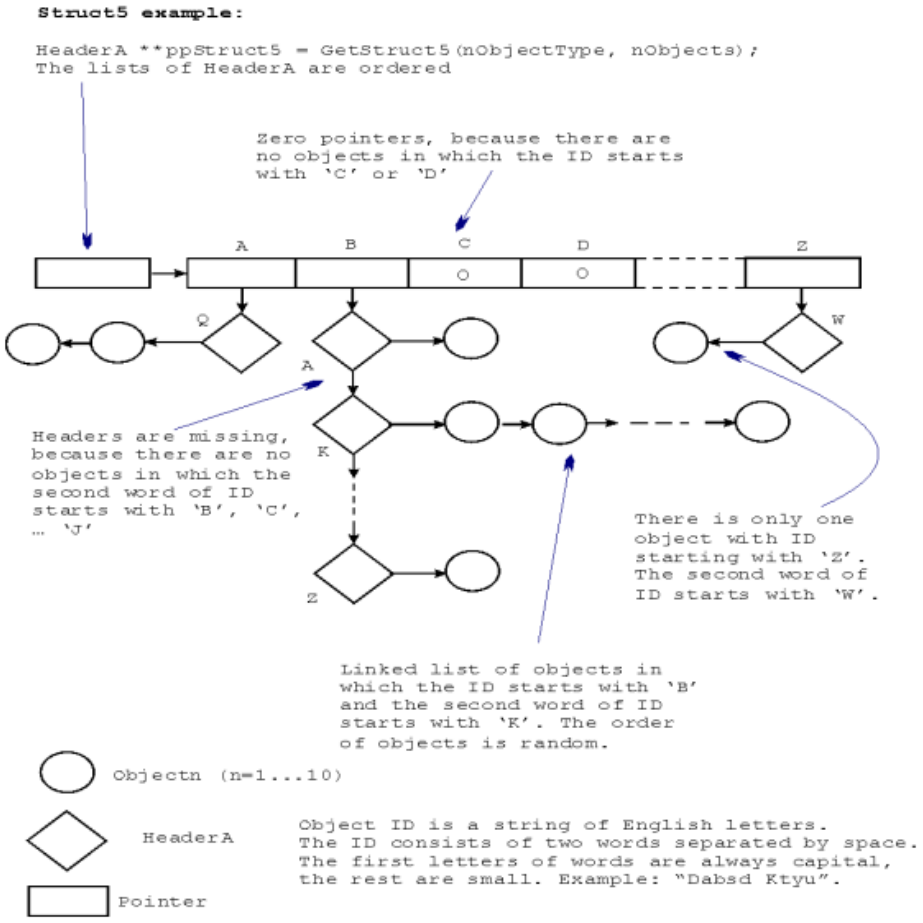
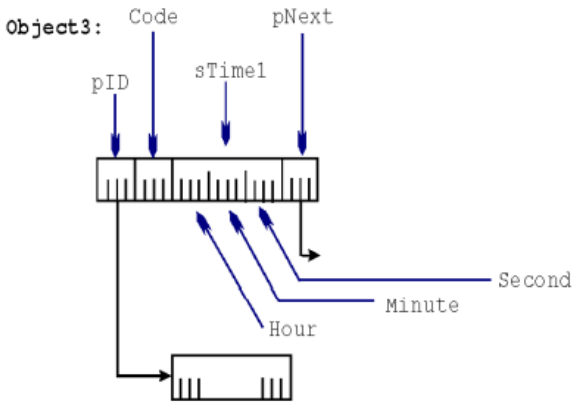
5.1. Source data

The source data are the objects connected in linked lists, which are connected by the HeaderA linked lists and vector of pointers to form a complete data structure. Objects are of type Object3. The declarations of Object3 and C struct are given in the *Objects.h* file. The HeaderA declaration can be found in the file *Headers.h*. An example of the source structure is shown in the figure below: (**files have multiple Objects, Structs... My work is with Struct5 and Object3)**





In all cases, the object ID is a C string consisting of two words.

Both words consist of lower case letters of the English alphabet, but the first character of the word is an upper case letter of the English alphabet. There is one space between the words. The lengths of the words can be can be any number greater than zero.
The order of the objects in the linked list is random.

The position of each pointer vector corresponds to one the first letter of the first word of the ID. If there are no objects associated with the first letter of a first word, the pointer vector in that position points to 0 (null).

Each HeaderA type of binder corresponds to one of the first letters of the second word of a specific ID. For example, objects with ID's of „Smith John", „Simpson Jack" and „Sanders Jim" are all located in the chain of objects that come out of the HeaderA type associated with the letter 'J'. and this binder in turn is located in the chain from the letter 'S' to the corresponding position of the pointer vector.

The links in the HeaderA chain are in alphabetical order. It is important to underline, that if there are no objects associated with the first letter of another word, there is no corresponding binder.

The source structure is generated by the GetStruct5 function (located in the in the object module Structs.obj, prototype is in the file Structs.h.).

5.2. Task

The task is to write the following 6 functions:

5.2.1. First function

*void PrintObjects(HeaderA **ppStruct5);*

The input parameter is a pointer to the source struct vector of pointers. The aim of the function is to output the descriptions of all objects in the structure (each object on separate line). The output function is *printf*. The output formatting string (*printf* first parameter) is given in the *Objects.h* file.

This function must be written first, because without it it is not possible to test the other functions.

5.2.2. Second function

*int InsertNewObject(HeaderA **ppStruct5, char *pNewID, int NewCode);*

The input parameter is a pointer to the source struct vector of pointers; the pointer to new object ID and NewCode of new object. The aim of the function is to create a new object and insert it into the data structure.

The function first checks whether the new ID matches the right format (described before). If this condition is not met, the function returns a 0. Next the function must check whether an object with such ID already exists. If it does, the function returns a 0. If there is no object with such ID, it must be created and inserted into the correct linked list. All required memory must be created using the *malloc* function. Also the ID needs its own memory field, where the input string must be copied to. The time or date must be read from the clock on the computer and converted (corresponding functions are in *DateTime.obj*, prototypes in *DateTime.h*). The location of the new object in its linked list is irrelevant. If the function completed its task, it outputs a value of 1.

It may be that the objects with the second letter of the given new ID did not exist before and therefore there is no corresponding binder. In such case, the missing binder must first be created and placed in the position foreseen in the binder linked list.

When testing the function, be sure to test the following situations:
1. The new object must be placed in an existing linked list.
2. There were no objects with such first letter before, so HeaderA chain does not yet exist.
3. There were objects with this first letter of first word before, but not with first letter of second word, so there is no necessary HeaderA binder:
   a. The new binder comes first in the HeaderA chain list.
   b. The new binder comes last in the HeaderA chain list.
   c. The new binder comes somewhere in the middle of the HeaderA chain list.
4. The new object ID is in the wrong format.
5. The ID of the new object is the same as that of an existing object.

5.2.3. Third function

Object3 RemoveExistingObject(HeaderA **ppStruct5, char *pExistingID);*

The input parameter is a pointer to a pointer, that points to the first existing binder of type HeaderC; a pointer to a string that matches an existing ID. The function's job is to find this object and remove it from the data structure. A removed object must not be deleted. The output is a pointer pointing to the removed object. If no object was found, the output will be 0(null).

It may be that the object to be removed is the only one in it's linked list. In such case, the HeaderA type binder corresponding to the initial letter of the object must also be deleted after removing the object. If no binders remain after this, write zero to the corresponding position of vector of pointers.

When testing the function, be sure to test the following situations:

1.The object to be removed is located in a linked list with more than one link:
   a. The object to be removed is first in the linked list
   b. The object to be removed is last in the linked list
   c. The object to be removed is somewhere in the middle of the linked list
2. The object to be removed is the only one in its linked list, but there are more objects with the same first letter of the first word, so HeaderA type of chain binder remains.
3. The object to be removed is the only one in its linked list, and there are no more objects with the same first letter of the first word, so HeaderA type of chain binder gets removed.
4. There is no object with given input ID.

5.2.4. Fourth function

*Node *CreateBinaryTree(HeaderA **ppStruct5);*

The input parameter is a pointer to the source structure vector of pointers. The aim of the function is to create a binary search tree, whose nodes point to the strings in the source struct. The strings are *Code* of Object3. The output parameter is a pointer to node of tree root. The C struct representing the node of tree is found in *Headers.h*.

Function must be non-recursive. Refer to following image

Kirje lisamine kahendpuusse

```
NODE *InsertNode(NODE *pTree, void *pNewRecord,
                 int(*pCompare)(const void *, const void *)) {
 NODE *pNew = (NODE *)malloc(sizeof(NODE));// uus tipp
 pNew->pRecord = pNewRecord;
 pNew->pLeft = pNew->pRight = 0;
 if(!pTree)
   return pNew;// puu oli tühi
 for(NODE *p = pTree; 1; ) {
   if ((pCompare)(pNewRecord, p->pRecord) < 0) {
      if (!p->pLeft) {// leidsime tühja koha
         p->pLeft = pNew;
         return Tree;
      }
      else
         p = p->pLeft;// liigume vasakule
   }
   else {
      if(!p->pRight) { // leidsime tühja koha
         p->pRight = pNew;
         return pTree;
      }
      else
         p = p->pRight;// liigume paremale
      }
   }
}
```

5.2.5. Fifth function

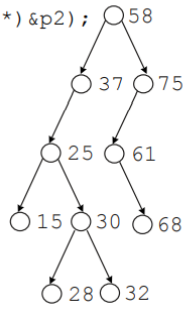
*void TreeTraversal(Node *pTree);*

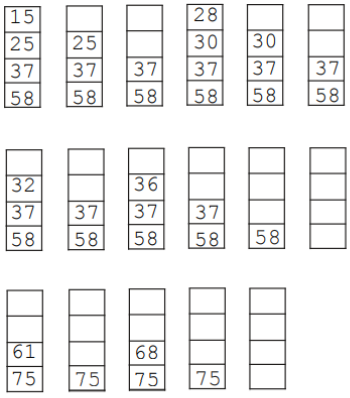
The input parameter is a pointer to the tree created in the fourth function. The aim of the function is to traverse all the tree nodes using „left-root-right(symmetrical)“ method and print out the object belonging to it (like in the first function).

Function must be non-recursive. Refer to the following image.

Puu läbikäik (3)

```
void SymmetricalTraversal(NODE *pTree,void(*pProcess)(NODE *))
{
 STACK *pStack = 0;
 NODE *p1 = pTree, *p2;
 if (!pTree)
   return;
 do
 {
   while(p1)
   {
      pStack = Push(pStack, p1);
      p1 = p1->pLeft;
   }
   pStack = Pop(pStack, (void**)&p2);
   (pProcess)(p2);
   p1 = p2->pRight;
 }
 while(!(!pStack && !p1));
}
```

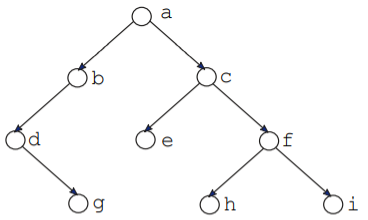


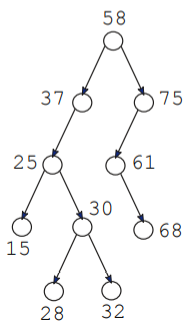


Rekursiivne algoritm on realiseeritud stack-i kasutava mitterekursiivse funktsiooniga

Left-root-right method:

1) d, g, b, a, e, c, h, f, i
2) 15, 25, 28, 30, 32, 37, 58, 61, 68, 75





Nivooti: 58
Süviti: 15
(vasak - pa
Laiuti: 58
(juur - vas
Sümmeetrili
(juur - vas

Sümmeetrili

5.2.6. Sixth function

*Node *DeleteTreeNode(Node *pTree, unsigned long int Code);*

The input parameter is a pointer to the tree created in the fourth function and a possible Code. The aim of the function is to delete a tree node, which points to the object including Code. The output parameter is a pointer to the node after removal.

Function must be non-recursive.

When testing the function, be sure to try the following situations:
1. The removable node is the root of the tree.
2. The node to be removed does not branch out (have a daughter node).
3. The node to be removed has only a right-side daughter node.
4. The node to be removed has only a left-side daughter node.
5. The node to be removed has both daughter nodes.
6. There is no item with inputted Code.

Tests

First 3 functions
1. N = 35.
2. Output the source structure
3. Add objects with ID's in following order: Dx Gz, Dx Ga, Db Aa, Dk Za, Dr Wa, Aa Aa, Ab Ba, Za Aa, Za Ab, Za Ba, Wx Xa, Wx Aa, zb Kk, Zc ca, Dr Wa, ZB kk, Fa, Fa_Fa
   and print the result. 'Code' value can be a random integer.
   Note: adding 6 last objects must create an error
4. Remove objects in the same order and output the changed structure
   Note: Removing 6 last objects must create an error

Last 3 functions
1. N = 35
2. Output the source structure
3. Create a binary tree and traverse all its nodes, printing out the objects belonging to node.
4. Remove the root node of the tree and by traversing all its nodes, print each object.
5. N = 10
6. Output the source structure
7. Create a binary tree and traverse all its nodes, printing out the objects belonging to node.
8. Make visual graph of tree (using paint, paper or whatever method), writing their key(code) by each node
9. Remove 3 nodes from tree (own choice) and print out the tree.
10. Attempt to remove a non-existing node

Start code

```
#include "stdio.h"
#include "DateTime.h"
#include "Objects.h"
#include "Headers.h"
#include "Structs.h"
#pragma warning ( disable : 4996 )

void PrintObjects(HeaderA**);
int InsertNewObject(HeaderA**, char*, int);
Object3* RemoveExistingObject(HeaderA**, char*);
Node *CreateBinaryTree(HeaderA **ppStruct5);
void TreeTraversal(Node *pTree);
Node *DeleteTreeNode(Node *pTree, unsigned long int Code);

int main(){
  HeaderA **pStruct5 = GetStruct5(3, 35);
  return 0;
}
```

Visual Studio guide

1. Start Visual Studio
2. Create new C++ Windows Console Empty Project
3. In **Solution Explorer** right click on project name, **Add -> New item.** Click new C++ file and name it *Main*.
4. Input following to Main.cpp

```
#include "stdio.h"
#pragma warning ( disable : 4996 )
int main()
{
printf("First run\n");
return 0;
}
```

5. **Build -> Build solution**
6. **Debug -> Start debugging**
7. **.h** files to . . . . . \ *PROJECT_NAME* \ *PROJECT_NAME*
   .obj files to . . . . \ *PROJECT_NAME* \ *PROJECT_NAME* \ *Debug*
8. In **Solution Explorer** right click **Header files**, **Add -> Existing item,** *DateTime.h, Headers.h, Objects.h, Structs.h*
9. In **Solution Explorer** right click project name, **Add -> Existing item,** *DateTime.obj, Objects.obj, Structs.obj*
10. Input **Start Code** and make sure **Build -> Build solution** works.