```
//
// ///////////////////////////////////////////////////////////////////////////
//
// >>> analyse.cpp <<<
//
// Beispielprogramm zum Einlesen von Daten und Fuellen eines Histogramms
//
// *********************************************************
//   Versuch Z0,
//   Fortgeschrittenenpraktikum Humboldt-Universitaet zu Berlin  *
//
// *********************************************************
//
// T. Hebbeker  (RWTH Aachen) Vers.  2.0  2004-10-07
//                                   2.1  2006-03-16   (V.Vorwerk: fin.open)
//
// M. zur Nedden (HU Berlin)  Vers.  3.0  2007-02-02
//
// Allgemeine Bemerkungen:
//
//   C++ Kenntnisse werden vorausgesetzt.
//
//   Die Routinen TFile, TH1F, Fill, Write sind Teile aus der Root-Bibliothek
//   und dienen zur Erstellung von Histogrammen
//
// Erlaeuterungen zu diesem Programm:
//
//   - Class cevent dient zur Speicherung eines e+e- - Kollisions-
//       ereignisses und zur Extraktion der Impulse und Massen der
//       zugehoerigen Teilchen. Bitte nicht veraendern!
//
//   - Die am Ende angehaengte subroutine read_event zum Einlesen
//       eines Ereignisses kann als black box betrachtet werden.
//
//   - Der Histogramm-file kann mit dem (graphischen) Analyseprogramm
//       ROOT eingelesen und weiterverarbeitet werden
//       siehe dazu das ROOT-Beispiel-Script plot.C
//
//   - Namensgebung fuer loop-Variable/Indices:
//      n...  = event counter
//      k...  = "particle" counter
//      l     = momentum component x, y, z
//
// ///////////////////////////////////////////////////////////////////////////
//

// general header files:
#include <iostream.h>
#include <string.h>
#include <math.h>
#include <fstream.h>

// needed for ROOT routines:
#include "TROOT.h"
#include "TFile.h"
#include "TH1.h"
#include "TF1.h"

//
/////////////////////////////////////////////////////////////////////////////
//
// class cevent enthaelt e+e- Daten (jeweils ein Ereignis):
```



```
//       ======
//
//  _ktotce = Zahl der Teilchen/Kalorimetertreffer ("particle") im Detektor
//
//   Fuer jedes Teilchen/Treffer k (1 ... _ktotce) ist gespeichert:
//
//  _pvecce(k,1) = Impulskomponente in x-Richtung (in LEP-Ebene)
//  _pvecce(k,2) = Impulskomponente in y-Richtung (senkrecht zu LEP-Ebene)
//  _pvecce(k,3) = Impulskomponente in z-Richtung (entlang e- Richtung)
//  _massce(k) = Masse
//  _charge(k) = Ladung (+- 1 fuer Myonen, sonst 0!)
//
//   Massen: alle Massen sind hypotetisch
//
//   Im Myondetektor rekonstruierte Teilen haben die Masse 0.106 GeV, wenn sie
//   durch die Particle ID als solche identifiziert wurden.
//     Diese Masse wird zwar hier nicht gemessen, wir aber als bekannt vorrausge
stetz
//     und kann damit den Myonen zugeordnet werden.
//
//   Bei Kalorimetertreffern gibt es keine eindeutige Zuordnung
//     zwischen Treffer und Teilchen. Man kann insbesondere das Teilchen
//     weder identifizieren noch seine Ladung messen. Da aber die meisten
//     Teilchen in Jets Pionen sind, wird den Kalorimetertreffern
//     (etwas willkuerlich) die Masse 0.14 GeV zugeordnet.
//
//   Manchmal kann man Kalorimeterhits als elektromagnetischen Ursprungs
//    (Photon, Elektron) erkennen. Dann wird _massce = 0 gesetzt.
//
//  Alle Einheiten: GeV
//
//  Fuer die Analyse duerfen nur die folgenden Funktionen benutzt werden:
//    print();                druckt Inhalt von cevent und mehr (Bildschirm)
//    number_particles();     Zahl der Teilchen im betrachteten Ereignis
//    momentum(k,l);          Impulskomponente l (x,y,z=1,2,3) von Teilchen k
//    mass(k);                Masse von Teilchen k
//    charge(k);              Ladung von Teilchen k (nur sinnvoll fuer Myonen)
//    Daraus koennen alle weitern Groessen, wie die Gesammtenergie, Gesammtimpul
s,
//    invariante Masse zweiter Muonspuren etc.. berechnent werden


const int ktotmx=1000;      // maximal zulaessige Teilchenzahl pro Ereignis

class cevent {
private:
  int _ktotce;
  float _pvecce[ktotmx+1][4];
  float _massce[ktotmx+1];
  float _charge[ktotmx+1];
  bool _ready;
public:
  cevent();
  int setnce(int);
  int setce(int,float,float,float,float);
  int print();
  int number_particles();
  float momentum(int,int);
  float mass(int);
  float charge(int);
};
```

```
cevent::cevent()                    // Konstruktor zum Erzeugen der Klasse ceven
t vom
{
  _ktotce = -1;
  _ready = 0;
}


int cevent::setnce(int ktot)        //  zum Setzen von _ktotce
{                                   //  Danach muss alle Teilcheninformation
  int k;                            //  neu gefuellt werden!

  if(ktot<0) {
    cout << "ERROR cevent::setnce: " << ktot << "negative" << endl;
    return -9;
  }
  else if (ktot<ktotmx) {
    _ktotce = ktot;
    for (k=1; k<=_ktotce; k++) {
      _charge[k]=-9.;
    }
    return 0;
  }
  else {
    cout << "ERROR cevent::setnce: " << ktot << "too large" << endl;
    return -9;
  }
}


int cevent::setce(int k, float px, float py, float pz, float m)
{
  int kk;

  if(_ktotce < 1) {
    cout << "ERROR cevent::setce: _ktotce = " << _ktotce << endl;
    return -9;
  }
  else if (k<=_ktotce) {
    _pvecce[k][1] = px;
    _pvecce[k][2] = py;
    _pvecce[k][3] = pz;
    _massce[k] = fabs(m);
    _charge[k] = 0;
    if(_massce[k]>0) _charge[k] = m/fabs(m);
    _ready = 1;
    for (kk=1; kk<=_ktotce; kk++) {
      if(_charge[kk]<-1.) _ready = 0;
    }
    return 0;
  }
  else {
    cout << "ERROR cevent::setce: _ktotce > k = " << k << endl;
    return -8;
  }
}



int cevent::print()             // Zum ausdrucken der Eventinformation, wir fuer
 die ersten
```



```
{                                 // fuenf Ereignisse aufgerufen
  int k;
  float p2, ptotxe, etotxe;
  float pxtote, pytote, pztote, etote;
  float ptote, mtote;

  if(!_ready) {
    cout << "ERROR cevent::print:  event not (fully) initialized "<< endl;
    return -9;
  }

  cout << endl
       << "Dump von cevent + weitere Groessen: "
       << endl << endl
       << "particle"
       << "  px/GeV "
       << "  py/GeV "
       << "  pz/GeV "
       << " mass/GeV"
       << " ptot/GeV"
       << " etot/GeV" << endl << endl;

  pxtote = 0.;
  pytote = 0.;
  pztote = 0.;
  etote = 0.;

  for (k=1; k<=_ktotce; k++) {
//
// calculate more variables....:
//
    p2 = _pvecce[k][1]*_pvecce[k][1] +
         _pvecce[k][2]*_pvecce[k][2] +
         _pvecce[k][3]*_pvecce[k][3];
    ptotxe = sqrt(p2);
    etotxe = sqrt(p2 + _massce[k]*_massce[k]);

    pxtote = pxtote + _pvecce[k][1];
    pytote = pytote + _pvecce[k][2];
    pztote = pztote + _pvecce[k][3];
    etote  = etote  + etotxe;

    cout.width(5);
    cout << k << " ";
    cout.setf(ios::fixed | ios::right);
    cout.precision(3);
    cout.width(9);
    cout << _pvecce[k][1];
    cout.width(9);
    cout << _pvecce[k][2];
    cout.width(9);
    cout << _pvecce[k][3];
    cout.width(9);
    cout << _massce[k];
    cout.width(9);
    cout << ptotxe;
    cout.width(9);
    cout << etotxe << endl;
  }

  ptote = sqrt(pxtote*pxtote+pytote*pytote+pztote*pztote);
  mtote = sqrt(etote*etote-ptote*ptote);
```

```
  cout.width(7);
  cout << endl << " TOTAL ";
  cout.width(9);
  cout << pxtote;
  cout.width(9);
  cout << pytote;
  cout.width(9);
  cout << pztote;
  cout.width(9);
  cout << mtote;
  cout.width(9);
  cout << ptote;
  cout.width(9);
  cout << etote << endl << endl;

  return 0;

}



int cevent::number_particles()   // Gibt die Zahl der Kalorimetertreffen (Teilch
en)
{                                // im Event zurueck

  if(!_ready) {
    cout << "ERROR cevent::number_particles: "
         <<  "event not (fully) initialized "<< endl;
    return -9;
  }

  return _ktotce;

}


float cevent::momentum(int k, int l)  // Liefert den Impuls des Teilchens k an
{                                     // der Position l (1=x, 2=y, 3=z)

  if(!_ready) {
    cout << "ERROR cevent::momentum:  event not (fully) initialized "<< endl;
    return -9;
  }

  return _pvecce[k][l];

}



float cevent::mass(int k)           // Liefert die Masse des Teilchens k
{

  if(!_ready) {
    cout << "ERROR cevent::mass:  event not (fully) initialized "<< endl;
    return -9;
  }

  return _massce[k];

}
```



```

float cevent::charge(int k)        // Liefert die Ladung des Teilchens k
{

  if(!_ready) {
    cout << "ERROR cevent::charge:  event not (fully) initialized "<< endl;
    return -9;
  }

  return _charge[k];

}

////////////////////////////////////////////////////////////////////////////

char datfile[100];                          // Name des Datenfiles

int read_event(cevent &event);


int main()
{


  int n;
  int k;

  int result;

  int ktot;

  float px_mu;

  char* hbooktitle;
  char* hropenname;
  char* hropenfile;
  char* hropenflag;

  int irecl, istat;
  int icycle;

  int nevmax;
  int nevent = 0;

  char hisfile[100];

//
// Begruesung.....
//
  cout << endl;
  cout << " ***************************************************************\n";
  cout << " *      Willkommen zum F-Praktimumsversuch Z0 !!!         *\n";
  cout << " * >>> analyse.cpp <<<    C++ version  3.0  M.z.N.  02/2007 *\n";
  cout << " ***************************************************************\n";
  cout << endl;

//
//  Eingabeinformation:
//

  cout << "Bitte Namen des Datenfiles angeben (e.g. 89gev.dat)" << endl;
```

```
  cin >> datfile;

  cout << "Bitte Namen des Histogrammfiles angeben (e.g. 89gev.root)" << endl;
  cin >> hisfile;

  cout << "Bitte Zahl der zu verarbeitenden Ereignisse angeben (e.g. 1000)"
       << endl;
  cin >> nevmax;

//
// Initialisierung des Analyse-Paketes ROOT fuer die Offline und Histogramanalys
e

   TFile *histofile = new TFile(hisfile,"RECREATE");
//
// Hier wird das File mit dem bei der Eingabe gewaehlten Namen geoeffnet, in
// dem die Histogramme abgelegt werden
//

   TH1F *N_cluster = new TH1F("N_cls","Anzahl Teilchen im Calo",100,0.,100.);
   TH1F *Muon_px   = new TH1F("mu_px","x-Komponente des Muon-Impulses",100,-50.,50.);
//
//   Hier wurde ein Histogramm erzeugt mit dem Namen N_part,
//      mit 100 Bins zwischen 0. und 100.:
//      (Achtung: die Bin-Grenzen immer als float angeben!)
//        - N_cluster ist der Name des Histogramms innerhalb des Porgrammes
//        - N_cls ist der Name des Histogrammes im Histogrammfile (fuer Root)
//        - "Anzahl .." ist der Histogramm-Titel, der in Root erscheint
//        - 100 ist die Zahl der Bins (Intervalle)
//        - 0. ist die untere Grenze (hier Anzahl der Cluster) x-Achse
//        - 100. ist di obere Grenze der x-Achse
//
//   Definierten Sie hier alle weiteren Histogramme, die Sie benoetigen.
//
//   2. Beispiel: x-Komonente der Muonen

//
//  Schleife ueber alle EREIGNISSE (n)
//
  for(n=1; n<=nevmax; n++) {
//
//  Erzeuge die Klasse event vom Typ cevent
//
    cevent event;
//
//  Einlesen eines Ereignisses aus dem oben ausgewaehlten Datenfile
//
    result = read_event(event);
//
//  read_event gibt zurueck
//      0 falls alles ok ist
//     -2 falls das Ende des Files erreicht ist
//

//
// Sicherheitsabfrage:
// Fuehre die Analyse nur aus , falls das Ereginis nicht leer ist !
//
    if(result==0) {

      nevent++;
//
```



```
//  Die ersten 5 Ereginsse sollen auf den Bildschirm geschrieben werden:
//
      if(nevent<=5) {
        event.print();
      }

//
// Auslese der totalen Anzahl an Teilchen (k) im gegebenen Ereignis
//
      ktot = event.number_particles();

//
//  Fuellen des oben definierten Histogrammes,
//  einmal pro Ereignis fuellen, die darzustellende Variable ist ktot
//
      N_cluster->Fill(ktot);

//
//  Schleife ueber alle TEILCHEN (k=1 bis k=ktot) im gegebenen Ereignis
//

      for(k=1; k<=ktot; k++) {

//
//  Beginne hier die Analyse
//  Beispiel: Selektiere die Muonen (Massen-Kriterium)
//
       if(fabs(fabs(event.mass(k))-0.106)<0.001) {
//
//  Impuls in x-Richtung
//
         px_mu = event.momentum(k,1);

//
// Fuellen des Histogrammes Muon_px mit der Impulskomponente px_mu,
// jetzt fuer alle Muonen im Ereignis!
//
         Muon_px->Fill(px_mu);
       }
      }
    }
    else if(result==-2) {
      break;
    }
  }

//
//  Statistik:
//
  cout << "Zahl der analysierten Ereignisse = " << nevent << endl;

//
// Alle Histogramme werden auf den oben definierten File geschrieben
//

  histofile->Write();

  if(nevent > 0) {
    return 0;
  }
  else {
    return -9;
```

```
  }
}


/////////////////////////////////////////////////////////////////////


int read_event(cevent &event)
{
  int k;
  int l;
  int ktot;
  float px, py, pz, m;
  static bool first = 1;
  static ifstream fin;

  if(first) {
    cout << datfile << endl;
    fin.open(datfile,ios::in);
    first=0;
    if (!fin) {
      cout << "ERROR read_event: " << datfile << " cannot be read !" << endl;
      exit(-1);
    }
  }

  if(!fin.eof()) {
    ktot = -1;
    fin >> ktot;
    if(ktot > 0) {
      event.setnce(ktot);
      for (k=1; k<=ktot; k++) {
        fin >> l >> px >> py >> pz >> m;
        event.setce(l,px,py,pz,m);
      }
      return 0;
    }
    else if(ktot==0) {
      return -1;
    }
    else if(ktot <0) {
      fin.close();
      return -2;
    }
  }
  else {
    fin.close();
    return -2;
  }

  return -9;

}
```