Nesneye Yönelmek

veya
sadece formülleri bilgisayarın anlayacağı dile çevirmeyi
bilen birinin C++ kullanma yöntemleri

Gökhan Ünel

Önce fizik ...

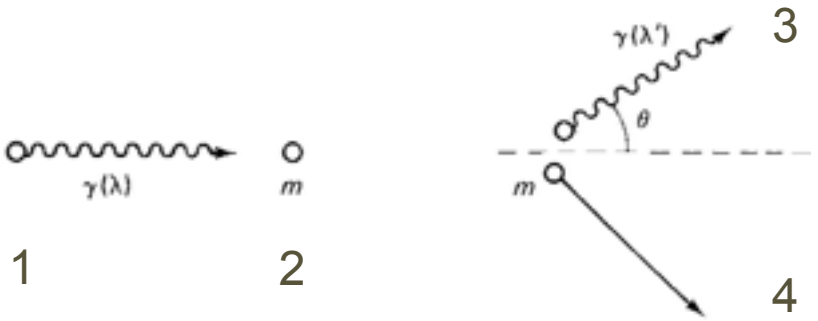
$$
\begin{aligned}
x^{0\prime} &= \gamma(x^0 - \beta x^1) \\
x^{1\prime} &= \gamma(x^1 - \beta x^0) \\
x^{2\prime} &= x^2 \\
x^{3\prime} &= x^3
\end{aligned}
$$

matris olarak yazarsak

$$
\begin{bmatrix} x^{0\prime} \\ x^{1\prime} \\ x^{2\prime} \\ x^{3\prime} \end{bmatrix}
=
\begin{bmatrix} \gamma & -\gamma\beta & 0 & 0 \\ -\gamma\beta & \gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\begin{bmatrix} x^0 \\ x^1 \\ x^2 \\ x^3 \end{bmatrix}
$$

- Başka neyi böyle yazabilirim?
  - ➡ (Enerji ve momentumlar), (hızlar)
- Bunlar ne böyle?
  - ➡ 4 elemanı olan ve Lorentz dönüşümlerine uyan cisimlere Lorentz vektörü denir. X ve/veya P
- S <==> S' dönüşümü altında değişmez kalan var mı?
  - ➡ evet bu: $(x^0)^2 - (x^1)^2 - (x^2)^2 - (x^3)^2$ veya bu: $-(x^0)^2 + (x^1)^2 + (x^2)^2 + (x^3)^2$
  - ➡ Lorentz Değişmezi

compton saçılması



λ' ve λ arasındaki ilişkiyi saçılma açısı cinsinden bulunuz. c=1

✓ Foton için: $E=hc/\lambda$ $E=p$

➡ enerji, momentum(x) momentum (y) : 3 denklem

- θ, φ, p_4 üç bilinmeyen, birbirinin içine koy çöz, bol sabırlar dilerim.

➡ Veya LV kullanalım, ilgilenmediğimiz 4ü tek başına bırakalım

➡ $(P_1+P_2-P_3)^2 = (P_4)^2$

➡ $P_1^2+P_2^2 +2P_1\cdot P_2 -2P_1\cdot P_3 -2P_2\cdot P_3= m^2$

➡ $0 + m^2 +2E_1m -2(E_1E_3-E_1E_3\cos\theta) -2(mE_3) = m^2$

➡ $E_1m = E_1E_3(1-\cos\theta) + mE_3$

➡ $E_3(m +E_1(1-\cos\theta)) = E_1m$ $E_1=h/\lambda$ $E_3=h/\lambda'$

➡ $\lambda(m + (1-\cos\theta)\, h/\lambda) = m\lambda'$

➡ $\lambda(+ (1-\cos\theta)\, h/m\lambda) = \lambda' = \lambda + (1-\cos\theta)\, h/m$ ➡ m'nin compton dalga boyu

bu işleri bilgisayara yaptırsak

- Kötü yöntem
  - float e_tot= e1+e2 ;
  - float px_tot=p1x+p2x ;
  - ....
- İyi yöntem
  - aynı int, float, gibi lorentz vektörü de tanımlı olsa
  - LorentzVector P = P1 + P1 diyebilsem ne güzel olur.
- Bunu struct (C) ile veya common block(fortran) ile yapamaz mıyım?
  - hayır: bunlar durağan kavramlar, yani sadece veri saklamak için.
- Yeni bir sınıf ve nesne yapmalıyım. Lorentz Vector

Root ile örnek

- Root içinde LorentzVector sınıfı var.
  - ➡ int a=3;
  - ➡ TLorentzVector p1(-0.1, 0.1, 3500, 3501);
    - ilk degerler px, py, pz, E olarak verilir
  - ➡ p1'in kütlesi ne?
    - √(3501^2 - 3500^2) = 83.672
  - ➡ TLorentzVector sinifi bize bunu kolayca hesaplar:
    - cout << p1.M()<<endl;
    - 83.6719

- ayrıntılar
  - ➡ http://root.cern.ch/root/html/TLorentzVector.html

- 10 dakka ödev molası.
  - ➡ yukardaki örneği yapın
  - ➡ ayrıca p1'in eta, phi, pT degerlerini de yazdirin.

parçacık sınıfı

- Bir parçacık tanımlamak için LV yeterli mi?
  - ➡haaayııııır!
  - ➡hani ya bunun yükü (q=?)
- O zaman ben kendim yeni bir sınıf yazayım!
  - ➡böylece c++ class vs işlerini de öğrenirim
- Deneysel bir parçacık için neler lazim?
  - ➡LV olmalı
  - ➡yük olmalı
  - ➡parçacık kendini ekrana yazabilmeyi bilmeli
  - ➡Ptcone ve Etcone bilgisi olmalı
    - iç algıçta parçacık etrafında ne kadar pT ölçülmüş?
    - kalorimetrede parçacık etrafında ne kadar ET ölçülmüş?
  - ➡vb...

C++ da sınıf tanımlama

- 2 kütük lazım
  - ➡HPFparticle.h
    - burada bildiriler olacak.
    - Ör: dump() diye bir işlev var.
  - ➡HPFparticle.C
    - burada tanımlamalar, önceden bildirilen işlerin nasıl yapıldığı anlatılacak.
    - Ör: dump() işlevi bunu yapar.
- Bazı Nesneye Yönelik Programlama kavramları lazım.
  - ➡Sınıf yapmak
    - neden sınıf yapmak gerektiğini öğrendik
  - ➡Özel veriyi saklamak
    - sınıfların her bildiklerini paylaş**ma**maları programları daha gürbüz yapar
  - ➡Miras
    - bunu ilerde göreceğiz
  - ➡Çokbiçimlilik - ekyük taşıma
    - bunu zaten biliyoruz. + işlemi iki sayıyı toplar ama 2 LV de toplayabilirim.

Örnek ile devam

- hpfParticle.h - bildiriler

```
#include "TLorentzVector.h"
class hpfParticle {

 public:
        hpfParticle();
        hpfParticle(TLorentzVector);
        hpfParticle(TLorentzVector, int);
       ~hpfParticle();

        void dump();
        int setCharge( int);
        int setEtCone( double );
        int setPtCone( double );
        int setTlv( TLorentzVector );


        int q()              { return p_charge; }
        double EtCone()      { return p_et_cone; }
        double PtCone()      { return p_pt_cone; }
        TLorentzVector lv() { return p_lvector; }

private:
       int p_charge;
       double p_et_cone;
       double p_pt_cone;
       TLorentzVector p_lvector;
};
```

sınıfı başlatmak için 3 seçenek

sınıfı kapatmak için

sınıfını ekrana yazmak için

yükünü vermek için

Et ve pT vermek için

LV değişmek için

yükünü almak için

Et ve pT almak için

LV almak için

sınıfın özel değişkenleri

hpfParticle.cpp - tanımlamalar, açıklamalar ...1

sınıfı başlatmak, seçenek 1: hiç ön bilgi vermeden

```
#include "hpfParticle.h"
#include <iostream>

hpfParticle:: hpfParticle ( ){
  p_et_cone = -999; p_pt_cone = -999; p_charge=-999; // not initialized
  p_lvector=TLorentzVector (-1,-1,-1,-1);
}

hpfParticle:: hpfParticle (TLorentzVector lv){
  p_et_cone = -999; p_pt_cone = -999; p_charge=-999; // not initialized
  p_lvector=lv;
}

hpfParticle:: ~hpfParticle() {}

hpfParticle:: hpfParticle (TLorentzVector lv, int q){
  p_et_cone = -999; p_pt_cone = -999;
  p_charge=q;
  p_lvector=lv;
}

int hpfParticle:: setCharge (int q){
 p_charge=q;
 return 0;
}
```

sınıfı başlatmak, seçenek 2: LV vererek

kapatırken ek iş yapma

sınıfı başlatmak, seçenek 3: LV ve Q vererek

yük böyler verilir veya değişilir

hpfParticle.cpp - tanımlamalar, açıklamalar ...2

```
int hpfParticle:: setCharge (int q){
 p_charge=q;
 return 0;
}

int hpfParticle:: setEtCone (double iso){
 p_et_cone=iso;
 return 0;
}

int hpfParticle:: setPtCone (double iso){
 p_pt_cone=iso;
 return 0;
}

int hpfParticle:: setTlv (TLorentzVector lv){
 p_lvector=lv;
 return 0;
}

void hpfParticle:: dump (){
 std::cout << "Px="<<p_lvector.Px()<< "  Py="<<p_lvector.Py()
           << "  Pz="<<p_lvector.Pz()<< "  E="<<p_lvector.E()<<std::endl;
}
```

EtCone böyle verilir.

EtCone böyle verilir.

LV böyle verilir.

ekrana böyle yazalım.

root'a bunları tanıtmak için

- Derleyelim

```
Gokhans-MacBook-Pro:ders ngu$ root -l -L  hpfParticle.cpp+
root [0]
Processing hpfParticle.cpp+...
Info in <TUnixSystem::ACLiC>: creating shared library /users/ngu/ders/./hpfParticle_cpp.so
(class hpfParticle)140515964457280
```

- Kullanalim

```
root [1] TLorentzVector v1(-0.1, 0.1, 3500, 3501);
root [2] TLorentzVector v2(-0.1, 0.1, -3500, 3501);
root [3] hpfParticle p1(v1,-1)
root [4] hpfParticle p2(v2,-1)
root [5] p2.SetCharge(+1)
(int)0
root [7] p1.q()
(int)(-1)
root [8] p2.q()
(int)1
```

```
root [10] hpfParticle p3
root [12] TLorentzVector alv(0,0,12,13);
root [13] p3.setTlv(alv);
root [14] p3.lv().M()
(const Double_t)5.00000000000000000e+00
```

```
root [17] p3.dump()
Px=0  Py=0  Pz=12  E=13
```

- 15 dakka ödev molası.
  - örnekleri siz de yapın

kopyalayıp yapıştırın diye .h

```
#include "TLorentzVector.h"
class hpfParticle {

 public:
        hpfParticle();
        hpfParticle(TLorentzVector);
        hpfParticle(TLorentzVector, int);
       ~hpfParticle();

        void dump();
        int setCharge( int);
        int setEtCone( double );
        int setPtCone( double );
        int setTlv( TLorentzVector );


        int q()              { return p_charge; }
        double EtCone()      { return p_et_cone; }
        double PtCone()      { return p_pt_cone; }
        TLorentzVector lv() { return p_lvector; }

private:
       int p_charge;
       double p_et_cone;
       double p_pt_cone;
       TLorentzVector p_lvector;
};
```

kopyalayıp yapıştırın diye .cpp

```
#include "hpfParticle.h"
#include <iostream>

hpfParticle:: hpfParticle ( ){
  p_et_cone = -999; p_pt_cone = -999; p_charge=-999; // not initialized
  p_lvector=TLorentzVector (-1,-1,-1,-1);
}

hpfParticle:: hpfParticle (TLorentzVector lv){
  p_et_cone = -999; p_pt_cone = -999; p_charge=-999; // not initialized
  p_lvector=lv;
}

hpfParticle:: ~hpfParticle() {}

hpfParticle:: hpfParticle (TLorentzVector lv, int q){
  p_et_cone = -999; p_pt_cone = -999;
  p_charge=q;
  p_lvector=lv;
}

int hpfParticle:: setCharge (int q){
 p_charge=q;
 return 0;
}

int hpfParticle:: setEtCone (double iso){
 p_et_cone=iso;
 return 0;
}

int hpfParticle:: setPtCone (double iso){
 p_pt_cone=iso;
 return 0;
}

int hpfParticle:: setTlv (TLorentzVector lv){
 p_lvector=lv;
 return 0;
}

void hpfParticle:: dump (){
 std::cout << "Px="<<p_lvector.Px()<< "  Py="<<p_lvector.Py()
           << "  Pz="<<p_lvector.Pz()<< "  E="<<p_lvector.E()<<std::endl;
}
```

peki ya bir muon?

- Özellikler
  - ➡ Bir parçacıktır. -tamam-
  - ➡ Ek bilgi taşır: olayı tetiklemiş mi?

- O halde hpfParticle'dan alacağı mirasla tanımlanır
  - ➡ hpfMuon.h

Tetikleme bilgisini vermek için hem bildiri hem tanimlama ayni yerde

```
#include "hpfParticle.h"


class hpfMuon : public hpfParticle {
 public:
        hpfMuon( ): hpfParticle(){};
        hpfMuon( TLorentzVector lv): hpfParticle(lv){};
        ~hpfMuon(){};
        int setMuInTrigger(bool v) {p_topmutrigdec=v; return 0;}
        bool   MuInTrigger() { return p_topmutrigdec; }

private:
       bool   p_topmutrigdec;

};
```

Muon sınıfının kendine özel değişkeni

muon'u kullanayım:

```
Gokhans-MacBook-Pro:ders ngu$ root -l -L  hpfParticle.cpp+
root [0]
Processing hpfParticle.cpp+...
(class hpfParticle)140205099640720
root [1] .L hpfMuon.h+
Info in <TUnixSystem::ACLiC>: creating shared library /users/ngu/ders/./hpfMuon_h.so
root [2] hpfMuon m1
root [3] m1.dump()
Px=-1  Py=-1  Pz=-1  E=-1
```

kopyalayıp yapıştırın diye

```
#include "hpfParticle.h"


class hpfMuon : public hpfParticle {
 public:
        hpfMuon( ): hpfParticle(){};
        hpfMuon( TLorentzVector lv): hpfParticle(lv){};
        ~hpfMuon(){};
        int setMuInTrigger(bool v) {p_topmutrigdec=v; return 0;}
        bool   MuInTrigger() { return p_topmutrigdec; }

private:
       bool   p_topmutrigdec;

};
```

Bir de electron yapalım

- Bir parçacıktır. -tamam-
- Ek bilgi taşır:
  - tetikleyen elektron mu ?, elektron izinin eta ve phi değerleri (sadece iç algıçtan gelen bilgi)
- hpfElectron.h

```
#include "hpfParticle.h"

class hpfElectron : public hpfParticle {
 public:
        hpfElectron( ): hpfParticle( ){};
        hpfElectron( TLorentzVector lv): hpfParticle(lv){};

        int setElTriggerMatch(bool v) { p_eltriggermatch=v; return 0;}
        int setTrkEta(double v) { p_trketa=v; return 0;}
        int setTrkPhi(double v) { p_trkphi=v; return 0;}

        bool ElTriggerMatch() { return p_eltriggermatch; }
        double TrkEta() { return p_trketa; }
        double TrkPhi() { return p_trkphi; }


private:
       bool p_eltriggermatch;
       double p_trketa;
       double p_trkphi;

};
```

derleyip kullanmak için

- derleme

```
Gokhans-MacBook-Pro:ders ngu$ root -l -L  hpfParticle.cpp+
root [0]
Processing hpfParticle.cpp+...
(class hpfParticle)140672730982512
root [1] .L hpfMuon.h+
root [2] .L hpfElectron.h+
Info in <TUnixSystem::ACLiC>: creating shared library /users/ngu/ders/./hpfElectron_h.so
```

- Kullanma

```
root [4] TLorentzVector alv(-0.1, 0., 5, 13);
root [5] hpf

hpfParticle
hpfMuon
hpfElectron
```

```
root [7] hpfElectron e1(alv);
```

```
root [11] e1.lv().Eta()
(const Double_t)4.60527017099166613e+00
root [12] e1.lv().Phi()
(const Double_t)3.14159265358979312e+00
```

kopyalayıp yapıştırın diye

```
#include "hpfParticle.h"

class hpfElectron : public hpfParticle {
 public:
        hpfElectron( ): hpfParticle( ){};
        hpfElectron( TLorentzVector lv):
hpfParticle(lv){};

        int setElTriggerMatch(bool v)
{ p_eltriggermatch=v; return 0;}
        int setTrkEta(double v) { p_trketa=v;
return 0;}
        int setTrkPhi(double v) { p_trkphi=v;
return 0;}

        bool ElTriggerMatch() { return
p_eltriggermatch; }
        double TrkEta() { return p_trketa; }
        double TrkPhi() { return p_trkphi; }


private:
       bool p_eltriggermatch;
       double p_trketa;
       double p_trkphi;

};
```

- 15 dakka ödev molası.
  - örneği tekrarlayın
  - 1 elektron 1 pozitron tanımlayın
    - dump() ile ekrana yazdırın.

işaretçiler

- int *p ;
  - ➡ burada p : işaretçi *p: işaretçinin gösterdiği sayı

- new komutu hafızada yer ayırır
  - ➡ ayırdığı yer *p'nin cinsi kadardır.
    - p = new int → 32 bit ayırır
  - ➡ hpfMuon *m1 ;
    - m1 = new hpfMuon → hpfMuon nesnesi kadar ayırır.

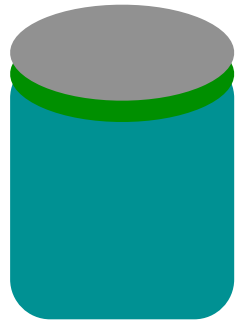
- delete komutu ayrılan yeri siler
  - ➡ değişkenle işim bitince, silerim. Yoksa programımın kullandığı hafıza gittikçe artar.

önceki örnek - işaretçiler ile

- aynı işlerin işaretçiler kullanılarak yapılması

```
root [13] hpfMuon *m1;
root [14] m1=new hpfMuon(alv);
root [15] m1->lv().M();
root [16] cout << m1->lv().M()<<endl;
11.9996
root [17] m1->dump()
Px=-0.1  Py=0  Pz=5  E=13
```

- Benzetme yapıyorum
  - ➡ işaretçi → kapak
  - ➡ "new" → kavanoz
  - ➡ "delete" → kavanozu kır
    - kırılan kavananozun içi yok



```
root [20] delete m1
root [21] m1->dump()
Error: illegal pointer to class object m1 0x0 3116  (tmpfile):1
*** Interpreter error recovered ***
```

- Neden ?
  - ➡ kavanozun camı, yani hafıza az. İşim bitince cam kumbarasına geri atmalıyım. Doğanın dengesini koru.

dikkat ve not

- camı kırmadan kapağa yeni kavanoz takmayın:
  - ➡ yoksa eski kavanozu kaybedersiniz. sonunda hafıza biter -> segfault olur.

```
          te = new hpfElectron(alv);
         *te=*ae;
          te->setTlv(alv_up);
....
          te = new hpfElectron(alv);
         *te=*ae;
          te->setTlv(alv_down);


          delete (ae);
```

- Diyelim elimde bir çok elektron. muon vb var.
  - ➡ kaç tane kullanacağımı bilmiyorum.
  - ➡ o halde ne yapmalıyım? Bunu mu? **HAYIR !!!**
    - const int max_muons = 1000;
    - hpfMuon [max_muons] ;
  - ➡ çalışır ama hafızayı gerekli gereksiz hep kullanır.

vector

http://www.cplusplus.com/reference/stl/vector/

- STL vector'ler bize sayısını son anda belirleyeceğimiz kadar nesne ile çalışma olanağı verir.

```
{
#include <vector>
vector<int> sayi_dizisi;

int gelgec;

gelgec=2;
sayi_dizisi.push_back(gelgec);
gelgec=32;
sayi_dizisi.push_back(gelgec);
gelgec=12;
sayi_dizisi.push_back(gelgec);
gelgec=17;
sayi_dizisi.push_back(gelgec);

cout << "dizideki sayi adedi:"<<sayi_dizisi.size()<<endl;
cout << "bu sayilar :";
for (int i=0; i<sayi_dizisi.size(); i++) {
 cout << sayi_dizisi.at(i)<<" ";
}
cout << endl;
}
```

vector değişkeni

diziye sayı ekleme

dizinin boyu

dizideki i. sayı

vector

comparison operators
vector::vector
vector::~vector
member functions:
vector::assign
vector::at
vector::back
vector::begin
vector::capacity
vector::clear
vector::empty
vector::end
vector::erase
vector::front
vector::get_allocator
vector::insert
vector::max_size
vector::operator=
vector::operator[]
vector::pop_back
vector::push_back
vector::rbegin
vector::rend
vector::reserve
vector::resize
vector::size
vector::swap

- 10 dakika ödev molası: örneğin aynısını yapın

İki bilgiyi birleştirelim

```
#include <vector>
void v2() {
gSystem->Load("hpfParticle_cpp.so");
gSystem->Load("hpfMuon.h");
gSystem->Load("hpfElectron.h");

/*
gROOT->LoadMacro("hpfParticle.cpp");
gROOT->LoadMacro("hpfMuon.h");
gROOT->LoadMacro("hpfElectron.h");
*/
gROOT->LoadMacro("is.C");

vector<hpfMuon> muons;
vector<hpfElectron> electrons;
hpfElectron *an_e;
hpfMuon     *an_m;
TLorentzVector  alv;

     alv.SetPtEtaPhiM( 31.2, -2.1 , 0.6 ,   0.106 );
     an_m=new hpfMuon(alv);
     an_m->setEtCone( 21.1 );
     an_m->setPtCone( 13.4 );
     an_m->setCharge( -1 );
     an_m->setMuInTrigger( 0 );
     muons.push_back(*an_m);
     delete an_m;

     alv.SetPtEtaPhiM( 91.2, 1.8 , -0.6 ,   0.106 );
     an_m=new hpfMuon(alv);
     an_m->setEtCone( 1.1 );
     an_m->setPtCone( 3.4 );
     an_m->setCharge( +1 );
     an_m->setMuInTrigger( 1 );
     muons.push_back(*an_m);
     delete an_m;

  cout << "hazirlik tamam, hesaba geciyoruz."<<endl;
  hesapyap ( muons, electrons);
}
```

- ayrıca işaretçilerle çalışalım ve daha derli-toplu olsun
  - ➡ bir işlev elektron, muon listesini hazırlasın, bir başkası kullansın.
  - ➡ v2.C solda, is.C yi siz yazın.

böyle yüklenince arıza veriyor.

is.C elektron ve muon sayılarını ekrana koysun, ayrıca parçacıkların momentum-enerji değerlerini de. (bende 8 satırda oldu)

muon bilgileri başa bir kütükte veya başka bir biçimde olabilir.

Asıl İşi yapacak kısım, muonların elektronların doldurulma ayrıntılarından bağımsız ve oradaki değişikliklerden korunmalı.

- 10 dakika ödev molası: is.C yazın, hesapyap işlevini tanımlasın.