```c
////////////////////////////////////////////////////////////////////////////
//     Projeto 2 IAED                                  Joao Bernardo 86443     //
//     Grupo 88                                        Pedro Antunes 86493     //
////////////////////////////////////////////////////////////////////////////

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define Key(A) (A->chave)
#define Key2(C) (C->unidades)
#define eq(A, B) (strcmp(A, B) == 0)
#define less(A, B) (strcmp(A, B) < 0)
#define CHAVE 8

typedef char* Key;

typedef struct produto {
    char chave[CHAVE+1];
    int unidades;
}* Produto;

////////////////////////////////////////////////////////////////////////////
// Utilizacao de arvores binarias para organizar as informacoes dos produtos //
////////////////////////////////////////////////////////////////////////////

#define Item Produto

typedef struct STnode* link;

struct STnode{
    Item item;
    link l, r;
};

static link head;
static int flag_del;    // toma o valor 1 quando o maximo foi eliminado ou subtraido algo ao seu valor
static int maximo=0;    // variavel que guarda as unidades do maximo
static char max_chave[CHAVE+1]="ffffffff";      // variavel que guarda a chave associada ao maximo


Item newItem(char ch[CHAVE+1], int uni){
    Item x = (Item)malloc(sizeof(struct produto));
    strcpy(x->chave, ch);
    x->unidades = uni;
    return x;
}

void deleteItem(Item a){
    free(a);
}

void visitItem(Item a) {
    printf("%s %d\n",a->chave,a->unidades);
}


link NEW(Item item, link l, link r){
    // Cria um novo no na arvore para Item
    link x = (link)malloc(sizeof(struct STnode));
    x -> item = item;
    x -> l = l;
    x -> r = r;
    // Devolve um ponteiro para este no
    return x;
}


void STinit(link*head){
    // Inicia a arvore
    *head = NULL;
}


link insertR(link h, Item item){
    // Insere o Item de forma ordenada na arvore, com base na sua referencia (Key)
```

```
77      if (h == NULL)
78          return NEW(item, NULL, NULL);
79      if (less(Key(item), Key(h->item)))
80          h->l = insertR(h->l, item);
81      else
82          h->r = insertR(h->r, item);
83      // Devolve a raiz da arvore atualizada
84      return h;
85  }
86
87
88  void STinsert(link*head, Item item){
89      *head = insertR(*head, item);
90  }
91
92
93  Item searchR(link h, Key v){
94      // Procura na arvore de raiz 'h' o no com um Item de referencia v, devolve
95      // um ponteiro para esse no, caso o Item exista e NULL em caso contrario
96      if (h == NULL)
97          return NULL;
98      if (eq(v, Key(h->item)))
99          return h->item;
100     if (less(v, Key(h->item)))
101         return searchR(h->l, v);
102     else
103         return searchR(h->r, v);
104 }
105
106
107 Item STsearch(link head, Key v){
108     return searchR(head, v);
109 }
110
111
112 void atualiza_maximo(char ch[CHAVE+1], int uni){
113     if (uni > maximo){       // se uni for maior que o maximo
114         maximo = uni;            //maximo passa a ser uni
115         strcpy(max_chave,ch);   //e max_chave passa a ser chave
116     }
117     else if(uni == maximo && less(ch, max_chave)){      // se forem iguais e a chave for
    lexicograficamente menor que max_chave
118         strcpy(max_chave,ch);   // max_chave assume o valor de chave
119     }
120     else
121         return;
122 }
123
124 link max(link h){
125     if (h==NULL || h->r==NULL)
126         return h;
127     else
128         return max(h->r);
129 }
130
131
132 link deleteR(link h, Key k){
133     if (h==NULL)
134             return h;
135     else if (less(k, Key(h->item)))
136         h->l=deleteR(h->l,k);
137     else if (less(Key(h->item), k))
138         h->r=deleteR(h->r,k) ;
139     else{
140         if (h->l !=NULL && h->r !=NULL){
141         link aux = max(h->l);
142         {Item x; x=h->item; h->item=aux->item; aux->item=x;}
143         h->l= deleteR(h->l, Key(aux->item));
144         }
145         else {
146             link aux = h;
147             if (h->l == NULL && h->r == NULL)
148                 h=NULL;
149             else if (h->l==NULL)
150                 h=h->r;
151             else
152                 h=h->l;
153             deleteItem(aux->item);
```

```
            free(aux);
        }
    }
    return h;
}

void STdelete(link*head, Key k){
    *head = deleteR(*head, k);
}



int count(link h){
    if (h==NULL)
        return 0;
    else
        return count(h->r)+count(h->l)+1;
}


int STcount(link head){
    return count(head);
}


void sortR(link h, void (*visit)(Item)){
    if (h == NULL)
        return;
    sortR(h->l, visit);
    visit(h->item);
    sortR(h->r, visit);
}


void STsort(link head, void (*visit)(Item)){
    sortR(head, visit);
}


link freeR(link h){
    if (h==NULL)
        return h;
    h->l=freeR(h->l);
    h->r=freeR(h->r);
    return deleteR(h,Key(h->item));
}


void STfree(link*head){
    *head=freeR(*head);
}


void traverse(link h){
    // visita a raiz depois do filho esquerdo e antes do direito
    if (h == NULL)
        return;
    traverse(h->l);
    atualiza_maximo(Key(h->item), Key2(h->item));   // verifica se houve alteracao no maximo
    traverse(h->r);
}


////////////////////////////////////////////////////////////////////////////
//                                    Comandos                            //
////////////////////////////////////////////////////////////////////////////


void funcao_a(){
    char ch[CHAVE+1];
    int uni;
    getchar();
    scanf("%s", ch);
    getchar();
    scanf("%d", &uni);
    if (STsearch(head,ch) == NULL){     // se a chave nao existir na arvore, criamos o produto
        if (uni < 0)    // se o valor for negativo, criamos o produto com 0 unidades
            uni = 0;
```

```
232             atualiza_maximo(ch,uni);
233             STinsert(&head, newItem(ch, uni));
234         }
235         else if (uni < 0){
236             int a = Key2(STsearch(head,ch));
237             if ((a + (uni)) <= 0)
238                 Key2(STsearch(head,ch)) = 0;
239             else
240                 Key2(STsearch(head,ch)) = a + uni;
241             if (eq(max_chave, Key(STsearch(head,ch))))
242                 flag_del = 1;       //se diminuirmos o maximo, colocamos a flag a 1
243         }
244 
245         else{
246             int b = Key2(STsearch(head,ch));
247             Key2(STsearch(head,ch)) = b + uni;
248             atualiza_maximo(ch,b+uni);
249         }
250     }
251 
252 
253     void funcao_l(){
254         if (count(head) == 0)       //verifica se a arvore esta vazia
255             return;
256         else                            //lista todas as chaves e unidades
257             STsort(head,visitItem);     //associadas aos produtos existentes
258     }
259 
260     void funcao_m(){
261         if (count(head) == 0)   //verifica se a arvore esta vazia
262             return;
263         else if (flag_del == 1){        // caso a chave associada ao maximo tenha sido apagada
264             maximo=0;
265             strcpy(max_chave,"ffffffff");
266             traverse(head);
267             printf("%s %d\n",max_chave,maximo);
268             flag_del = 0;       // voltamos a colocar a flag a 0
269         }
270         else
271             printf("%s %d\n",max_chave,maximo);
272     }
273 
274 
275     void funcao_r(){
276         char ch[CHAVE+1];
277         getchar();
278         scanf("%s", ch);
279         if(STsearch(head,ch) == NULL)
280             return;
281         else {
282             if (eq(max_chave,ch))   // caso a chave a apagar seja a do maximo
283                 flag_del = 1;   // indicamos que o maximo foi apagado
284             STdelete(&head,ch);
285         }
286     }
287 
288     void funcao_x(){
289         printf("%d\n",count(head)); // exibe o numero de produtos na arvore
290         STfree(&head);
291     }
292 
293 
294 
295     ////////////////////////////////////////////////////////////////////////////
296     //                                  Main                                  //
297     ////////////////////////////////////////////////////////////////////////////
298 
299     int main(){
300         char op;
301         STinit(&head);
302         while((op=getchar())!='x'){
303             switch(op){
304                 case 'a':
305                 funcao_a();
306                 break;
307                 case 'l':
308                 funcao_l();
309                 break;
```

```
            case 'm':
            funcao_m();
            break;
            case 'r':
            funcao_r();
            break;
        }
    }
    funcao_x();
    return 0;
}
```