BLG 233E

DATA STRUCTURES AND LABORATORY

CRN: 11146

REPORT OF HOMEWORK #3

Submission Date: 25.12.12

STUDENT NAME: TUĞRUL YATAĞAN

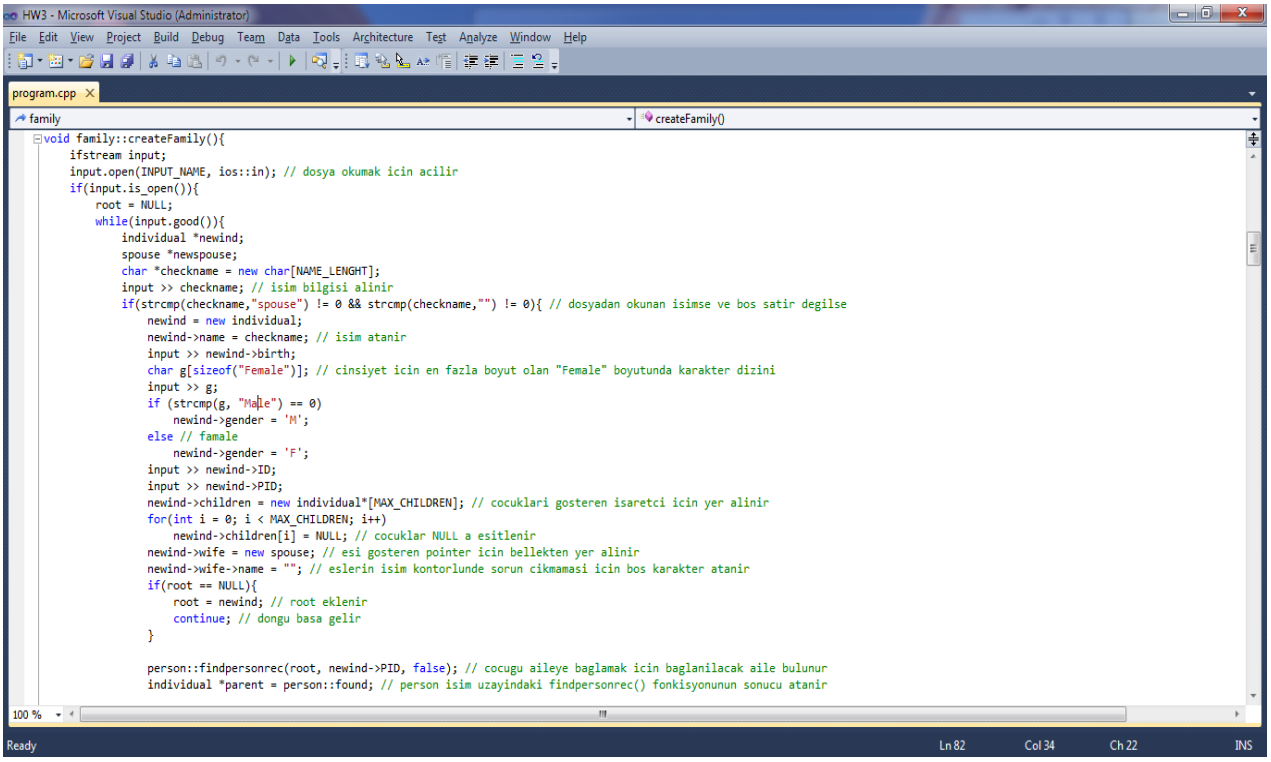
STUDENT NUMBER: 040100117

1. Introduction

In this homework, a family was represented using a tree. Each individual was represented as a node in the tree, and relationships will be represented with the connections between the nodes. There are two types of people, namely; individual and spouse.

2. Development and Operating Environments

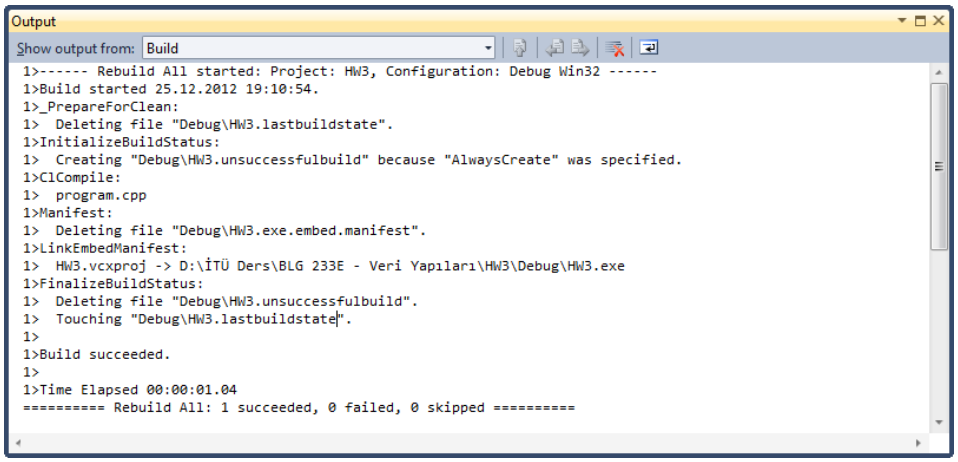
Microsoft Visual C++ 2010 environment has been used to write the source code in Windows 7 operation system and again Microsoft Visual C++ 2010 compiler was used to compile the program.




```
void family::createFamily(){
    ifstream input;
    input.open(INPUT_NAME, ios::in); // dosya okumak icin acilir
    if(input.is_open()){
        root = NULL;
        while(input.good()){
            individual *newind;
            spouse *newspouse;
            char *checkname = new char[NAME_LENGHT];
            input >> checkname; // isim bilgisi alinir
            if(strcmp(checkname,"spouse") != 0 && strcmp(checkname,"") != 0){ // dosyadan okunan isimse ve bos satir degilse
                newind = new individual;
                newind->name = checkname; // isim atanir
                input >> newind->birth;
                char g[sizeof("Female")]; // cinsiyet icin en fazla boyut olan "Female" boyutunda karakter dizini
                input >> g;
                if (strcmp(g, "Male") == 0)
                    newind->gender = 'M';
                else // famale
                    newind->gender = 'F';
                input >> newind->ID;
                input >> newind->PID;
                newind->children = new individual*[MAX_CHILDREN]; // cocuklari gosteren isaretci icin yer alinir
                for(int i = 0; i < MAX_CHILDREN; i++)
                    newind->children[i] = NULL; // cocuklar NULL a esitlenir
                newind->wife = new spouse; // esi gosteren pointer icin bellekten yer alinir
                newind->wife->name = ""; // eslerin isim kontorlunde sorun cikmamasi icin bos karakter atanir
                if(root == NULL){
                    root = newind; // root eklenir
                    continue; // dongu basa gelir
                }

                person::findpersonrec(root, newind->PID, false); // cocugu aileye baglamak icin baglanilacak aile bulunur
                individual *parent = person::found; // person isim uzayindaki findpersonrec() fonksiyonunun sonucu atanir
```

The program compiled without warning or error:



Finally the program is executed. Sample outcome is below:

```
C:\Windows\system32\cmd.exe
TREE FAMILY PROGRAM
BLG 233E Data Structures and Laboratory (11146)
Assignment #3
TUGRUL YATAGAN
040100117

Options:
        (1) Print Parent
        (2) Print Children
        (3) Print Cousins
        (4) Print Grand Children
        (5) Print Grand Parents
        (6) Print Siblings
        (7) Check Sibling
        (8) Print Family
        (0) Exit
Enter Options: 8

Ali-Fatma
Berna-Tacettin  Cem-Sinem  Deniz-Ece
Ceyda-Ibrahim  Ahmet  Hasan-Gozde  Omer-Hatice  Aysegul-Recep  Ziya-Burcu
Mustafa  Emre-Bahar  Zafer  Can  Burak  Asli  Sibel
Meral  Neval

Options:
        (1) Print Parent
        (2) Print Children
        (3) Print Cousins
        (4) Print Grand Children
        (5) Print Grand Parents
        (6) Print Siblings
        (7) Check Sibling
        (8) Print Family
        (0) Exit
Enter Options: 1

Enter name: Hasan
Hasan's parent are: Cem and Sinem

Options:
        (1) Print Parent
        (2) Print Children
        (3) Print Cousins
        (4) Print Grand Children
        (5) Print Grand Parents
        (6) Print Siblings
        (7) Check Sibling
        (8) Print Family
        (0) Exit
Enter Options:
```

3. Data Structures and Variables

Each individual have a name (char*), year of birth (int), gender (char), pointer array to children (node**) pointer to wife/husband (spouse*), node ID (int), and parent ID (int). The spouse type has two data types: name (char*) and year of birth (int).

```
struct spouse{
      char *name;
      int birth;
};
```

```
struct individual{
      char *name;
      int birth;
      char gender;
      individual **children;
      spouse *wife;
      int ID;
      int PID;
};
```

- <u>createFamily():</u> This function creates the tree from input text file. Individual and spouse structs was added according to relatives when the program starts.

- <u>printParent(individual *traverse, char *search, bool stop):</u> This function prints the names of the parents (both mother and father) for the given parameter.

- <u>printChildren(individual *traverse, char *search, bool stop):</u> This function prints the names of the children for the given parameter.

- <u>printCousins(individual *traverse, char *search, bool stop):</u> This function should print the names of the cousins for the given parameter. Here, cousins are only the people who have the same grandparents.

- <u>printGrandchildren(individual *traverse, char *search, bool stop):</u> This function should print the names of the grandchildren for the given parameter.

- <u>printGrandparents(individual *traverse, char *search, bool stop):</u> This function should print the names of the grandparents for the given parameter.

- <u>printSiblings(individual *traverse, char *search, bool stop):</u> This function prints the names of the siblings for the given parameter.

- <u>isSibling(char *ptr1, char *ptr2):</u> This function returns true if the two parameters have the same parents.

-<u>printFamily():</u> This function prints all the tree layer by layer.

-<u>closeFamily(individual *traverse):</u> This function deletes the tree.

```
namespace person{
      bool findpersonrec(individual *, int, bool);
      individual *found;
}
namespace sibling{
      bool checkParent(individual *,char *, bool);
      individual *result;
}
namespace level{
      int findLayerNum(individual *, int);
      int printLayer(individual *, int, int);
      int max_layer;
}
```

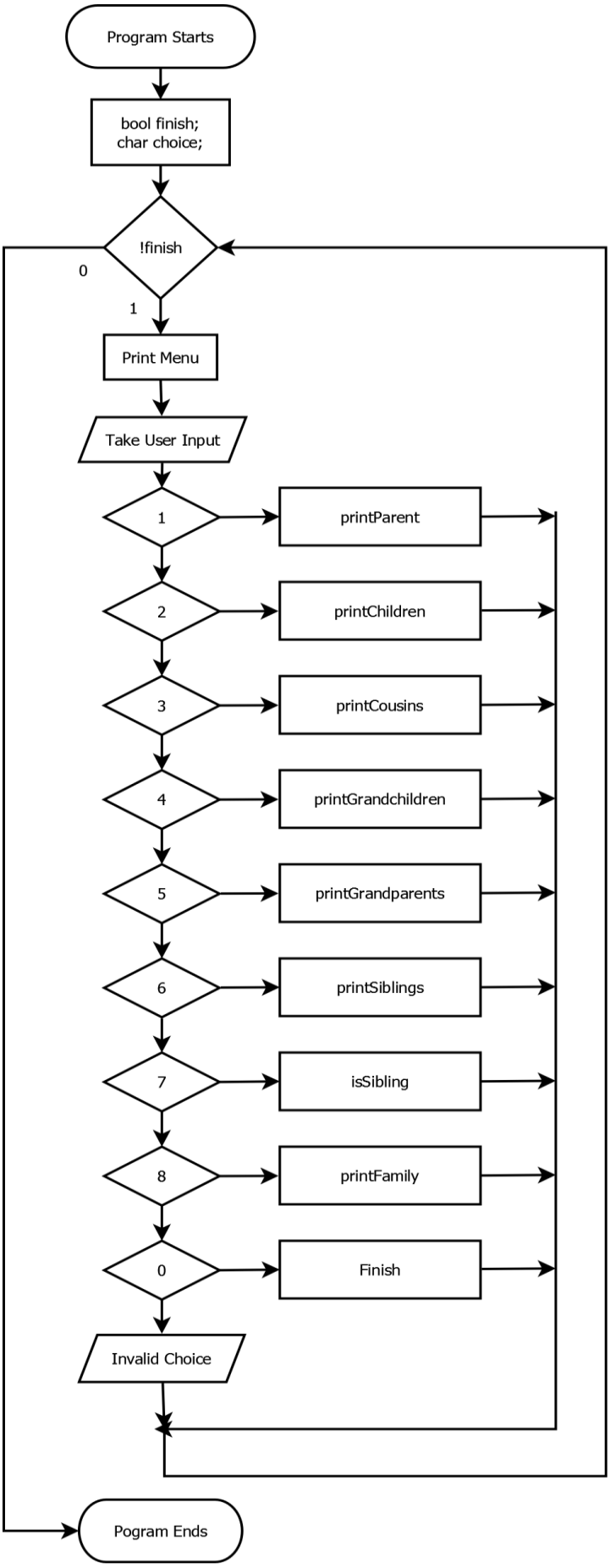
-<u>findpersonrec(individual *, int, bool):</u> This function finds the persons position for creating the tree. Return value is stored in individual *found variable.

<u>checkParent(individual *,char *, bool):</u> This function finds the parents of parameter for sibling check. Return value is stored in individual *result variable.

<u>findLayerNum(individual *, int):</u> This function counts the depth of the tree (total number of layer) for printing the tree. Return value is stored in int max_layer variable.

<u>printLayer(individual *, int, int):</u> This function prints wanted layer of the tree.

6. Program Flow



5. Conclusion

In this homework, I have become more familiar with the concept of data structures, trees and recursive functions. I had the chance to intensify my knowledge about their structures.