Klassen – Einleitung

Ziel von C++ ist der Support objektorientierter Programmierung (OOP). Dabei handelt es sich um ein Konzept zur Strukturierung von Programmen bei dem Programmlogik (Funktionalität) und Programmzustand (Datenstrukturen) vereinigt werden. Die Vereinigung wird Objekt genannt.

- Objekte werden im Programm erzeugt und vernichtet.
- Objekte haben zur Laufzeit des Programms definierte Zustände
- OOP erlaubt die Kommunikation zwischen Objekten

OOP wird in C++ mit Hilfe von Klassen realisiert. Sie vereinigen Datenstrukturen und Funktionen zur Veränderung der Daten.

- Anschaulich: Klassen sind Vorlagen und Konstruktionspläne aus denen zur Laufzeit Objekte erzeugt werden (Instanzen).
- Alorithmen die auf den Datenstrukturen von Klassen operieren heißen Methoden
- Klassen kapseln zusammengehörige Daten und Funktionen und unterstützen Datenzugriff nur über spezielle Funktionen. Methoden die Objekte erzeugen oder zerstören werden als Konstruktoren oder Destruktoren bezeichnet.

Klassen – ein Beispiel

Ziel von C++ ist der Support von OOP. Dabei spielen Klassen eine wichtige Rolle. Sie vereinigen Datenstrukturen und Methoden zur Änderung der Daten.

- Betrachte ein Beispiel: Klasse von Vierervektoren zur Beschreibung von Teilchenreaktionen

Vierervektor:

$$(E, \vec{P}) = (E, p_x, p_y, p_z)$$

```
class FourVector
{
   public:
   Definiere Funktionen, die die Elemente des Vektors
   füllen
   Definiere Funktionen, die den Impuls, Energie und
   invariante Masse zurückgeben.

   private:
   definiere Daten, die zum Vektor gehören
} vect;
int main(){
FourVector Elektron, Proton;
…..
```

Klassenname, identifiziert die Klasse

Klassenmitglieder
- Datenstrukturen
- Funktionen/Methoden

Optional Objektnamen

Erzeuge Instanzen von FourVector

Klassen – ein Beispiel

Ziel von C++ ist der Support von OOP. Dabei spielen Klassen eine wichtige Rolle. Sie vereinigen Datenstrukturen und Methoden zur Änderung der Daten.

- Betrachte ein Beispiel: Klasse von Vierervektoren zur Beschreibung von Teilchenreaktionen

Zugriffsrechte werde explizit mit Hilfe von Schlüsselwörtern gewährt

```
class FourVector
{
   Auf alle Klassenmitglieder die hier stehen
   gibt es nur von Mitgliedern Zugriff
   Public:
   Diese Klassenmitglieder können vom gesamten
   Programm benutzt werden
   private:
   Nur Mitgliedern ist der Zugriff erlaubt
};

int main(){
FourVector Elektron, Proton;
…..
```

Klassen – ein Beispiel

- Betrachte ein Beispiel: Klasse von Vierervektoren zur Beschreibung von Teilchenreaktionen

```
class FourVector{
   public:
   void setE(double e) {E = e;}
   void setP(double x, double y, double z){
             px = x; py = y ; pz = z;}
   double getE(void)const {return E;}
   double getP(void)const {
      return sqrt(px*px+py*py+pz*pz);}
   private:
   double E, px, py, pz;
} ;

int main(){
  FourVector Elektron, Proton;
  Elektron.setE(100.);
  Proton.setP(10., 20., 50.);
  Proton.getP();
  return 0 ;}
```

Methoden erlauben Zugriff auf die Datenstruktur.

FourVectorClass_0.cc

Datenstruktur ist nicht zugänglich

Zwei Instanzen von FourVector werden erzeugt.

Mit dem Punkt Operator werden die Methoden erreicht.

Klassen – ein Beispiel

- Betrachte ein Beispiel: Klasse von Vierervektoren zur Beschreibung von Teilchenreaktionen

```
class FourVector{
   public:
   void setE(double e);
   void setP(double x, double y, double z);
   double getE(void)const;
   double getP(void)const;
   private:
   double E, px, py, pz;
} ;
FourVector::setE(double e){
   E = e;
}
double FourVector::getE(void)const{
   return E;
}
```

Scope operator

Klassendefinition werden im Header File untergebracht.

Die Methoden können auch ausserhalb der Klassendefinition definiert werden. Auf die Klasse wird mit Klassenname:: bezug genommen.

…….

FourVectorClass_I.cc

Klassen – ein Beispiel

- Konstruktor und Destruktor einer Klasse

```
class FourVector{
   public:
   FourVector(void){ E=0.;px=0.;py=0.;pz=0.}
   ~FourVector(void) {  }
   void setE(double e) {E = e;}
   void setP(double x, double y, double z){
              px = x; py = y ; pz = z;}
   double getE(void) {return E;}
   double getP(void) {return sqrt(px*px+py*py+pz*pz);}
   private:
   double E, px, py, pz;
} ;
int main(){
  FourVector Elektron, Proton;
```

Konstruktor, code wird bei der Instanzierung gerufen

Destruktor

Zwei Instanzen von FourVector werden erzeugt.

Beide sind durch den Konstruktor initialisiert.

Klassen – ein Beispiel

- Konstruktor einer Klasse

  - Kein Rückgabetyp, Argumente können void oder eine Variableliste sein
  - Konstruktor hat den Namen der Klasse und dient der Initialisierung des Objektes

```
class FourVector{
   Public:
      FourVector(double e,double x,double y, double z);
      FourVector();
      ….
   Private:
      double E, px, py, pz;
};
FourVector::FourVector(double e,double x,double y,
double z): E(e),Px(x),Py(y),Pz(z){}
FourVector::FourVector(){ E=0.; px=0.; py=0.; pz=0.}

……..
FourVector Pion;
FourVector Kaon(5.,1.0,1.5,19.);
```

Initialisierung der Variablen

Synthax

Zuweisung der Variablen, um zu initialisieren

Initialisierung von array geht nur über Zuweisung !

Klassen – ein Beispiel

- Konstruktor einer Klasse
  - Im Header können Default Parameter angegeben werden.
  - Konstruktor kann ueberladen werden
  - Kopierkonstruktur wird vom System erzeugt, erstellt ein Objekt der Klasse anhand eines vorhandenen Objektes. Der Parameter ist immer eine Referenz auf ein konstantes Objekt der selben Klasse.

```
class FourVector{
   Public:
   ~FourVector(){}; Destruktor
   FourVector(double e=0.,double x,double y, double z);
   FourVector(FourVector const& FourVectorObjekt);
      ….
   Private:
      double E, px, py, pz;
};
```

- Destruktor einer Klasse
  - Einer pro Klasse
  - Ist fuer Aufräumarbeiten zuständig

Klassen – ein Beispiel

- Eine Klasse wird instanziert, indem die Klassenbezeichnung und ein Name angegeben wird. Der Zugriff auf Member Funktionen und Variablen erfolgt über den Operator „ . “ Für Pointer auf Klassen wird der Operator „ → “ verwendet. Er besorgt die Dereferenzierung und den Memberzugriff.

```
class FourVector
{
  ….
}

int main() {
FourVector Kaon;
FourVector *Pion;
…..
Kaon.setE(20.);
…..
Pion->GetMomentum();
……
}
```

Wir ergänzen die Übungsklasse FourVector um einen Destruktor und mehrere Konstruktoren.

Arbeitsvorschlag:

Schreiben Sie ein Programm ausgehend von FourVectorClass_I.cc, dass 3 Konstruktoren zur Verfügung stellt; den Konstruktor ohne Argumente, mit einer Variablen soll nur die Energie übergeben werden, mit 3 Variablen sollen alle 3 Impulskomponenten gegeben werden. Benutzen Sie dabei Initialisierungen.

Die Konstruktoren und auch der Destruktor sollen ausgeben, wenn Sie benutzt wurden. Geben Sie die Initialisierungswerte aus.

Einem neuen FourVector Objekt ein existierendes zuweisen. Welcher Konstruktor wird benutzt?
Können wir eine Swap Funktion schreiben. Wird unser Konstruktor benutzt oder ein Kopierkonstruktor des Systems. Was wird benötigt?

Erzeugen Sie in einer Funktion mit new ein FourVector Objekt? Wird das Objekt zerstoert?

FourVectorClass_II.cc

this Zeiger

- this Pointer einer Klasse

Jede nicht statische Methode hat eine versteckte Zeigervariable, in der die Adresse des Objektes gespeichert wird, die die Methode aufgerufen hat. this ist ein C++ Schlüsselwort. Die “interne“ Deklaration kann man sich so vorstellen:

```
myClass * const this = &object;
```

Der this Zeiger wird benutzt wenn eine Adresse der eigenen Klasse zurückgegeben wird.
In vielen Fällen wird this automatisch ergänzt. In nicht eindeutigen Fällen z.B. lokale Variablen heissen wie die Klassen Variablen, sollte this hinzugefügt werden.

```
class FourVector {
…….
FourVector & FourVector::compare (FourVector & v){
  If ( condition )
       return v ;    // v ist Argument
  else
       return *this ;  // aufrufendes Objekt
}
```

Operatoren

- Operatoren in Klassen

  Klassen definieren neue Typen, die in C++ genutzt werden. Typ Definitionen werden aber nicht nur in Form von Konstrukten und Zuweisungen verwendet, sondern müssen auch Operatoren (z.B + - ..) verstehen.

```
class FourVector{
    double E, px, py, pz;
    public:
    FourVector(){ E=0.;px=0.;py=0.;pz=0.}
    void setE(double e) {this->E = e;}
    …..
    double getP(void){return sqrt(Px*Px+Py*Py+Pz*Pz);}
    FourVector operator+(const FourVector& v) {
      FourVector vec;
      vec.E=this→E+v.E;  vec.Px=this→vec.Px + v.Px;
      vec.Py=this→Py+v.Py;
      vec.Pz=this->Pz+v.Pz;
      return vec;
    }
    …….
```

Erlaubt das Summieren von zwei Instanzen von FourVector

Das Schlüsselwort this ist ein pointer auf das aktuell benutzte Objekt.

Wir ergänzen die Übungsklasse FourVector um Operatoren

Arbeitsvorschlag:

Schreiben Sie ein Programm ausgehend von FourVectorClass_II.cc , dass die Operatoren + und += für die Klasse FourVector zur Verfügung stellt.
Geben Sie Energie und Impuls der Summen zweier FourVector Objekte mit den get Methoden aus.
In ähnlicher weise kann die skalare Multiplikation implementiert werden.

Erzeugen Sie ein FourVector Objekt mit new und weisen Sie die Summe von 2 FourVector Objekten zu. Geben Sie wieder Energie und Impuls der Summe mit den get Methoden aus.

FourVectorClass_III.cc

Vererbung

Klassen können die Implementierung (Methoden und Variablen) von einer anderen Klasse übernehmen (Vererbung). Altes Verhalten kann umgeändert und neues hinzugefügt werden, die grundlegende Schnittstellen bleiben jedoch gleich (Polymorphie). Das Ableiten einer neuen Klasse (Sub- oder Unterklasse) von einer bereits bestehenden (Ober- oder Basis-) Klasse wird folgendermaßen erreicht:

```
class BaseClass {
 …..
};
class SubClass: ZugriffsOperator BaseClass {
 …..
};
```

public	In SubClass stehen fast alle member von BaseClass zur Verfügung
private	Alle member von BaseClass sind privat
protected	Alle member von BaseClass sind protected

- Zugriffskontrolle auf die BaseClass Variablen von SubClass aus

  Variable im private Bereich können von SubClass aus nicht erreicht werden. Lösung: verschiebe private BaseClass Variable in protected

Vererbung

Beispiel mit unserer FourVector Klasse, die als Basis Klasse die ThreeVector Klasse hat :

```
class ThreeVector {
  double Px,Py,Pz;
  public:
  ThreeVector(double x, double y, double z):
                                    Px(x),Py(y),Pz(z){}
  …….
};
class FourVector: public ThreeVector {
  double E ;
  public:
  FourVector(double e, double x, double y, double z):
                          E(e), ThreeVector(x,y,z) {}
  …….
};
```

inherit.cc

Funktionen - Overloading

Funktionen können gleichzeitig mit den selben Namen definiert werden, wenn Sie eine unterschiedliche Anzahl von Parametern besitzen oder die Parameter von unterschiedlichem Typ sind.

- Overloading

overloading.cc

```
…….
class stdOutputData
{
public:
  void print(int j){cout << "Write integer: " << j << endl;}
  void print(double x){cout << "Write double: " << x << endl;}
  void print(char* c){cout << "Write character: " << c << endl;}
};
…….
stdOutputData p;
int k = 5 ;    double f = 2.3 ;     char c[256] = "Hi there" ;
// Write print to print integer ,  float or character
p.print(k);
p.print(f);
p.print(c);
…..
```