

Recording information related to a supply chain.

For explaining the application of the above block chain, a supply chain related to vehicle industry is considered.

When considering the supply chain in the vehicle industry from collecting raw materials to delivering the finished product to the customer, there are a lot of information that should be recorded in order to maintain a secure and transparent supply chain process. These information includes records related to the supply of raw materials, records related to the labour involved in the supply chain process, records related to the machinery used in the supply chain process and the stock management information. In order to maintain these records with high integrity and security, the above blockchain can be used.