========== Plist.h ==========

```
#ifndef _PLIST_
#define _PLIST_

#include <stdlib.h>
#include "produs.h"

class Nod {
            Produs *data;
            Nod  *next;

            Nod(Produs *d, Nod *p) {
              data = d; next = p;
            }

            friend class PList;
            friend class PLIterator;
};

class Plist {
            Nod *head;
            Nod *tail;
       public:
            PList() {
              head = tail = NULL;
            };

            PList(PList&);
            ~PList();

            PList& operator= (PList&);

            void add(Produs*);
            int  sterge(Produs*);
            Produs* search(char* cod);
            void clear();
            void sort(int);

            friend class PLIterator;
};

class PLIterator {
            PList* l;
            Nod* curent;
      public:
            PLIterator(PList *l) {
              this->l = l;
              curent = l->head;
            }

            PLIterator(PLIterator& pli) {
              l = pli.l;
              curent = pli.curent;
            }

            int hasMoreElements();
            Produs* getCurent();
            void operator= (PLIterator&);
            void* operator++ (int);
            void reset();
};
```

```
#endif
```

========== Plist.cpp ==========

```
#include "plist.h"

PList::PList(PList& l) {
  Nod *p = l.head;
  head = new Nod(p->data, NULL);
  tail = head;

  while (p) {
    head = new Nod(p->data, head);
    p = p->next;
  }
}

PList::~PList() {
  clear();
}

PList& PList::operator= (PList& l) {
  if (this == &l)
    return *this;

  clear();
  Nod *p = l.head;
  head = new Nod(p->data, NULL);
  tail = head;
  while (p) {
    head = new Nod(p->data, head);
    p = p->next;
  }
  return *this;
}

void PList::add(Produs *p) {
  if (!head) {
    head = new Nod(p, head);
    tail = head;
  }
  else {
    tail->next = new Nod(p, NULL);
    tail = tail->next;
  }
}

int PList::sterge(Produs *d) {
  Nod *p, *q;

  q = NULL; p = head;
  while (p && (strcmp(p->data->getCod(), d->getCod()) != 0)) {
    q = p;
    p = p->next;
  }
  if (p) {
    if (!q) {
      p = head;
      if (tail == head)
        tail = NULL;
      head = head->next;
```

```
      delete p;
    }
    else {
      q->next = p->next;
      if (tail == p)
      tail = q;
      delete p;
    }
    return 1;
  }
  return 0;
}

Produs* PList::search(char *cod) {
  Nod *p = head;

  while (p && (strcmp(p->data->getCod(),cod) != 0))
    p = p->next;

  if (p == NULL)
    return NULL;
  else
    return p->data;
}

void PList::clear() {
  Nod *p;

  while (head) {
    p = head;
    head = head->next;
    delete p;
  }
  tail = NULL;
}

void PList::sort(int type) {
  Nod *p;
  Nod *q;
  Produs *tmp;

  p = head;
  while (p->next) {
    q = p->next;
    while (q) {
      if (cmp(type, *p->data, *q->data) > 0) {
        tmp = p->data; p->data = q->data; q->data = tmp;
      }
      q = q->next;
    }
    p = p->next;
  }
}

/* ********************************** */

int PLIterator::hasMoreElements() {
 if (curent == NULL)
    return 0;
  else
    return 1;
```

```
}

Produs* PLIterator::getCurent() {
  Produs* p = curent->data;
  //curent = curent -> next;
  return p;
}

void PLIterator::operator= (PLIterator& pli) {
  if (this == &pli)
    return;

  l = pli.l;
  curent = pli.curent;
}

void* PLIterator::operator++ (int) {
  curent = curent->next;
  return NULL;
}

void PLIterator::reset() {
  curent = l->head;
}
```

========== Produs.h ==========

```
#ifndef _PRODUS_
#define _PRODUS_

#include <iostream.h>
#include <string.h>

class Produs {
            char* denumire;
            char* cod;
            int  nrBucati;
            long pret;
      public:
            Produs(char* , char*, int, long);
            ~Produs();

            char* getCod();
            int modificNrBucati(int );
            int getNrBucati();
            long getPret();

            friend ostream& operator << (ostream&, Produs&);
            friend int cmp(int, Produs&, Produs&);

            static int CANTITATE;
            static int DENUMIRE;
            static int COD;
};

#endif
```

========== Produs.cpp =========

```
#include "produs.h"

int Produs::CANTITATE = 1;
int Produs::DENUMIRE = 2;
```

```
int Produs::COD = 3;

Produs::Produs(char* _denumire, char* _cod, int _nrBucati, long _pret) {
  denumire = new char [strlen(_denumire) + 1];
  strcpy(denumire, _denumire);

  cod = new char [strlen(_cod) + 1];
  strcpy(cod, _cod);

  nrBucati = _nrBucati;
  pret = _pret;
}

Produs::~Produs() {
  delete[] denumire;
  delete[] cod;
}

int Produs::modificNrBucati(int cantitate) {
  if (nrBucati + cantitate < 0)
     return -1;
  else
     nrBucati += cantitate;

  return nrBucati;
}

char* Produs::getCod() {
  return cod;
}

int Produs::getNrBucati() {
  return nrBucati;
}

long Produs::getPret() {
  return pret;
}

ostream& operator<< (ostream& out, Produs& p) {
  out << p.denumire << endl;
  out << p.cod << endl;
  out << p.nrBucati << endl;
  out << p.pret << endl;

  return out;
}

int cmp(int type, Produs& p1, Produs& p2) {
  if (type == Produs::CANTITATE)
    return (p1.nrBucati - p2.nrBucati);
  else
    if (type == Produs::DENUMIRE)
      return strcmp(p1.denumire, p2.denumire);
    else
      if (type == Produs::COD)
        return strcmp(p1.cod, p2.cod);

  return 0;
}
```

```
========== Depozit.h ==========
#ifndef _DEPOZIT_
#define _DEPOZIT_

#define MAX 80
#include "plist.h"

class Depozit {
            char* nume;
            char* adresa;
            PList* produse;
      public:
            Depozit();
            Depozit(char*, char*);
            ~Depozit();

            char* getNume();
            char* getAdresa();
            PList* getProduse();

            void intrare(char*, char*, int, long);
            int iesire(char*, int);
            int getCantitateProdus(char* cod);
            int eliminProdus(char* cod);
            Produs* getProdusCantitateMaxima();
            void afisareSelectiva(int);

            void listare();
            friend ostream& operator<< (ostream&, Depozit&);
            friend istream& operator>> (istream&, Depozit&);
};

#endif

========== Depozit.cpp ========
#include "depozit.h"

Depozit::Depozit() {
  nume = NULL;
  adresa = NULL;
  produse = new PList();
}

Depozit::Depozit(char* _nume, char* _adresa) {
  nume = new char [strlen(_nume) + 1];
  strcpy(nume, _nume);

  adresa = new char [strlen(_adresa) + 1];
  strcpy(adresa, _adresa);

  produse = new PList();
}

Depozit::~Depozit() {
  delete[] nume;
  delete[] adresa;
  delete produse;
}

char* Depozit::getNume() {
  return nume;
```

```
}

char* Depozit::getAdresa() {
  return adresa;
}

PList* Depozit::getProduse() {
  return produse;
}

void Depozit::intrare(char* nume, char* cod, int cantitate, long pret) {
  Produs* p = produse -> search(cod);

  if (p == NULL) {
    p = new Produs(nume, cod, cantitate, pret);
    produse -> add(p);
  }
  else
    p -> modificNrBucati(cantitate);
}

int Depozit::iesire(char* cod, int cantitate) {
  Produs* p = produse -> search(cod);

  if (p == NULL)
    return -1;

  int value = p -> modificNrBucati(-cantitate);
  if (value == 0) {
    produse -> sterge(p);
    delete p;
  }
  return value;
}

int Depozit::getCantitateProdus(char* cod) {
  Produs* p = produse->search(cod);

  if (p == NULL)
    return -1;
  else
    return p->getNrBucati();
}

int Depozit::eliminProdus(char* cod) {
  Produs* p = produse->search(cod);

  if (p != NULL) {
    produse->sterge(p);
    return 0;
  }
  return -1;
}

Produs* Depozit::getProdusCantitateMaxima() {
  PLIterator* pli = new PLIterator(produse);
  int maxValue = 0;
  Produs* p = NULL;

  while (pli->hasMoreElements()) {
    if (maxValue < pli->getCurent()->getNrBucati()) {
```

```
      p = pli->getCurent();
      maxValue = p->getNrBucati();
    }
    (*pli)++;
  }

  return p;
}

void Depozit::afisareSelectiva(int limit) {
  PLIterator* pli = new PLIterator(produse);
  Produs* p = NULL;

  while (pli->hasMoreElements()) {
    p = pli->getCurent();
    if (limit <= p->getNrBucati())
      cout << (*p);
    (*pli)++;
  }
}

ostream& operator<< (ostream& out, Depozit& d) {
  out << d.nume << endl;
  out << d.adresa << endl;

  PLIterator* pli = new PLIterator(d.produse);
  while (pli->hasMoreElements()) {
    out << (*pli->getCurent());
    (*pli)++;
  }
  return out;
}

istream& operator>> (istream& in, Depozit& d) {
  char bufferA[MAX];
  char bufferB[MAX];

  in >> bufferA;
  d.nume = new char [strlen(bufferA) + 1];
  strcpy(d.nume, bufferA);

  in >> bufferB;
  d.adresa = new char [strlen(bufferB) + 1];
  strcpy(d.adresa, bufferB);

  while (in >> bufferA) {
    int cantitate;
    long pret;

    if (strcmp(bufferA, "") == 0)
      break;
    in >> bufferB;
    in >> cantitate;
    in >> pret;

    d.intrare(bufferA, bufferB, cantitate, pret);
  }

  return in;
}
```

```
========== Main.cpp ==========
#include <fstream.h>

#include "depozit.h"
#include "produs.h"

void citire(Depozit* dep) {
    cin >> (*dep);
}

void main(void) {
  Depozit* dep = new Depozit("Malxorul", "Str_Unirii_Nr32");

  citire(dep);

  ofstream fout("depozit.txt");
  fout << (*dep);
  fout.flush();
  fout.close();
}
```

Bibliografie

http://inf.ucv.ro/~mirel/courses/poo/poo.html

1. *The C++ Programming Language*, B. Stroustrup, Addison-Wesley. (Cap. 21 Streams)

2. *C++ Coding Standard*, Todd Hoff
http://www.possibility.com/Cpp/CppCodingStandard.html