EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH (CERN)

ALEPH 2000-076
SOFTWR 2000-002
21st August 2000

Summer Student Report

Further development and testing of ALPHA++

Abstract

A template class has been written for handling algorithms in ALPHA++. The template class inherits from the STL vector class and provides increased functionality for the user. Performance and memory leak checking have also been carried out on ALPHA++.

Dan Wallin
Dan.Wallin@cern.ch
Dan.Wallin@home.se
21st August 2000

Supervisor: Günther Dissertori
Guenther.Dissertori@cern.ch

1 New design of the analysis part of ALPHA++

The ALPHA++ project has been set up for converting ALEPH data stored in Fortran BOS banks to an object database (*Objectivity/Db*) and to provide a small version of the ALPHA analysis package in C++ [1].

The ALPHA++ package contains two main parts. There is a driver part handling the data transfer to and from the object database and the Fortran BOS memory. The second part takes care of the analysis and evaluation of data. A class diagram can be found in [2].

The main goal of this further development of the ALPHA++ project was to edit the analysis part of the program in order to find a better way to handle the algorithms implemented in the AlephAlgo class. The AlephAlgo class contains several member methods that can be used for evaluation of data, e.g., jet finding and thrust routines.

The Fortran based ALPHA analysis package uses a locking procedure to prevent uninteresting data from being analyzed. Finding a more object oriented approach instead of this locking procedure was an additional goal for the further development.

1.1 Changes in the analysis code

The largest change made to the analysis code of the program is that the AlephAlgo class has been removed. The algorithms implemented in this class have been moved to a new class called the AlephVector class.

1.1.1 The AlephVector class

The AlephVector class (appendix A1) is a template class inheriting from the Standard Template Library (STL) vector class [3]. AlephVectors therefore have the same functionality as the STL vector class, but some new member methods provide extra operations to the AlephVector objects.

The AlephVector class also contains the algorithms formerly present in the AlephAlgo class. That is, when the methods are to be called, they are simply invoked on the AlephVector objects.

1.1.2 Extending the AlephVector class

The AlephVector class could easily be extended with new algoritms for evaluation of data. In the new version of the program, all algoritms present at this stage are located in the AlephVector class.

This could be further developed by implementing subclasses of the AlephVector class. Algorithms that can only be invoked on certain objects could then be grouped into classes. At the moment there exists very little type checking which tests that the member method is of a certain type, e.g., the thrust algorithm can only be invoked on certain AlObjects.

The reasons for defining the AlephVector as a template subclass of the STL vector and not as an ordinary subclass, were to extend the functionality of the

STL vector and at the same time be able to call the member methods with different types of objects.

Templates are preprocessed before they are compiled and no code is generated until they are initiated by a certain type. Therefore all template methods have to be defined in the header file. It would look much nicer if the methods could be defined in the normal *.cpp* file. In the new ANSI standard this can be done by putting the keyword *export* in front of the *template* keyword. The *GNU* compiler installed on the *lxplus* system did not support this feature. To avoid template methods from being generated serveral times for each file, *GNU pragmas* can be used for telling the compiler where declarations and definitions of template methods are placed.

1.1.3 Other changes in the code

Some minor changes have been made to the analysis classes. An assignment operator has been added to each of the subclasses of AlObject. The copy constructors of these classes have also been supplied with initialization calls to the base class.

In the AlphaBanks class a number of new service methods have been implemented. These methods returns AlephVectors of pointers to subclasses of AlObject. Also a new version of the *addObject(...)* method has been added. It takes an AlephVector of pointers to AlObjects as a parameter. This method is useful when storing new objects in the *ObjectV* variable of AlphaBanks.

1.2 Writing a user program

In the former version of the program, the user had to create an object of the AlephAlgo class when running an analysis. The member methods were then called with a STL vector of data stored in the AlphaBanks class. The user could decide what data to use by locking uninteresting data objects. The interesting objects were then found by iterating over the non-locked objects stored in the AlphaBanks class.

In the new version of the program, the user creates a new AlephVector inside the user program with pointers to the objects from the AlphaBanks storage container. All operations are carried out on this new AlephVector of pointers. The AlephVector is edited using standard operations on vectors provided in the STL, e.g., *push_back()* for adding an element to the end of the vector and *erase()* for deleting an object from the vector.

Two new member functions *looperase(...)* and *append(...)* have been added in the AlephVector class. The *looperase(...)* call should be used instead of the *erase(...)* call as soon as the user wants to remove an object in the AlephVector in a for-loop. The *append(...)* call can be used to append the two vectors of the same type to each other.

When the user has created an AlephVector of pointers to the interesting objects, the algorithms can be invoked by calling member methods of the AlephVector. The reason for working with pointers to the objects instead of copies of the whole object is to increase the execution speed, whereas the whole AlephVector of objects would have been to be copied otherwise. It is important to keep in mind that the algorithm member methods of AlephVector only should be invoked from an AlephVector of pointers to objects.

In the earlier version of the program, containing an AlephAlgo class, some of the member methods created new objects that were stored in the *ObjectV* variable in the AlphaBanks class. In this new version this is not automatically carried out when calling these algorithms. If these objects are to be stored they have to be stored manually in the user program. The member method *addObject(...)* of the AlphaBanks class should then be invoked.

2 Performance

Performance tests were carried out on two different user programs called *user_example* and *user_analysis*. Each program was also tested with two different versions. One older version using the AlephAlgo class and one newer version using the AlephVector class called version 2 (v2). The programs can be found in appendix A2, A3, A4 and A5.

The performance checking was carried out using the *GNU* profiler *gprof* [4]. A short program called *Profanalyzer* was written for analyzing the output from *gprof*. The program generates a textfile with information about the functions that have the longest execution time. It also creates a *.kumac* file that can be executed in the Physics Analysis Workstation (*PAW*) [5, 6], producing a histogram.

2.1 Usage of the *Profanalyzer* program

The following steps should be taken in order to perform a profiling of a program. These commands work if all the programs are located in the same directory as your ALPHA++ program. Otherwise the whole path should be added.

1. Compile the program with the *-pg* option. This can be done by setting 'PROFILING=1' in the *GNUmakefile*.

```
[PROMPT] gmake all
```

2. Run the program after editing the *alpha++.input* file.

```
[PROMPT] Linux/user_example -l1
```

3. Run the *GNU* profiler *gprof* on the program that is to be tested. Save the output to a profiler file (e.g. *mc1000.out*).

```
[PROMPT] gprof Linux/user_example > mc1000.out
```

4. Run the *Profanalyzer* program with the (*mc1000.out*) file. The *.kumac* file is a file created for drawing a *PAW*-diagram. The *.dat* file gives a short desription of the functions that took the longest time to execute.

```
[PROMPT] Profanalyzer mc1000.out mc1000.kumac mc1000.dat
```

5. Start *PAW* and type 'exec mc1000'. This draws a *PAW* histogram.

```
PAW > exec mc1000
```

6. To make sure no old data is left, delete the *gmon.out* file before the next test run.

2.2 Editing the *Profanalyzer* program

Two kinds of information is written to the *gprof* profiling file: (1) execution time of each function, (2) a total execution tree of the program. At the moment *Profanalyzer* is only concerned with the execution time.

Profanalyzer contains functions for picking out useful data from the *gprof* file. Examples of these functions can be seen in the *macrowriter.cc* file (commented code). The *macrowriter.cc* file can easily be modified to output other data. A makefile is also provided that can be used to compile the edited program.

The program was written based on an old schoolbook example of a linked list. It turned out at a later stage of the development of the program that a queue (the data first stored is the data first released) would have been more appropriate to use. Therefore there is a member method *formatlist::reverse()* that inverts the list. The program should be seen as a first rough approach to the problem of studying the *gprof* output. Performance is not a problem in the program.

The *Profanalyzer* program contains the following files:

- *Profanalyzer* - The main program. Opens and closes files for reading and writing and creates a *macrowriter* object.

- *macrowriter* - A class that creates a *formatlist* and calls memberfunctions of the *formatlist*. *Macrowriter* also contains functions for printing a datafile *.dat* and a *PAW file .kumac*. This is the class that should be edited for other output.

- *formatlist* - contains a linked list of *format* elements, with member functions for creating, inserting, copying, deleting, picking out and printing *format* elements from the list. In a list the first stored data is the data that is released last. Therefore the list has to be reordered in many member methods to get the correct output.

- *format* - contains an element with data about each function

The program is included in appendix B1 to B7.

2.3 Results

The *Profanalyzer* program was run on *gprof* files produced from the test-programs *user_example*, *user_example_v2*, *user_analysis* and *user_analysis_v2*.

An example of a *PAW* histogram generated by *Profanalyzer* can be seen below.

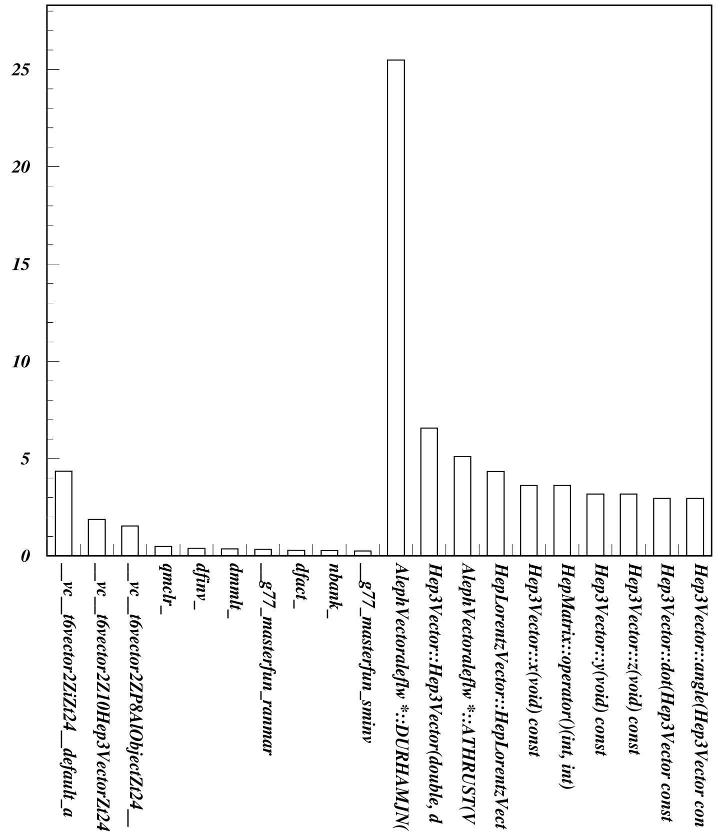

Figure 1: *Execution time in percent of the total time for the ten Fortran and C++ functions that have the longest execution time. Monte Carlo simulation of 10000 events with the user_analysis_v2 program.*

In order to study the dependence on the number of events, testruns were carried out for different number of events ranging from 100 to 10000. Tests were made with both data and Monte Carlo simulations on two files of ALEPH data, *data94.filicards* and *mc94.filicards*.

The percentage of total runtime for Fortran77 and C++ functions is presented in the table below together with the total execution time of each run.

Monte Carlo				
program	events	Fortran 77 (%)	C++ (%)	runtime (s)
user_example	100	94.49	5.46	1.28
	1000	86.92	12.92	5.86
	10000	87.39	12.96	52.6
user_example_v2	100	95.09	4.95	1.62
	1000	89.78	10.04	5.85
	10000	88.19	12.11	54.09
user_analysis	100	20.95	78.95	9.00
	1000	13.40	86.28	88.95
	10000	13.11	86.60	869.96
user_analysis_v2	100	20.73	79.12	9.60
	1000	13.68	86.09	91.37
	10000	12.99	86.58	897.18

Data				
program	events	Fortran 77 (%)	C++ (%)	runtime (s)
user_example	100	94.02	6.08	1.32
	1000	90.79	9.06	4.50
	10000	83.13	16.94	37.16
user_example_v2	100	96.36	3.75	1.34
	1000	90.86	9.35	4.43
	10000	84.89	15.57	38.09
user_analysis	100	19.28	80.63	10.86
	1000	13.10	86.65	91.37
	10000	11.51	88.28	925.63
user_analysis_v2	100	19.70	80.35	10.07
	1000	11.92	87.92	92.29
	10000	11.41	88.18	944.81

2.4 Discussion

The runtime should not be seen as an absolute test of performance, because the tests were carried out on a public server. Different loads on the server can cause the runtime to differ significantly between the runs. Despite this fact, some conclusions can be made.

The *user_program* program mainly spends its runtime executing Fortran code, whereas *user_analysis* has a lot of execution in C++. The older and the newer versions of the programs have similar characteristics, both in efficiency and in amount of time spent in C++ and Fortran.

Looking more carefully in the files generated by *gprof* and *Profanalyzer*, shows that the *user_example* program spends most of its time clearing BOS memory (Fortran routine *qmclr*) and doing matrix inversions (Fortran routine *dfinv*) and matrix multiplications (Fortran routine *dmmlt*). The *user_analysis* program has the longest execution time in the thrust and jet finding algoritms (C++ routines *AThrust* and *DurhamJn*).

3 Memory leak checking

Two different programs were tested for the tracing of memory leaks in ALPHA++. The first was a very simple freeware called *Leaktracer* [7]. It turned out that this program was not possible to use, because all files had to be compiled with the debugging flag *-g*. Not all external libraries were available in debugged versions.

The second program was a commercial program, *Insure++*, from *ParaSoft* [8]. This program could be used for the ALPHA++ project.

3.1 Insure++ results

The user programs *user_example* and *user_example_v2* were run on data in the file *data94.filicards*. Five different runs were made on each user program, with the number of events ranging between 100 and 5000.

Insure++ generated a long list of possible memory leaks. The leaks mainly seemed to be located in the *Objectivity/Db* code for database handling. The remaining leaks are mostly related to the allocation of the AlephSession and the AlephManagers. These operations are carried out only once during the execution. Very few leaks are related to the code in ALPHA++.

In the following table the total reported leaks are presented.

memory leaks in ALPHA++		
program	events	memory leaks (byte)
user_example	100	2980069
	500	3107531
	1000	3142294
	2000	3353154
	5000	4167823
user_example_v2	100	2977475
	500	3104974
	1000	3159650
	2000	3371326
	5000	4165210

The memory leaks grow linearly with the number of events. A fixed size of about 2900000 bytes occurs independently from the number of events of the problem. A graph of the size of the memory leak as a function of the number of events can be found below.

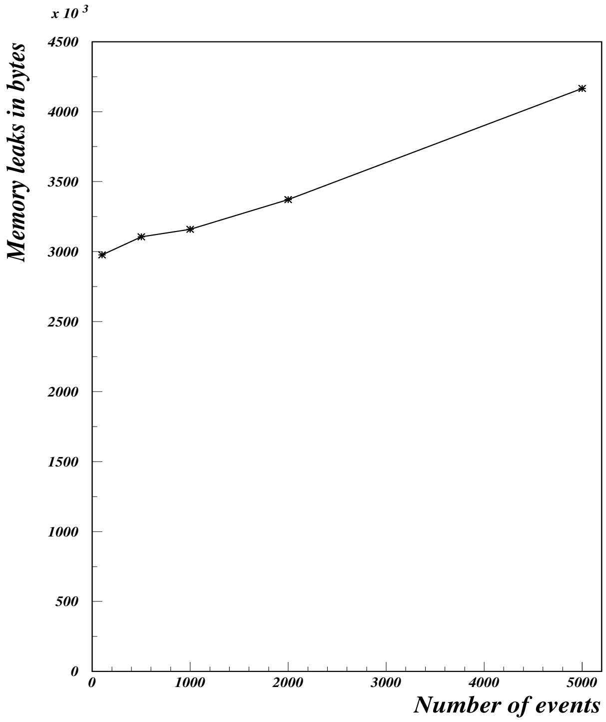

Figure 2: *Total memory leaks in the user_program_v2 program as a function of the number of events.*

No big difference is found between the older and the newer version of the program. This is due to the fact that almost all registered leaks are in the *Objectivity/Db* and not related to the analysis part of ALPHA++.

4 Acknowledgements

I especially would like to thank my supervisor Günther Dissertori for helping me to get started and for answering all my questions.

I also would like to thank Dirk Düllmann for answering questions about *LeakTracer* and C++. Vincent Lemaitre helped me with some Fortran related questions. Finally I thank Alvise Spano for teaching me about the compiling stages in C++ related to template classes.

References

[1] ALPHA++ - The ALPHA++ Project in ALEPH, http://alephwww.cern.ch/ disserto/objty/

[2] Hörndl M., ALEPH Collaboration, CERN Summer Student Project: ALPHA++, ALEPH 99-097, 1999, p 4.

[3] Musser D. R. and Saini A., STL Tutorial and Reference Guide, Addison-Wesley, 1996.

[4] Fenlason J. and Stallman R, The GNU profiler, http://www.cs.utah.edu/dept/old/texinfo/as/gprof_toc.html.

[5] PAW: Physics Analysis Workstation, CERN Program Library http://www.cs.utah.edu/dept/old/texinfo/as/gprof_toc.html.

[6] HBOOK: Statistical Analysis and Histogramming, CERN Program Library.

[7] Andreasen E., LeakTracer, http://www.andreasen.org/LeakTracer/

[8] CERN API Group Software Development Tools, Insure++, http://sdt.cern.ch/Insure/

A1 - AlephVector.h

```
////////////////////////////////////////////////
// AlephVector.h
// Subclass of vector.h
//
// Author: Dan Wallin  Date: 2000-08-14
//
////////////////////////////////////////////////

#ifndef __AlephVector__
#define __AlephVector__

#include <iostream.h>
#include <ospace/std/vector>
#include "AlObject.h"

#include "AlThrust.h"
#include "AlJet.h"
#include "CLHEP/Matrix/Matrix.h"

// Prevents the compiler from generating several copies of
// methods during the precompiling stage, which leads to longer
// compile time.
#if defined(__GNUC__)
#pragma interface
#endif

template <class Type>
class AlephVector: public vector<Type>{

public:

  // Constructors
  AlephVector::AlephVector():vector<Type>(){}

  // Constructors
  AlephVector::AlephVector(int i):vector<Type>(i){}

  // Print all elements
  void print();
  void print2();

  // Add two vectors to each other
  void append(AlephVector<Type>);
```

```
  // Erase an element from the vector, should be used in loops
  void looperase(AlephVector<Type>::iterator &it){
    AlephVector::iterator tmp=it;
    it--;
    erase(tmp);
  }

  // The Thrust algorithm
  AlThrust AThrust();

  // DurhamJn
  AlephVector<AlJet> DurhamJn(int, float);

  // DurhamYn
  AlephVector<float> DurhamYn(int);

  // Removing elements
  //void removeall_x(int (*logic)(int,int),int);
  //void removeall_y(int (*logic)(int,int),int);

};

#if defined(__GNUC__)
#pragma implementation
#endif
// Implementations...

// Debugging print function
template <class Type>
void AlephVector<Type>::print(){
  iterator i;
  int ii=1;
  for (i=begin();i<end();i++){
    cout << ii << " My vector: " << *i << " " << (**i).QX() << endl;
    ii++;
  }
}

// Debugging print function
template <class Type>
void AlephVector<Type>::print2(){
  iterator i;
  int ii=1;
  for (i=begin();i<end();i++){
    cout << ii << " My vector: " << i << " " << *i << endl;
```

```
    ii++;
  }
}

// Appends a vector of a certain kind to another vector
template <class Type>
void AlephVector<Type>::append(AlephVector<Type> cv){
  iterator i;
  for(i=cv.begin();i<cv.end();i++){
    push_back(*i);
  }
}

// AlThrust algoritm
template <class Type>
AlThrust AlephVector<Type>::AThrust() {
  iterator i;
  //by the default constructor
  for (i=begin();i<end();i++){
    if ((**i).TYPE() == TRACK  && (**i).TYPE() != EFLOW   &&
        (**i).TYPE() == GAMPEC && (**i).TYPE() != ALJET){
      cout << "AThrust : WARNING : type not supported!" << endl;
      exit(1);
    }
  }

  AlephVector<Hep3Vector> p;
  int Np = size();      //number of particles
  AlThrust thr;         //variables of thr are zero as set
                        //by the default constructor

  // select the three-momenta
  AlephVector<Type> tmp = *this;
  for(int i=0; i < Np; i++){
    p.push_back(Hep3Vector(tmp[i]->A4V()));
  }

  Hep3Vector qtbo;
  Hep3Vector zero(0.,0.,0.);
  float vnew;

  // for more than 2 tracks
  if (Np > 2){
    float vmax  = 0.;
    Hep3Vector vn, vm, vc, vl;
```

```
for(int i=0; i< Np-1; i++){
  for(int j=i+1; j < Np; j++){
    vc = p[i].cross(p[j]);
    vl = zero;

    for(int k=0; k<Np; k++){
      if ((k != i) && (k != j)){
        if (p[k].dot(vc) >= 0.){
          vl = vl + p[k];
        }
        else {
          vl = vl - p[k];
        }
      }
    }

    // make all four sign-combinations for i,j

    vn = vl + p[j] + p[i];
    vnew = vn.mag2();
    if (vnew >  vmax){
      vmax = vnew;
      vm = vn;
    }

    vn = vl + p[j] - p[i];
    vnew = vn.mag2();
    if (vnew >  vmax){
      vmax = vnew;
      vm = vn;
    }

    vn = vl - p[j] + p[i];
    vnew = vn.mag2();
    if (vnew >  vmax){
      vmax = vnew;
      vm = vn;
    }

    vn = vl - p[j] - p[i];
    vnew = vn.mag2();
    if (vnew >  vmax){
      vmax = vnew;
      vm = vn;
    }
```

```
      }
    }
    // sum momenta of all particles and iterate

    for(int iter=1; iter<=4; iter++){
      qtbo = zero;
      for(int i=0; i< Np; i++){
        if (vm.dot(p[i]) >= 0.){
          qtbo = qtbo + p[i];
        }
        else{
          qtbo = qtbo - p[i];
        }
      }
      vnew = qtbo.mag2();
      if (vnew  == vmax) {
        break;
      }
      fout << endl << " I have to iterate again " <<  endl << endl;
      vmax = vnew;
      vm = qtbo;
    }
  }  // of if Np > 2
  else{
    if (Np == 2){
      if (p[0].dot(p[1]) >= 0.){
        qtbo = p[0] + p[1];
      }
      else{
        qtbo = p[0] - p[1];
      }
    }
    else if (Np == 1){
      qtbo = p[0];
    }
    else{
      qtbo = zero;
      // thrustvalue and direction are set to zero by
      // default constructor
      // shall it also be filled into ObjectV ???
      return thr;
    }

  }

  // normalize thrust -division by total momentum-
```

```
  float vsum = 0.;
  for(int i=0; i < Np; i++){
    vsum = vsum + p[i].mag();
  }

  vnew  = qtbo.mag();
  float v = vnew/vsum;
  float x = qtbo.x()/vnew;
  float y = qtbo.y()/vnew;
  float z = qtbo.z()/vnew;

  thr.setA4V(x, y, z, v);

  //saknas lite...


  return thr;
}

template <class Type>
AlephVector<AlJet> AlephVector<Type>::
                   DurhamJn(int Scheme, float Ycut){
  /*
    Routine to calculate number of jet and jet momenta for
    a given Ycut and a given AlephVector of pointers to
    AlObjects
    Input:   float Ycut                Ycut value
    int Scheme                         Recombination Scheme
    Scheme = 0 : E0 - Scheme
    Scheme = 1 : E - Scheme
    int Np                             Number of particles
    AlephVector<AlObject*> P           AlephVector of Pointer
                                       to AlObjects

    Output: AlephVector<AlJet> jets    Vector of Jets
    */
  float max(float i, float j);
  float max(double i, float j);
  float min(float i, float j);
  float min(double i, float j);

  int Np = size();

  int Imin, Jmin, IminC, Njet;
  float Ymin, charge;
```

```
AlephVector<AlObject*> P; //copy of Pext, so that changes of P
// do not influence Pext
AlephVector<int> Iflag;

// Copy momenta to internal AlephVector P, initialize
// rescale all momenta to zero mass in case E0
float Evis = 0.;

AlephVector<Type> tmp2 = *this;
for(int i=0; i< Np; i++){
  QvecBase* tmp1 = (QvecBase*)(tmp2[i]);
  AlObject* tmp = new QvecBase(*tmp1);
  P.push_back(tmp);
  Iflag.push_back(0);
  float e = P[i]->A4V().e();
  Evis = Evis + e;
  float PnClus = Hep3Vector(P[i]->A4V()).mag();
  if (Scheme == 0) {
    float px = P[i]->A4V().px()*P[i]->A4V().e()/
                              (max(pow(10,-7),PnClus));
    float py = P[i]->A4V().py()*P[i]->A4V().e()/
                              (max(pow(10,-7),PnClus));
    float pz = P[i]->A4V().pz()*P[i]->A4V().e()/
                              (max(pow(10,-7),PnClus));
    P[i]->setA4V(px,py,pz,e);
  }
}

HepMatrix Y(Np,Np);
HepMatrix StoreObj(Np,Np);

AlephVector<AlJet> jets(Np);

for(int i=1; i<=Np; i++)
  for(int j=1; j<=Np; j++)
    StoreObj(i,j)=0;


// Check if # of particles is o.k.
if (Np < 2){
  jets.resize(0);
  fout << "there is only one particle, therefore DurhamJn
            cannot be performed" << endl;
  return jets;
}
```

```
Njet = Np;
IminC = -1;
Ymin  = 0.;
//loop over # of jets

while ((Njet >= 1) && (Ymin <= Ycut)){
  Ymin  = 100000.;

  //loop over combination of particles

  for(int i=0; i<Np-1; i++){
    if (Iflag[i] == 1)
      continue;
    for (int j=i+1; j< Np; j++){
      if (Iflag[j] == 1)
        continue;
      // if nothing has changed for i and j
      if (((i != IminC) && (j != IminC)) && (Njet != Np)){
        if (Y(i+1,j+1) < Ymin){
          Ymin = Y(i+1,j+1);
          Imin = i;
          Jmin = j;
        }
      }
      else{  //else compute metric now
        Y(i+1,j+1) = 2.*min(pow(P[i]->A4V().e()/Evis,2),
                            pow(P[j]->A4V().e()/Evis,2)  ) *
          (1. - cos(P[i]->A4V().angle(Hep3Vector(P[j]->A4V()))));
        if (Y(i+1,j+1) < Ymin) {
          Ymin = Y(i+1,j+1);
          Imin = i;
          Jmin = j;
        }
      }    // else
    }  // for (int j ....
  }    // for (int i ....


  // compare Ymin with Ycut
  if (Ymin > Ycut){
    //fill vector of jets
    int i = 0;
    int iclus = 0;
    while (i < Np){
      if (Iflag[i] == 0){
        jets[iclus].setA4V(P[i]->A4V());
```

```
      jets[iclus].setScheme(Scheme);
      charge = P[i]->QCH();
      jets[iclus].addObject(tmp2[i]);
      for(int j=0; j< Np; j++){
        if (StoreObj(i+1,j+1) == 1){
          jets[iclus].addObject(tmp2[j]);
          charge = charge + P[j]->QCH();
        }
      }
      jets[iclus].setQCH(charge);
      iclus ++;
    }
    i++;
  }
  jets.resize(iclus);
}
else{
  // recombine according to E-Scheme
  if (Scheme == 1){
    P[Imin]->setA4V(P[Imin]->A4V() + P[Jmin]->A4V());
  }

  // recombine according to E0-Scheme
  if (Scheme == 0){
    float e = P[Imin]->A4V().e() + P[Jmin]->A4V().e();
    // P[Imin]->A4V().setE(P[Imin]->A4V().e()+
    // P[Jmin]->A4V().e());
    float P2i = Hep3Vector(P[Imin]->A4V()+
                P[Jmin]->A4V()).mag();
    if (P2i <= pow(10,-14)){
      cout << " Error: Division by zero! " << endl;
      exit(1);
    }
    float px = e * (P[Imin]->A4V().px() +
                    P[Jmin]->A4V().px())/P2i;
    float py = e * (P[Imin]->A4V().py() +
                    P[Jmin]->A4V().py())/P2i;
    float pz = e * (P[Imin]->A4V().pz() +
                    P[Jmin]->A4V().pz())/P2i;

    P[Imin]->setA4V(px,py,pz,e);
  }

  //  lock the other track
  Iflag[Jmin] = 1;
  //  Store Jmin and all objnumbers of Jmin in StoreObj
```

```
      StoreObj(Imin+1,Jmin+1)=1;
      for(int j=0; j < Np; j++){
        if (StoreObj(Jmin+1,j+1) == 1) StoreObj(Imin+1,j+1) = 1; }
      IminC = Imin;
      Njet --;
    } // else
  }   //while

  // delete all the copies of the AlephObjects
  for (AlephVector<AlObject*>::iterator io = P.begin();
       io < P.end(); io++ ){
    AlObject* tmp = *io;
    // set the pointers in the vector to NULL, to be sure
    // that the vector does not suffer from deleting the memory
    *io  = NULL;
    delete tmp;
  }

  // give the jets vector back
  return jets;
}

template <class Type>
AlephVector<float> AlephVector<Type>::DurhamYn(int Scheme){
  /* Routine to calculate Yn, where Yn is minimum metric in case
     of a n-jet event

     AlephVector<AlObject*> Pext     AlephVector of pointers to
              AlObjects, which this procedure gets from outside
     int Np                          Number of external particles
     Input: int Scheme               Recombination Scheme
     Scheme = 0 : E0 - Scheme
     Scheme = 1 : E  - Scheme
     Output: AlephVector<float> Yn   AlephVector of Yn
     */

  float max(float i, float j);
  float max(double i, float j);
  float min(float i, float j);
  float min(double i, float j);

  int Np=size();
  AlephVector<HepLorentzVector> P;

  int Njets, Imin, Jmin, IminC;
  float Ymin;
```

```
AlephVector<int> Iflag;

// Copy momenta to internal HepLorentzVector P, initialize
// rescale all momenta to zero mass in case E0
float Evis = 0.;

AlephVector<Type> tmp = *this;
for(int i=0; i< Np; i++){
  P.push_back(tmp[i]->A4V());
  Iflag.push_back(0);
  Evis = Evis + P[i].e();
  float PnClus = Hep3Vector(P[i]).mag();
  if (Scheme == 0){
    P[i].setPx(P[i].px()*P[i].e()/(max(pow(10,-7),PnClus)));
    P[i].setPy(P[i].py()*P[i].e()/(max(pow(10,-7),PnClus)));
    P[i].setPz(P[i].pz()*P[i].e()/(max(pow(10,-7),PnClus)));
  }
}

AlephVector<float> Yn(Np-2);
HepMatrix Y(Np,Np);

// Check if # of particles is o.k.
if (Np < 3){
  Yn.resize(0);
  fout << "there are only two tracks, therefore DurhamYn
           cannot be performed" << endl;
  return Yn;
}

Njets = Np;
IminC = -1;
//loop over # of jets

while (Njets >= 3){
  Ymin  = 100000.;

  //loop over combination of particles

  for(int i=0; i < Np-1; i++){
    if (Iflag[i] == 1)
      continue;
    for (int j=i+1; j < Np; j++){
      if (Iflag[j] == 1)
        continue;
```

```
    // if nothing has changed for i and j
    if (((i != IminC) && (j != IminC)) && (Njets != Np)){
      if (Y(i+1,j+1) < Ymin){
        Ymin = Y(i+1,j+1);
        Imin = i;
        Jmin = j;
      }
    }
    else{//else compute metric now
      Y(i+1,j+1) = 2.*min(pow(P[i].e()/Evis,2),pow(P[j].e()/Evis,2))
                     * (1. - cos(P[i].angle(Hep3Vector(P[j]))));
      if (Y(i+1,j+1) < Ymin){
        Ymin = Y(i+1,j+1);
        Imin = i;
        Jmin = j;
      }
    }    // else
  }  // for (int j ....
}    // for (int i ....

// store Ymin in Yn
Yn[Njets-3] = Ymin;
Njets  --;

// recombine according to E-Scheme
if (Scheme == 1){
  P[Imin] = P[Imin] + P[Jmin]; }

// recombine according to E0-Scheme
if (Scheme == 0){
  P[Imin].setE(P[Imin].e() + P[Jmin].e());
  float P2i = Hep3Vector(P[Imin]+ P[Jmin]).mag();
  if (P2i <= pow(10,-14)){
    cout << " Error: Division by zero! " << endl;
    exit(1);
  }
  P[Imin].setPx(P[Imin].e() * (P[Imin].px() +
                               P[Jmin].px())/P2i);
  P[Imin].setPy(P[Imin].e() * (P[Imin].py() +
                               P[Jmin].py())/P2i);
  P[Imin].setPz(P[Imin].e() * (P[Imin].pz() +
                               P[Jmin].pz())/P2i);
}
//  lock the other track
Iflag[Jmin] = 1;
IminC = Imin;
```

```
  }   //while
  return Yn;
}

#endif
```

A2 - user_example.cpp - old version

```
//////////////////////////////////////////////////////////
// example code for user analysis routines
//
// Author : G.Dissertori, G.Bagliesi
//
///////////////////////////////////////////////////////////

#include <ospace/std/fstream>
#include "AlephExManager.h"
#include "Hbook.h"

extern ofstream fout;

///////////////////////////////////////////////////////////
//  User Init
///////////////////////////////////////////////////////////

static int Ngood = 0;

void AlephExManager::UserInit(){
  Hbook1d(10,"N_tpc",21,0.5,22.5);
  Hbook1d(20,"ptch ",100,0.,45.);
  Hbook1d(30,"costhch",100,-1.,1.);
  Hbook1d(40,"d0 ",50,-5.,5.);
  Hbook1d(50,"z0 ",50,-20.,20.);
  Hbook1d(60,"Nch raw ",100,-0.5,99.5);
  Hbook1d(70,"Ech raw ",100,0.,100.);

  Hbook1d(100,"Nch sel ",100,-0.5,99.5);
  Hbook1d(110,"Ech sel ",100,0.,100.);
  Hbook1d(120,"xp ch   ",100,0.,1.);
}

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

///////////////////////////////////////////////////////////
//  User Event
///////////////////////////////////////////////////////////
void AlephExManager::UserEvent(AlphaBanks& bb)
{
  int Ich;

// initialize
  int   Nchraw = bb.TrackV().size();
```

```
  int   Nchsel = 0;
  float Echraw = 0.;
  float Echsel = 0.;

  typedef AlphaBanks::AlTrackVec::iterator TrackItr;

// selection of good tracks
   for (TrackItr Ich=bb.TrackV().begin();
        Ich<bb.TrackV().end(); Ich++)
   {
      int    Ntpc = (*Ich).NT();
      float  px   = (*Ich).QX();
      float  py   = (*Ich).QY();
      float  pz   = (*Ich).QZ();
      float  ene  = (*Ich).QE();
      float  ptch = sqrt(px*px + py*py);
      float  costhch = pz/(*Ich).QP();
      float  d0      = (*Ich).QDB();
      float  z0      = (*Ich).QZB();

      Echraw  += ene;

// fill the track attributes into histos
      Hfill1d(10,(float)Ntpc);
      Hfill1d(20,ptch);
      Hfill1d(30,costhch);
      Hfill1d(40,d0);
      Hfill1d(50,z0);

// do track selection
      if ((Ntpc < 4)    ||
          (ptch < 0.2)  ||
          (fabs(costhch) > 0.9) ||
          (d0 > 2.)     ||
          (z0 > 10.))
        {
          // lock it if bad
          (*Ich).Lock();
          //  fout << "track locked " << Ich << endl;
        }
   }

// fill global properties
   Hfill1d(60,(float)Nchraw);
   Hfill1d(70,Echraw);
```

```
// now select the event according to the good charged tracks
   for (TrackItr Ich=bb.TrackV().begin();
        Ich<bb.TrackV().end(); Ich++)
     {
       if ((*Ich).isLocked() == 0)
         {
           Nchsel++;
           Echsel += (*Ich).QE();
         }
     }

   if ((Nchsel >= 5)     &&
       (Echsel >= 15.))
     {

// now we have a good event
       Ngood++;

// fill the momentum for good tracks
       for (TrackItr Ich=bb.TrackV().begin();
            Ich<bb.TrackV().end(); Ich++)
         {
           if ((*Ich).isLocked() == 0)
             {
               float xp = 2*(*Ich).QP()/91.2;
               Hfill1d(120,xp);
             }
         }

       Hfill1d(100,(float)Nchsel);
       Hfill1d(110,Echsel);

     }
}
////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////
//  User Term
////////////////////////////////////////////////////////////
void AlephExManager::UserTerm(){
  fout << endl;
  fout << " USER TERM : Ngood = " << Ngood << endl;
}
```

A3 - user_example_v2.cpp - new version

```
///////////////////////////////////////////////////////////
// example code for user analysis routines
//
// Author : G.Dissertori, G.Bagliesi
//
// Modified by Dan Wallin    Date: 2000-08-14
//
////////////////////////////////////////////////////////////

#include <ospace/std/fstream>
#include "AlephExManager.h"
#include "Hbook.h"
#include "AlephVector.h"

extern ofstream fout;

////////////////////////////////////////////////////////////
//  User Init
////////////////////////////////////////////////////////////

static int Ngood = 0;

void AlephExManager::UserInit(){
  Hbook1d(10,"N_tpc",21,0.5,22.5);
  Hbook1d(20,"ptch ",100,0.,45.);
  Hbook1d(30,"costhch",100,-1.,1.);
  Hbook1d(40,"d0 ",50,-5.,5.);
  Hbook1d(50,"z0 ",50,-20.,20.);
  Hbook1d(60,"Nch raw ",100,-0.5,99.5);
  Hbook1d(70,"Ech raw ",100,0.,100.);

  Hbook1d(100,"Nch sel ",100,-0.5,99.5);
  Hbook1d(110,"Ech sel ",100,0.,100.);
  Hbook1d(120,"xp ch   ",100,0.,1.);
}


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


////////////////////////////////////////////////////////////
//  User Event
```

```
////////////////////////////////////////////////////////////
void AlephExManager::UserEvent(AlphaBanks& bb){

  typedef AlephVector<AlTrack *>::iterator TI;
  AlephVector<AlTrack *> altp=bb.TrackPV();

  int   Nchraw = altp.size();
  int   Nchsel = 0;
  float Echraw = 0.;
  float Echsel = 0.;

  int Ntpc;
  float px, py, pz, ene, ptch, costhch, d0, z0;
  TI Ich;

  for (Ich=altp.begin(); Ich<altp.end(); Ich++){
    Ntpc = (**Ich).NT();
    px   = (**Ich).QX();
    py   = (**Ich).QY();
    pz   = (**Ich).QZ();
    ene  = (**Ich).QE();
    ptch = sqrt(px*px + py*py);
    costhch = pz/((**Ich).QP());
    d0      = (**Ich).QDB();
    z0      = (**Ich).QZB();

    Echraw  += ene;
    cout << Ntpc << endl;
    // fill the track attributes into histos
    Hfill1d(10,(float)Ntpc);
    Hfill1d(20,ptch);
    Hfill1d(30,costhch);
    Hfill1d(40,d0);
    Hfill1d(50,z0);
    if ((Ntpc<4) || (ptch<0.2) || (fabs(costhch)>0.9) ||
        (d0>2.)  || (z0>10.)){
      altp.looperase(Ich);  // Erases elements in the altp vector
    }                       // Should be used in loops
  }

  Hfill1d(60,(float)Nchraw);
  Hfill1d(70,Echraw);

  // now select the event according to the good charged tracks
  for (Ich=altp.begin(); Ich<altp.end(); Ich++){
    Nchsel++;
```

```
    Echsel += (**Ich).QE();
  }

  if ((Nchsel >= 5) && (Echsel >= 15.)) {
    // now we have a good event
    Ngood++;
    for (Ich=altp.begin(); Ich<altp.end(); Ich++){
      float xp = 2*(**Ich).QP()/91.2;
      Hfill1d(120,xp);
      Hfill1d(100,(float)Nchsel);
      Hfill1d(110,Echsel);
    }
  }
}

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



//////////////////////////////////////////////////////////
//  User Term
//////////////////////////////////////////////////////////
void AlephExManager::UserTerm(){
  fout << endl;
  fout << " USER TERM : Ngood = " << Ngood << endl;
}
```

A4 - user_analysis.cpp - old version

```
///////////////////////////////////////////////////////////
// DO_NOTHING version of user analysis routines
//
// Author : G.Dissertori
//          M.Hoerndl
//
//
////////////////////////////////////////////////////////////

#include <ospace/std/fstream>

#include "AlephExManager.h"
#include "Hbook.h"
#include "AlephAlgo.h"
#include "AlephVector.h"

extern ofstream fout;

static int Ngood = 0;
//static int Nev = 0;
extern int Nev = 0;

static int nYcut = 10;

static float Ycut[10] = { 0.001, 0.0025, 0.004, 0.0055,
               0.0085, 0.01, 0.025, 0.04, 0.08, 0.1 };
static float Yclo[11] = { 0., 0.0015, 0.003, 0.005,
        0.007, 0.009, 0.015, 0.04, 0.065, 0.09, 0.11};


////////////////////////////////////////////////////////////
//  User Init
////////////////////////////////////////////////////////////
void AlephExManager::UserInit()
{

  Hbookb(10, "2jetrate as function of Ycut",nYcut,Yclo);
  Hbookb(11, "3jetrate as function of Ycut",nYcut,Yclo);
  Hbookb(12, "4jetrate as function of Ycut",nYcut,Yclo);
  Hbookb(13, "5jetrate as function of Ycut",nYcut,Yclo);

  Hbook1d(21,"Mom, 1st Jet", 50, 0., 50.);
  Hbook1d(22,"Mom, 2nd Jet", 50, 0., 50.);
```

```
  Hbook1d(23,"Mom, 3rd Jet", 50, 0., 50.);

  Hbook1d(31,"Energy, 1st Jet", 50, 0., 50.);
  Hbook1d(32,"Energy, 2nd Jet", 50, 0., 50.);
  Hbook1d(33,"Energy, 3rd Jet", 50, 0., 50.);

  Hbook1d(41,"Nch, 1st Jet", 50, 5., 30.);
  Hbook1d(42,"Nch, 2nd Jet", 50, 5., 30.);
  Hbook1d(43,"Nch, 3rd Jet", 50, 5., 30.);

  Hbook1d(51," psi 12", 100, 0., 3.141592);
  Hbook1d(52," psi 13", 100, 0., 3.141592);
  Hbook1d(53," psi 23", 100, 0., 3.141592);

  Hbook2d(100, "Nch of first jet as function of energy",
           50, 0., 50., 20, 0.5, 20.5);

  // book an ntuple
  int ntags = 3;
  char** mytags = new char*[ntags];
  mytags[0] = "var1";
  mytags[1] = "ecm1";
  mytags[2] = "mom";

  Hbookn(1000,"myNtuple",ntags,mytags);

  delete [] mytags;


  fout << endl << " =====> in UserInit " << endl;
}


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



//////////////////////////////////////////////////////////
//  User Event
//////////////////////////////////////////////////////////
void AlephExManager::UserEvent(AlphaBanks& bb){
  Nev++;

  typedef AlphaBanks::AlObjectVec::iterator ObjItr;
```

```
AlephAlgo a(&bb);

// only EFLOW objects are used, lock all the other objects
for(ObjItr it = bb.ObjectV().begin();
    it < bb.ObjectV().end(); it++){
  if ((*it)->TYPE() != EFLOW){
    (*it)->Lock();
  }
}

// do event selection
for(ObjItr it = bb.ObjectV().begin();
    it < bb.ObjectV().end(); it++){
  if ((*it)->isLocked() == 0){

    int type = ((AlEflw*)(*it))->getEfType();

    QvecBase* tmp = (QvecBase*)(*it);
    float costheta = tmp -> QCT();

    // selection of good charged tracks
    if (type <=3){
      int Ntpc;
      AlTrack* theAssTrack = ((AlEflw*)(*it))->getTrack();
      if (theAssTrack == NULL){
        Ntpc = 0;
      }
      else{
        Ntpc = theAssTrack->NT();
      }
      float ptch = tmp->QPT();  // transverse momentum

      // as QDB and QZB are members of QvecBase
      float d0 = tmp -> QDB();
      float z0 = tmp -> QZB();

      // lock if bad
      if ((d0>2.) || (z0>5.) || (fabs(costheta)>0.94)
                  || (ptch<0.2)){
        (*it)->Lock();
      }
    }
    // all Eflow objects with eflow type > 5 are locked
    if (type > 5){
      (*it)->Lock();
    }
```

```
  }
}

// check if Nchgood and Echgood are big enough
int Nchgood = 0;
float Echgood = 0.;
for(ObjItr it = bb.ObjectV().begin();
    it < bb.ObjectV().end(); it++){
  if ((*it)->isLocked() == 0){
    int type = ((AlEflw*)(*it))->getEfType();
    if (type <=3){
      Nchgood ++;
      Echgood += (*it)->QE();
    }
  }
}

fout << Nev << " : Nch = " << Nchgood << "  Ech = "
     << Echgood << endl;

if ((Nchgood <= 0) || (Echgood <= 0)){
  fout << " ******** :-0 :-0  WARNING : Nev = " << Nev << endl;
}
else{
  //compute thrust axis with all good Eflows
  AlThrust theThrust =  a.AThrust(EFLOW);
  Hep3Vector theThrAxis = theThrust.getThrustDirection();
  float costhr = theThrAxis.cosTheta();

  // final event selection
  if ((Nchgood>=5) && (Echgood>=15.) && (fabs(costhr)<0.82)){
    // now he have a good event
    Ngood++;

    // store all not locked objects in the
    // AlephVector<AlObject*> selobj;
    AlephVector<AlObject*> selobj;
    for(ObjItr it = bb.ObjectV().begin();
        it < bb.ObjectV().end()-1; it++){
      if (((*it)->isLocked()==0)&&((*it)->TYPE()==EFLOW)){
        selobj.push_back(*it);
      }
    }

    // calculate various properties with the non locked objects
```

```
// compute the different jet-rates as a function of Ycut;
// 10 different Ycut values set

int jetr;

HepLorentzVector jetmomentum[3];
float multiplicity[3] = {0., 0., 0.};
float interjetangle[3];
for(int i=0; i<10; i++){
  AlephVector<AlJet> Jn = a.DurhamJn(0, Ycut[i], selobj);
  jetr = Jn.size();
  fout << " jetr  = " << jetr << endl;
  if (jetr == 2) Hfill1d(10,Ycut[i]);
  if (jetr == 3) Hfill1d(11,Ycut[i]);
  if (jetr == 4) Hfill1d(12,Ycut[i]);
  if (jetr >= 5) Hfill1d(13,Ycut[i]);


  // take three jet events for Ycut = 0.01
  if (( i == 5) && ( jetr == 3 )){
    for(int j=0; j<3; j++){
      jetmomentum[j] = Jn[j].A4V();

      // calculate the multiplicity
      for(ObjItr it1 = Jn[j].getObjects().begin();
          it1 <Jn[j].getObjects().end(); it1 ++){
        int type = ((AlEflw*)(*it1))->getEfType();
        if (type <=3){
          multiplicity[j]++;
        }
      }
    }


    // store the jets so that the energy decreases
    // from jetmom[0] to jetmom[2]
    int ready = 0;
    while (ready == 0){
      ready = 1;
      for (int k = 0; k < 2; k++){
        for (int l=1; l < 3; l++){
          if (l != k){
            if (jetmomentum[l].e() > jetmomentum[k].e()){
              HepLorentzVector mom = jetmomentum[k];
              jetmomentum[k] = jetmomentum[l];
              jetmomentum[l] = mom;
```

```
                float mult = multiplicity[k];
                multiplicity[k] = multiplicity[l];
                multiplicity[l] = mult;
                ready = 0;
              } // of if
            }  // of if
          }  // of for
        }  // of for
      } // of while

      // calculate the interjetangles
      interjetangle[0]=jetmomentum[0].
                       angle(Hep3Vector(jetmomentum[1]));
      interjetangle[1]=jetmomentum[0].
                       angle(Hep3Vector(jetmomentum[2]));
      interjetangle[2]=jetmomentum[1].
                       angle(Hep3Vector(jetmomentum[2]));

      // fill histograms
      for (int ih=0; ih<3; ih++){
        Hfill1d(20+ih+1,((Hep3Vector)jetmomentum[ih]).mag());
        Hfill1d(30+ih+1,jetmomentum[ih].e());
        Hfill1d(40+ih+1,(float)(multiplicity[ih]));
        Hfill1d(50+ih+1,interjetangle[ih]);
      }

      Hfill2d(100,jetmomentum[0].e(),multiplicity[0], 1.);
    }  // of if 3jet
  }  // of for over ycuts
}  // of if good event
} // end of check if Nch or Ech > 0
if ( Nev%100 == 0 )
  fout << endl << " =====> in UserEvent " << Nev << endl;
}


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



////////////////////////////////////////////////////////////
//  User Term
////////////////////////////////////////////////////////////
void AlephExManager::UserTerm(){
```

```
  fout << endl << " =====> in UserTerm " << endl;
  fout << endl << " Number of selected events = "
       << Ngood << endl << endl;
}
```

A5 - user_analysis_v2.cpp - new version

```
////////////////////////////////////////////////////////////
// DO_NOTHING version of user analysis routines
//
// Author : G.Dissertori
//          M.Hoerndl
//
// Modified by Dan Wallin 2000-08-14
//
////////////////////////////////////////////////////////////

#include <ospace/std/fstream>

#include "AlephExManager.h"
#include "Hbook.h"
#include "AlephVector.h"

extern ofstream fout;

static int Ngood = 0;
extern int Nev = 0;

static int nYcut = 10;

static float Ycut[10] = { 0.001, 0.0025, 0.004, 0.0055,
                   0.0085, 0.01, 0.025, 0.04, 0.08, 0.1 };
static float Yclo[11] = { 0., 0.0015, 0.003, 0.005,
            0.007, 0.009, 0.015, 0.04, 0.065, 0.09, 0.11};

////////////////////////////////////////////////////////////
//  User Init
////////////////////////////////////////////////////////////
void AlephExManager::UserInit(){

  Hbookb(10, "2jetrate as function of Ycut",nYcut,Yclo);
  Hbookb(11, "3jetrate as function of Ycut",nYcut,Yclo);
  Hbookb(12, "4jetrate as function of Ycut",nYcut,Yclo);
  Hbookb(13, "5jetrate as function of Ycut",nYcut,Yclo);

  Hbook1d(21,"Mom, 1st Jet", 50, 0., 50.);
  Hbook1d(22,"Mom, 2nd Jet", 50, 0., 50.);
  Hbook1d(23,"Mom, 3rd Jet", 50, 0., 50.);

  Hbook1d(31,"Energy, 1st Jet", 50, 0., 50.);
```

```
  Hbook1d(32,"Energy, 2nd Jet", 50, 0., 50.);
  Hbook1d(33,"Energy, 3rd Jet", 50, 0., 50.);

  Hbook1d(41,"Nch, 1st Jet", 50, 5., 30.);
  Hbook1d(42,"Nch, 2nd Jet", 50, 5., 30.);
  Hbook1d(43,"Nch, 3rd Jet", 50, 5., 30.);

  Hbook1d(51," psi 12", 100, 0., 3.141592);
  Hbook1d(52," psi 13", 100, 0., 3.141592);
  Hbook1d(53," psi 23", 100, 0., 3.141592);

  Hbook2d(100, "Nch of first jet as function of energy",
          50, 0., 50., 20, 0.5, 20.5);

  // book an ntuple
  int ntags = 3;
  char** mytags = new char*[ntags];
  mytags[0] = "var1";
  mytags[1] = "ecm1";
  mytags[2] = "mom";

  Hbookn(1000,"myNtuple",ntags,mytags);

  delete [] mytags;

  fout << endl << " =====> in UserInit " << endl;
}


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


////////////////////////////////////////////////////////////
//  User Event
////////////////////////////////////////////////////////////
void AlephExManager::UserEvent(AlphaBanks& bb){
  Nev++;

  typedef AlephVector<AlEflw *>::iterator ObjItr;
  // Stores pointers to all Eflw objects in the alev AlephVector
  AlephVector<AlEflw *> alev=bb.EflwPV();

  ObjItr it;

  for(it = alev.begin(); it < alev.end(); it++){
```

```
  int type = (*it)->getEfType();
  float costheta = (*it)->QCT();

  if (type<=3){
    int Ntpc;
    AlTrack* theAssTrack = (*it)->getTrack();
    if (theAssTrack==NULL){
      Ntpc = 0;
    }
    else{
      Ntpc = theAssTrack->NT();
    }
    float ptch = (*it)->QPT();
    float d0 = (*it) -> QDB();
    float z0 = (*it) -> QZB();

    if ( (d0>2.) || (z0>5.) || (fabs(costheta)>0.94)
                 || (ptch<0.2)){
      alev.looperase(it);
    }
    // all Eflow objects with type > 5 are removed
    if (type > 5) {
      alev.looperase(it);
    }
  }
}

// check if Nchgood and Echgood are big enough
int Nchgood = 0;
float Echgood = 0.;
for(it = alev.begin(); it < alev.end(); it++){
  int type = (*it)->getEfType();
  if (type <=3){
    Nchgood ++;
    Echgood += (*it)->QE();
  }
}

if ((Nchgood <= 0) || (Echgood <= 0)){
  fout << " ******** :-0 :-0  WARNING : Nev = " << Nev << endl;
}
else{
  //compute thrust axis with all good Eflows
  AlThrust theThrust = alev.AThrust();
  // Add the object to the AlphaBanks ObjectV
  bb.addObject(&theThrust);
```

```
Hep3Vector theThrAxis = theThrust.getThrustDirection();
float costhr = theThrAxis.cosTheta();

// final event selection
if ((Nchgood>=5) && (Echgood>=15.) && (fabs(costhr)<0.82)){
  // now he have a good event
  Ngood++;

  int jetr;
  HepLorentzVector jetmomentum[3];
  float multiplicity[3] = {0., 0., 0.};
  float interjetangle[3];
  for(int i=0; i<10; i++){
    // Call Durham algorithm
    AlephVector<AlJet> Jn = alev.DurhamJn(0, Ycut[i]);
    // Add the objects in the vector to the AlphaBanks ObjectV
    bb.addObject(Jn);

    jetr = Jn.size();

    fout << " jetr  = " << jetr << endl;
    if (jetr == 2) Hfill1d(10,Ycut[i]);
    if (jetr == 3) Hfill1d(11,Ycut[i]);
    if (jetr == 4) Hfill1d(12,Ycut[i]);
    if (jetr >= 5) Hfill1d(13,Ycut[i]);

    AlephVector<AlObject *> JnObjects[3];
    // take three jet events for Ycut = 0.01
    if (( i == 5) && ( jetr == 3 )){
      for(int j=0; j<3; j++){
        jetmomentum[j] = Jn[j].A4V();

        typedef AlphaBanks::AlObjectVec::iterator ObjItr3;
        // calculate the multiplicity
        for(ObjItr3 it1=Jn[j].getObjects().begin();
          int type = ((AlEflw*)(*it1))->getEfType();
          if (type <=3){
            multiplicity[j]++;
          }
        }
      }

      // store the jets so that the energy decreases from
      // jetmom[0] to jetmom[2]
      int ready = 0;
```

```
          while (ready == 0){
            ready = 1;
            for (int k = 0; k < 2; k++){
              for (int l=1; l < 3; l++){
                if (l != k){
                  if (jetmomentum[l].e() > jetmomentum[k].e()){
                    HepLorentzVector mom = jetmomentum[k];
                    jetmomentum[k] = jetmomentum[l];
                    jetmomentum[l] = mom;
                    float mult = multiplicity[k];
                    multiplicity[k] = multiplicity[l];
                    multiplicity[l] = mult;
                    ready = 0;
                  } // of if
                }  // of if
              }  // of for
            }  // of for
          } // of while

          // calculate the interjetangles
          interjetangle[0]=jetmomentum[0].
                           angle(Hep3Vector(jetmomentum[1]));
          interjetangle[1]=jetmomentum[0].
                           angle(Hep3Vector(jetmomentum[2]));
          interjetangle[2]=jetmomentum[1].
                           angle(Hep3Vector(jetmomentum[2]));

          // fill histograms
          for (int ih=0; ih<3; ih++){
            Hfill1d(20+ih+1,((Hep3Vector)jetmomentum[ih]).mag());
            Hfill1d(30+ih+1,jetmomentum[ih].e());
            Hfill1d(40+ih+1,(float)(multiplicity[ih]));
            Hfill1d(50+ih+1,interjetangle[ih]);
          }
          Hfill2d(100,jetmomentum[0].e(),multiplicity[0], 1.);
        }  // of if 3jet
      }  // of for over ycuts
    }  // of if good event
  } // end of check if Nch or Ech > 0
}


////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////
```

```
//////////////////////////////////////////////////////////
//  User Term
//////////////////////////////////////////////////////////
void AlephExManager::UserTerm(){
  fout << endl << " =====> in UserTerm " << endl;
  fout << endl << " Number of selected events = " << Ngood
       << endl << endl;
}
```

B1 - Profanalyzer

```
// A program that reads gprof files to make
// them suitable for graphs.

#include <iostream.h>
#include <fstream.h>
#include <string.h>
#include "formatlist.h"
#include "macrowriter.h"

int main(int argc, char *argv[]){
  char out1file[100]="result.kumac";
  char out2file[100]="result.dat";
  char infile[100];
  ifstream readfile;
  ofstream pawfile, datafile;
  char stringtemp[600];

  // Read main function arguments.
  if (argc<2 || argc>4){
    cerr << "Error: Wrong format of call " << endl;
    cerr << "Profanalyzer infile <pawfile> <datafile>, <>=optional"
         << endl;
    exit(EXIT_FAILURE);
  }
  strcpy(infile,argv[1]);
  // Replace the name of the out1file and out2file
  if (argc==4){
    strcpy(out1file,argv[2]);
    strcpy(out2file,argv[3]);
  }

  // Open file for reading
  readfile.open(infile);
  if(!readfile){
    cout << "Could not open file " << infile <<  endl;
    exit(EXIT_FAILURE);
  }

  // Skip lines that don't start with a number.
  // The getline statement can be dangerous...
  readfile >> ws;
  while(readfile.peek()<48 || readfile.peek()>57){
    readfile.getline(stringtemp,600);
    readfile >> ws;
```

```
  }

  // Open file for writing pawdata
  pawfile.open(out1file);
  if(!pawfile){
    cout << "Could not open file" << out1file << endl;
    exit(EXIT_FAILURE);
  }

  // Open file for writing data
  datafile.open(out2file);
  if(!datafile){
    cout << "Could not open file" << out2file << endl;
    exit(EXIT_FAILURE);
  }

  // Init a new list and a macro
  formatlist theList;
  macrowriter theMacro(&pawfile, &datafile);

  // Read the whole file
  while (theList.readelement(&readfile)){}

  // Call the macro
  theMacro.createmacro(theList);

  // Close read and write files
  readfile.close();
  pawfile.close();
  datafile.close();
  return 1;
}
```

B2 - macrowriter.h

```
// macrowriter.h

#ifndef __macrowriter__
#define __macrowriter__

#include <iostream.h>
#include <fstream.h>
#include "formatlist.h"

class macrowriter{
  private:
    ofstream *pawfile;
    ofstream *datafile;
  public:
    macrowriter(ofstream *, ofstream *);
    void createmacro(formatlist);
    void pawhistogram(formatlist, char);
    void writetotalruntime(formatlist);
    void writepercentage(formatlist, formatlist);
    void writedatafile(formatlist);
    ~macrowriter();
};

#endif
```

B3 - macrowriter.cc

```
// macrowriter.cc
#include "macrowriter.h"

macrowriter::macrowriter(ofstream *pf, ofstream *df){
  pawfile=pf;
  datafile=df;
}

void macrowriter::createmacro(formatlist initList){
  // These files contains data from all function calls
  formatlist flist(initList);
  formatlist clist(initList);

  // Get the f77 functions
  flist.f77function();

  // Get the C++ functions
  clist.cppfunction();

  // Print the total execution time to the datafile
  writetotalruntime(initList);

  // Print the percentage of C++ and f77 functions
  // to the datafile
  writepercentage(clist,flist);

  // Get the ten largest f77 and C++ functions respectively
  flist.getlargest(10);
  clist.getlargest(10);

  // Add the clist to the end of the flist
  flist.add(clist);

  // Create a PAW histogram file.
  // 't'=function exec time 'p' = percent each
  // function call takes
  pawhistogram(flist,'p');

  // Write important data to the datafile
  writedatafile(flist);

  // Returns a list with all elements with owntime
  // greater than a double
  //flist.greateq_owntime(0.005);
```

```
  // Returns a list with all elements with percent
  // greater than a double
  //flist.greateq_percent(0.3);

  // Returns a list with all C++ functions.
  //flist.cppfunction();

  // Returns a list with all f77 functions.
  //flist.f77function();

  // Returns a list of the int functions that take
  // the most time
  //flist.getlargest(10);

  // Print on the screen
  //flist.print();

  // Print to file
  //flist.printtofile(&pawfile);

  // Print the the total percentage of the list
  //cout << flist.gettotalpercent() << endl;

  // Print the total execution time of the list
  //cout << flist.gettotalowntime() << endl;

}

// Write the PAW histogram file
void macrowriter::pawhistogram(formatlist f, char c){
  int elems=f.getnumberofelements();
  *pawfile << "Macro gprof_res " << "col=1" << endl;
  *pawfile << "******************************" << endl;
  *pawfile << "** Plots of profiling results" << endl;
  *pawfile << "exec ~/.pawlogon" << endl << endl;
  *pawfile << "** create a vector" << endl;
  *pawfile << "v/cr x(" << elems << ") r ";
  // Depending on choice of parameter...
  switch(c){
  case 'p':
    f.printpercenttofile(pawfile);
    break;
  case 't':
    f.printpercenttofile(pawfile);
    break;
```

```
  }
  *pawfile << endl << endl;
  *pawfile << "** Create a histogram from the vector " << endl;
  *pawfile << "h/crea/1d 10 'Function usage' "<< elems << " "
           << 0.5 << " " << elems+0.5 << endl;
  *pawfile << "h/put_vect/cont 10 x" << endl << endl;
  *pawfile << "** Rename the histogram with function names" << endl;
  *pawfile << "LABEL 1 " << elems << " ";
  f.printnamestofile(pawfile,600);
  *pawfile << endl << endl;
  *pawfile << "SET NDVX " << elems << ".17" << endl << endl;
  *pawfile << "** Marginal size" << endl;
  *pawfile << "SET YMGL 8" << endl << endl;
  *pawfile << "** plot the histogram" << endl;
  *pawfile << "opt utit" << endl;
  *pawfile << "h/pl 10 b" << endl << endl;
  *pawfile << "RETURN" << endl;
}

void macrowriter::writetotalruntime(formatlist f){
  *datafile << "The total runtime" << endl;
  *datafile << "Runtime in seconds: " << f.gettotalowntime()
            << endl << endl;
}

void macrowriter::writepercentage(formatlist c, formatlist f){
  *datafile << "The total percent of running time" << endl;
  *datafile << "C++ functions: " << c.gettotalpercent() << endl;
  *datafile << "f77 functions: " << f.gettotalpercent() << endl
            << endl;
}

void macrowriter::writedatafile(formatlist f){
  // The order array gives the order in which data is to be
  // printed. The first letter tells the first information to
  // be printed and so on. Possible choices are:
  // p=percent of total exec time
  // a=accumulative seconds
  // o=exec time
  // n=number of calls
  // m=us per call
  // t=total us per call
  // l=computer language
  // If the array is left blank nothing will be printed.
  // The name of the function is always printed last.
```

```
  char order[9]="ponl";
  *datafile << "Data generated from gprof file" << endl << endl;
  f.printtofile(datafile,order);

}

macrowriter::~macrowriter(){
}
```

B4 - formatlist.h

```
// formatlist.h

#ifndef __formatlist__
#define __formatlist__

#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include "format.h"

class formatlist{
  private:
    link first;

  public:
    formatlist();
    formatlist(const formatlist& f);
    formatlist& operator=(const formatlist& );
    ~formatlist();
    int readelement(ifstream *);
    void addelement(link);
    void add(formatlist);
    void reorder();
    int getnumberofelements();
    void greateq_owntime(double);
    void greateq_percent(double);
    void cppfunction();
    void f77function();
    void getlargest(int);
    double gettotalpercent();
    double gettotalowntime();
    void print();
    void printtofile(ofstream *, char[]);
    void printpercenttofile(ofstream *);
    void printowntimetofile(ofstream *);
    void printnamestofile(ofstream *, int);
};

#endif
```

B5 - formatlist.cc

```
// formatlist.cc
#include "formatlist.h"

formatlist::formatlist(){
  first=0;
}

// Copyconstructor
formatlist::formatlist(const formatlist& f){
  first=0;
  for (link l=f.first;l;l=l->next) {
    first=new format(l->percent,l->accumulative,l->owntime,
              l->num_of_calls,l->own_us_call,l->tot_us_call,
              l->name,l->language,first);
  }
  this->reorder();
}

// operator constructor
formatlist& formatlist::operator=(const formatlist& f){
  if(this!=&f){
    first=0;
    for (link l=f.first;l;l=l->next) {
      first=new format(l->percent,l->accumulative,l->owntime,
                l->num_of_calls,l->own_us_call,l->tot_us_call,
                l->name,l->language,first);
    }
  }
  this->reorder();
  return *this;
}

// Reads the next line from the file
int formatlist::readelement(ifstream *readfile){
  double p,a,ot,ouc,tuc;
  int noc;
  char nm[600]="";
  complanguage l;
  char ctemp;

  // Look at the next character
  ctemp=readfile->peek();
  // Check if it is a number
  if (ctemp>=48 && ctemp<=57){
```

```
  *readfile >> p >> a >> ot;
  // Take away all blanks
  while (readfile->peek()==' '){
    readfile->get();
  }
  // Look at the next character
  ctemp=readfile->peek();

  // Check if the next character is a number. In that case,
  // read three numbers, else set them to errorvalues.
  if (ctemp>=48 && ctemp<=57){
    *readfile >> noc >> ouc >> tuc;
  }
  else{
    noc=-1;
    ouc=-1;
    tuc=-1;
  }

  // Remove whitespaces
  *readfile >> ws;

  // Read the name of the function.
  readfile->getline(nm,600);

  // Decide whether the function is a f77 or a C++-function.
  l=f77;
  for (int i=0;i<600;i++){
    if (nm[i]==')'){
      l=cpp;
      break;
    }
  }

  // Remove whitespaces...
  *readfile >> ws;

  // Create a new element...
  first=new format(p,a,ot,noc,ouc,tuc,nm,l,first);

  // Reading successful...
  return 1;
}
else{
  // Reading failed...
  return 0;
```

```
  }
}

// Adds an element to a formatlist
void formatlist::addelement(link p){
  first=new format(p->percent,p->accumulative,p->owntime,
            p->num_of_calls,p->own_us_call,p->tot_us_call,
            p->name,p->language,first);
}

// Adds a formatlist to another formatlist
void formatlist::add(formatlist f){
  link p;
  f.reorder();
  for(p=f.first; p; p=p->next){
    addelement(p);
  }
  f.first=0;
}

// Reorders a formatlist
void formatlist::reorder(){
  link p;
  formatlist f;
  for(p=first; p; p=p->next){
    f.addelement(p);
  }
  first=f.first;
  f.first=0;
}

// Returns the number of elements in the list
int formatlist::getnumberofelements(){
  link p;
  int i=0;
  for(p=first; p; p=p->next){
    i++;
  }
  return i;
}


// Returns a formatlist of elements with owntime greater
// or equal to time.
void formatlist::greateq_owntime(double time){
  link p;
```

```
  formatlist f;
  for(p=first; p; p=p->next){
    if ((p->owntime)>time)
      f.addelement(p);
  }
  f.reorder();
  first=f.first;
  f.first=0;
}

// Returns a formatlist of elements with percent greater
// or equal to perc.
void formatlist::greateq_percent(double perc){
  link p;
  formatlist f;
  for(p=first; p; p=p->next){
    if ((p->percent)>perc)
      f.addelement(p);
  }
  f.reorder();
  first=f.first;
  f.first=0;
}

// Returns a formatlist of all C++-functions
void formatlist::cppfunction(){
  link p;
  formatlist f;
  for(p=first; p; p=p->next){
    if (p->language==cpp){
      f.addelement(p);
    }
  }
  f.reorder();
  first=f.first;
  f.first=0;
}

// Returns a formatlist of all Fortran 77-functions
void formatlist::f77function(){
  link p;
  formatlist f;
  for(p=first; p; p=p->next){
    if (p->language==f77)
      f.addelement(p);
  }
```

```
  f.reorder();
  first=f.first;
  f.first=0;
}

// get the num largest elements of the list, with greatest
// owntime or percent
void formatlist::getlargest(int num){
  link p;
  formatlist f;
  this->reorder();
  p=first;
  for(int i=0; i<num && p ; i++){
    f.addelement(p);
    p=p->next;
  }
  first=f.first;
  f.first=0;
}

// Get the total percentage of the functions of the list
double formatlist::gettotalpercent(){
  link p;
  double perc=0;
  for(p=first;p;p=p->next){
    perc += p->percent;
  }
  return perc;
}

// Get the total owntime of the functions of the list
double formatlist::gettotalowntime(){
  link p;
  double ot=0;
  for(p=first;p;p=p->next){
    ot += p->owntime;
  }
  return ot;
}

// Print the whole list
void formatlist::print(){
  this->reorder();
  for (link l=first;l;l=l->next) {
    l->print();
  }
```

```
  this->reorder();
}

// Print the whole list to a file
void formatlist::printtofile(ofstream *writefile, char order[]){
  this->reorder();
  int j,i=0;
  // Find the nummer of columns to be printed
  while(order[i]=='p' || order[i]=='a' || order[i]=='o' ||
        order[i]=='n' || order[i]=='m' || order[i]=='t' ||
        order[i]=='l'){
    i++;
  }
  // Print an explanations to each event
  for(j=0;j<i;j++){
    *writefile << "[" << j+1 << "] = ";
    switch(order[j]){
    case('p'):
      *writefile << "Percent of total time";
      break;
    case('a'):
      *writefile << "Cumulative time in
                    seconds (usually not very helpful)";
      break;
    case('o'):
      *writefile << "Execution time of function in seconds";
      break;
    case('n'):
      *writefile << "Number of calls to this function";
      break;
    case('m'):
      *writefile << "The average time in microseconds per call";
      break;
    case('t'):
      *writefile << "The average time in microseconds spent in
                    this function and its descendents per call";
      break;
    case('l'):
      *writefile << "Computer language";
    default:
       break;
    }
    *writefile << endl;
  }
  *writefile << "[" << j+1 << "] = Name of function" << endl;
  // Print the columns
```

```
  *writefile << "-----------------------------" << endl;
  for(j=1;j<=i+1;j++){
    *writefile << "[" << j << "]" << "\t";
  }
  *writefile << endl;
  //  Print the data
  for (link l=first;l;l=l->next) {
    l->printtofile(writefile,order);
  }
  this->reorder();
}

// Print the percentage of usage to a file
void formatlist::printpercenttofile(ofstream *writefile){
  this->reorder();
  for (link l=first;l;l=l->next) {
    l->printpercenttofile(writefile);
  }
  this->reorder();
}

// Print the percentage of usage to a file
void formatlist::printowntimetofile(ofstream *writefile){
  this->reorder();
  for (link l=first;l;l=l->next) {
    l->printowntimetofile(writefile);
  }
  this->reorder();
}

// Print the names of the files to a file.
// n gives the maximum length.
void formatlist::printnamestofile(ofstream *writefile, int n){
  this->reorder();
  for (link l=first;l;l=l->next) {
    l->printnamestofile(writefile, n);
  }
  this->reorder();
}

// Close input and output files and delete elements recursively...
formatlist::~formatlist(){
  delete first;
}
```

B6 - format.h

```
// format.h

#ifndef __format__
#define __format__

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

// A function can either be written in fortran 77 or in C++
enum complanguage{f77,cpp};

class format{
  private:
    double percent;
    double accumulative;
    double owntime;
    int num_of_calls;
    double own_us_call;
    double tot_us_call;
    char name[600];
    complanguage language;
    format *next;

  public:
    // Constructor
    format(double p, double a, double ot, int noc, double ouc,
           double tuc, char *nm, complanguage(l), format *n=0);
    // Copyconstructor
    format(const format* l);
    // Print a single element
    void print();
    // Print a single element to a file
    void printtofile(ofstream*,char[]);
    // Print the percentages to a file
    void printpercenttofile(ofstream *);
    // Print the owntimes to a file
    void printowntimetofile(ofstream *);
    // Print the names of the functions to a file.
    void printnamestofile(ofstream *, int);
    // Destructor
    ~format();

    // Make the private fields available for the formatlist
```

```
    friend class formatlist;
};

typedef format *link;

#endif
```

B7 - format.cc

```
// format.cc

#include "format.h"

format::format(double p, double a, double ot, int noc,
               double ouc, double tuc, char *nm,
               complanguage(l), format *n=0){
  percent=p;
  accumulative=a;
  owntime=ot;
  num_of_calls=noc;
  own_us_call=ouc;
  tot_us_call=tuc;
  strcpy(name,nm);
  language=l;
  next=n;
}

// Copyconstructor
format::format(const format* l){
  percent=l->percent;
  accumulative=l->accumulative;
  owntime=l->owntime;
  num_of_calls=l->num_of_calls;
  own_us_call=l->own_us_call;
  tot_us_call=l->tot_us_call;
  strcpy(name,l->name);
  language=l->language;
  next=l->next;
}

void format::print(){
  cout << percent << " " << accumulative << " " << owntime
       << " " << num_of_calls << " " << own_us_call << " "
       << tot_us_call << " " << name << " ";
  if (language==f77)
    cout << "f77" << endl;
  else
    cout << "C++" << endl;
}

// Prints the data to a file in the order given in the order-array.
void format::printtofile(ofstream *outfile,char order[]){
  for (int i=0; i<7; i++){
```

```
    switch(order[i]){
    case('p'):
      *outfile << percent << "\t";
      break;
    case('a'):
      *outfile << accumulative << "\t";
      break;
    case('o'):
      *outfile << owntime << "\t";
      break;
    case('n'):
      *outfile << num_of_calls << "\t";
      break;
    case('m'):
      *outfile << own_us_call << "\t";
      break;
    case('t'):
      *outfile << tot_us_call << "\t";
      break;
    case('l'):
      if (language==f77)
        *outfile << "f77"  << "\t";
      else
        *outfile << "C++"  << "\t";
      break;
    default:
      break;
    }
  }
  *outfile << name << "\t";
  *outfile << endl;
}

// Print the total percentage to a file
void format::printpercenttofile(ofstream *outfile){
  *outfile << percent << " ";
}

// Print the owntimes to a file
void format::printowntimetofile(ofstream *outfile){
  *outfile << owntime << " ";
}

// Prints the names of the functions.
// Shortens the names to n characters
void format::printnamestofile(ofstream *outfile, int n){
```

```
  int i=0;
  *outfile << "'";
  while(i<n && name[i]!='\0'){
    outfile->put(name[i]);
    i++;
  }
  if (i<n)
    *outfile << "' ";
  else
    *outfile << "...' ";
}

format::~format(){
  delete next;
}
```