3.1. Lớp

❑ a. Định nghĩa lớp

- Lớp được định nghĩa theo mẫu sau:

```
class  <tên_lớp>
{
  [quyền truy nhập:]
        <khai báo các thành phần dữ liệu của lớp>
  [quyền truy nhập:]
        <khai báo các thành phần hàm của lớp>
};
```

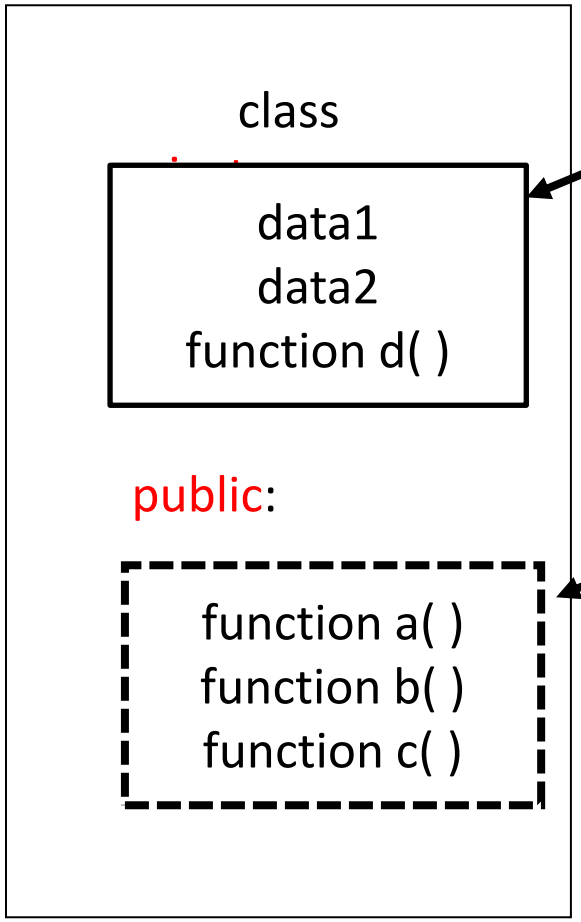
Trong đó:

- **<tên_lớp>**: Do người dùng đặt, tuân theo các qui tắc về tên
- **Ví dụ**: SinhVien, NGUOI, Hoa_Don, phanso, Ma_Tran…

[quyền truy nhập:]

- Là khả năng truy nhập thành phần dữ liệu, có thể là private, hoặc public, hoặc protected, ngầm định là private:
- private: Các thành phần private chỉ được sử dụng bên trong lớp (trong thân các phương thức của lớp).
- public: Các thành phần public được sử dụng ở cả bên trong lẫn bên ngoài lớp.
- protected: Các thành phần protected được sử dụng trong lớp đó và các lớp con kế thừa.

quyền truy nhập



Phần được khai báo với từ khóa private chỉ được truy nhập bởi các phương thức của cùng class

Phần được khai báo với từ khóa public có thể được truy nhập tại bất kỳ nơi nào trong chương trình

Thông thường, thành phần dữ liệu (member data) là private, và các hàm thành phần (member functions) là public

- **Thành phần của lớp:** có thể là
  - Thành phần dữ liệu (member data)
  - Phương thức (hoặc hàm thành phần – member function).
- **Khai báo thành phần dữ liệu:**

<kiểu dữ liệu > <tên_thành_phần>;

- ***Chú ý***: không được khởi tạo giá trị ban đầu
- ***Ví dụ:***

```
char  hoten[30];
int   namsinh;
float   diem;
```

- **Khai báo hàm thành phần:**
- **Cách 1**: Khai báo bên trong lớp và định nghĩa ở bên ngoài lớp

```
<kiểu trả về > <tên lớp>::<tên_hàm>([đối sô])
{
        // <thân hàm>
}
```

- **Cách 2*:*** định nghĩa ngay ở bên trong lớp.
- **Ví dụ 3.1*:*** Xây dựng lớp điểm trên mặt phẳng có tọa độ (x,y), các phương thức nhập điểm, in ra điểm, di chuyển điểm.

```
#include <iostream.h>
#include <conio.h>
class   diem
{
   private:
      float    x, y;
   public:
      void  nhap_diem();
      void  in_diem()   //Định nghĩa ở bên trong lớp
      {
          cout<<"Diem (" << x << "," << y << ")";
      }
      void  di_chuyen(float  dx, float  dy);
};
//Định nghĩa các hàm thành phần ở bên ngoài lớp
void  diem :: nhap_diem()
{
   cout << "Nhap hoanh do, tung do cua diem: ";
   cin >> x >> y;
}
//-------------
void   diem :: di_chuyen(float   dx, float   dy)
{
   x += dx;
   y += dy;
}
```

```
main()
{
    diem a;
    a.nhap_diem();
    a.in_diem();
    float   dx, dy;
    cout<<"\ndx = "; cin>>dx;
    cout<<"dy = "; cin>>dy;
    a.di_chuyen(dx, dy);
    a.in_diem();
}
```

- **Ví dụ 3.2**: Xây dựng lớp phân số có:
  - Dữ liệu: tử số, mẫu số.
  - Phương thức: nhập phân số, tối giản phân số, in phân số.

```
#include <iostream.h>
#include <conio.h>
#include <math.h>
class  Phanso
{
   private:
     int  ts, ms;
   public:
     void NhapPS();
     void Toigian();
     void InPS();
};
```

```
//Định nghĩa hàm InPS ở bên
ngoài lớp
void   Phanso :: InPS()
  {
     cout << ts << "/" << ms;
  }
// Định nghĩa hàm NhapPS ở bên
ngoài lớp
void  Phanso::NhapPS()
{
   cout<<"Tu so: "; cin>>ts;
   do
   {
     cout<<"Mau so: "; cin>>ms;
   } while (ms == 0);
}
```

```
//Định nghĩa hàm Toigian
void
Phanso::Toigian_PS()
{
   int a, b;
   if(ts!=0)
   {
      a = abs(ts);
      b = abs(ms);
   } while (a != b)
   if (a > b)   a -= b;
   else   b -= a;
       ts = ts/a;
       ms = ms/a;
}
```

```
// Ham main
main()
{
    Phanso  p;
    p.NhapPS();
    p.InPS();
    p.Toigian();
    p.InPS();
}
```

b. Đối tượng

- **Cú pháp khai báo đối tượng:**

<tên_lớp> <tên_đối_tượng>;

- **Ví dụ**: Phanso a, b;

c. Truy xuất thành phần

- **Truy xuất thành phần dữ liệu:**

<tên_đối_tượng>.<tên_tp_dữ_liệu>;

  - Nếu là con trỏ:

<tên_con_trỏ> -> <tên_tp_dữ liệu>;

- **Truy xuất thành phần hàm:**

<tên_đối_tượng>.<tên_hàm>([ds đối số]);

- **Ví dụ:** a.NhapPS(); a.InPS();
  - Với con trỏ: <tên_con_trỏ> -> <tên_hàm>([đối số]);
- **Ví dụ:**

```
Phanso *p;
p ->NhapPS(); p -> InPS();
```

3.2. Các phương thức

- **Một đối tượng thường có 4 kiểu phương thức cơ bản:**
  - Các phương thức khởi tạo (Constructor)
  - Các phương thức truy vấn (Queries)
  - Các phương thức cập nhập (Updates)
  - Các phương thức hủy (Destructor)
- **a. Hàm khởi tạo (constructor)**
  - Khai báo hàm tạo:

```
<tên_lớp>([ds tham số]);
```

  - Định nghĩa hàm tạo ở ngoài lớp:

```
<tên_lớp>::<tên_lớp>([ds tham số])
{
    //thân hàm
}
```

- **Ví dụ:** Hàm tạo:

```
diem(float  tx, float  ty);
```

- **Định nghĩa ở bên ngoài lớp:**

```
diem::diem(float   tx, float   ty)
{
      x = tx ;
      y = ty;
}
```

- **Như vậy hàm khởi tạo:**
  - Có với mọi lớp
  - Tên hàm giống tên lớp
  - Không có kiểu nên không cần khai báo kiểu trả về
  - Không có giá trị trả về
  - Nếu không xây dựng thì chương trình tự động sinh hàm khởi tạo mặc định
  - Được gọi tự động khi khai báo thể hiện của lớp

- **Một số hàm khởi tạo:**
  - Hàm khởi tạo mặc định (default constructor): Hàm tạo mặc định do chương trình dịch cung cấp khi trong khai báo lớp không có định nghĩa hàm tạo nào.
  - Hàm tạo có các giá trị ngầm định cho các tham số.
- **Ví dụ:**

```
diem(float  tx = 0, float  ty = 0)
{
     x = tx; y = ty;
};
```

- **Hàm khởi tạo sao chép (copy constructor)**

Khai báo:

<tên_lớp> (const <tên_lớp> &<tên_tham_số>);

Đối tượng mới sẽ là bản sao của đối tượng đã có.

- **Ví dụ:**

```
diem (const  diem  &p)
{
          x = p.x ; y = p.y;
}
```

b. Hàm hủy - Destructor

- **Khai báo:**

~ <tên_lớp>( );

- **Chức năng:** Hủy bỏ, giải phóng các đối tượng khi nó hết phạm vi tồn tại
- **Như vậy hàm hủy:**
  - Không có đối số
  - Không có giá trị trả về
  - Không định nghĩa lại
  - Trùng tên với lớp và có dấu ~ ở trước
  - Thực hiện một số công việc trước khi hệ thống giải phóng bộ nhớ
  - Chương trình dịch tự động sinh hàm hủy mặc định

3.3. Mảng và con trỏ của đối tượng

- **Khai báo mảng các đối tượng:**

<tên_lớp> <tên_mảng>[số phần tử];

Ví dụ: SinhVien sv[50]; Phanso p[8];

- **Khai báo con trỏ đối tượng:**

<tên_lớp> * <tên_con_trỏ>;

- **Ví dụ:** SinhVien *sv1 ; Phanso *p1;

3.4. Hàm bạn và lớp bạn

a. Hàm bạn

Hàm bạn của một lớp là hàm không phải là thành phần của lớp, nhưng có khả năng truy xuất đến mọi thành phần của đối tượng

Cú pháp khai báo hàm bạn:

friend <kiểu trả về> <tên hàm>(tham số);

Sau đó định nghĩa hàm ở ngoài lớp như các hàm tự do khác.

- **Ví dụ 3.3:** Xây dựng hàm tự do kiểm tra xem hai điểm có trùng nhau, là bạn của lớp **diem**:

```
#include <iostream>
#include <conio.h>
using namespace std;
class   diem
{
   private:
      float  x, y;
   public:
      diem(float   tx = 0, float   ty = 0)
      {
         x = tx; y = ty;
      }
      friend   int  trung(diem,  diem);
};
int   trung(diem  p, diem  q)
{
   if(p.x == q.x  &&  p.y == q.y)
      return  1;
   else  return  0;
}
//-------------------
```

```
//-------------------
void main( )
{
   diem  a(1,0), b(1), c;
   if(trung(a,b)) cout<<"a trung voi b";
   else  cout<<"\na khong trung voi b";
   if(trung(a,c))cout<<"\na trung voi c";
   else  cout<<"\na khong trung voi c";
   system("pause");
}
```

Kết quả??

- **Nhận xét:**
  - Hàm bạn không phải là hàm thành viên nên không bị ảnh hưởng của từ khoá truy xuất
  - Không hạn chế số lượng hàm bạn
  - Hàm bạn của một lớp có thể là hàm tự do
  - Hàm bạn của một lớp có thể là hàm thành phần của một lớp khác
  - Một hàm có thể là bạn của nhiều lớp khác nhau.

b. Lớp bạn:

- Lớp A là lớp bạn của lớp B nếu trong B có chứa khai báo:

friend class A;

- Nếu A là lớp bạn của B thì mọi hàm thành phần của A sẽ trở thành hàm bạn của B.

- **Ví dụ 3.4**: Chương trình xây dựng lớp ma trận vuông, lớp vector là bạn (friend) của nhau, tính tích vô hướng của 2 vector, tích của 2 ma trận, tích ma trận với vector, tích của vector với ma trận:

```
#include  <iostream.h>
#include  <stdio.h>
#include  <conio.h>
#define   MAX  50
```

```
class  ma_tran;
class  vec_to;
class  ma_tran
{
  int  n;
  float  a[MAX][MAX];
public:
  friend  class  vec_to;
  ma_tran() { n=0; }
  ma_tran(int  m);
  ~ma_tran() { };
  void nhap_mt();
  void in_mt();
  ma_tran   tich(const
ma_tran  &);
  vec_to   tich(const
vec_to  &);
};
```

```
class  vec_to
{
  int   n;
  float  x[MAX];
public:
  friend  class  ma_tran;
  vec_to() { n=0; }
  vec_to(int   m);
  ~vec_to() { };
  void  nhap_vt();
  void  in_vt();
  vec_to  tich(const   ma_tran
&);
  float  tich(const   vec_to
&);
};
```

```
ma_tran::ma_tran(int   m)
{
 n = m;
 for(int  i=0; i<n; i++)
    for(int  j=0; j<n; j++)
       a[i][j] = 0;
}
//-----------------------------------
void  ma_tran::nhap_mt()
{    for(int   i=0; i<n; i++)
          for(int   j=0; j<n; j++)
       {   cout<<"Nhap phan tu hang "<<i+1<<" cot
"<<j+1<<": ";
         cin>>a[i][j];
       }
}
```

```
vec_to::vec_to(int  m)
{
   n = m;
   for(int  i=0; i<n; i++) x[i] = 0;
}
//-------------------------
void   vec_to::nhap_vt()
{
   for(int  i=0; i<n; i++)
{
 cout<<"Nhap phan tu thu "<<i+1<<": ";
 cin>>x[i];
}
}
```

```
void  ma_tran::in_mt()
{
   for(int   i=0; i<n; i++)
      {
 for(int    j=0; j<n; j++) printf("%8.2f",a[i][j]);
 printf("\n");
      }
}
//-------------------------
void  vec_to::in_vt()
{
   for(int   i=0; i<n; i++) printf("%8.2f",x[i]);
}
```

```
ma_tran   ma_tran::tich(const  ma_tran   &b)
{
   ma_tran c;
   c.n = n;
   for(int   i=0; i<n; i++)
      for(int   j=0; j<n; j++)
 {
    c.a[i][j] = 0;
    for(int  k=0; k<n; k++)
       c.a[i][j] += a[i][k]*b.a[k][j];
 }
   return c;
}
```

```
vec_to ma_tran::tich(const   vec_to   &y)
{
   vec_to z;
   int i,j;
   z.n = n;
   for(i=0; i<n; i++)
 {
    z.x[i] = 0;
    for(j=0; j<n; j++)
       z.x[i] += a[i][j]*y.x[j];
 }
   return z;
}
```

```
vec_to   vec_to::tich(const  ma_tran   &b)
{
   vec_to  z;
   int i,j;
   for(j=0; j<n; j++)
      {
 z.x[j] = 0;
 for(i=0; i<n; i++)
    z.x[j] += b.a[i][j]*x[i];
      }
   z.n = n;
   return   z;
}
```

```
float   vec_to::tich(const
vec_to   &y)
{
   float  tg = 0;
   for(int i=0; i<n; i++)
      tg += x[i] * y.x[i];
   return  tg;
}
```

```
void main()
{   int  n;
    clrscr();       //hoặc system("cls") với #include
<stdlib.h>
   do
      {  cout<<"Nhap cap ma tran va vecto: ";
 cin >> n;
      } while (n <= 0 || n>MAX);
   ma_tran  a(n), b(n), c(n);
   vec_to  x(n), y(n);
   cout<<"Nhap ma tran A:\n"; a.nhap_mt();
   cout<<"\nNhap ma tran B:\n"; b.nhap_mt();
   cout<<"\nNhap ma tran C:\n"; c.nhap_mt();
   cout<<"\nNhap vecto X:\n"; x.nhap_vt();
   cout<<"\nNhap vecto Y:\n"; y.nhap_vt();
```

```
 ma_tran  d = a.tich(b);
   vec_to  u = a.tich(x);
   vec_to  v = y.tich(c);
   float   s = x.tich(y);
   clrscr();
   cout<<"\nMa tran A:\n"; a.in_mt();
   cout<<"\nMa tran B:\n"; b.in_mt();
   cout<<"\nMa tran C:\n"; c.in_mt();
   cout<<"\nVecto X:\n"; x.in_vt();
   cout<<"\n\nVecto Y:\n"; y.in_vt();
   cout<<"\n\nMa tran tich D = A*B:\n"; d.in_mt();
   cout<<"\nVecto tich U = A*X:\n"; u.in_vt();
   cout<<"\n\nVecto tich V = Y*C:\n"; v.in_vt();
   cout<<"\n\nTich vo huong X*Y = : "<< s;
   getch();  // hoặc system("pause") với #include
<stdlib.h>
}
```

```
 ma_tran  d = a.tich(b);
   vec_to  u = a.tich(x);
   vec_to  v = y.tich(c);
   float   s = x.tich(y);
   clrscr();
   cout<<"\nMa tran A:\n"; a.in_mt();
   cout<<"\nMa tran B:\n"; b.in_mt();
   cout<<"\nMa tran C:\n"; c.in_mt();
   cout<<"\nVecto X:\n"; x.in_vt();
   cout<<"\n\nVecto Y:\n"; y.in_vt();
   cout<<"\n\nMa tran tich D = A*B:\n"; d.in_mt();
   cout<<"\nVecto tich U = A*X:\n"; u.in_vt();
   cout<<"\n\nVecto tich V = Y*C:\n"; v.in_vt();
   cout<<"\n\nTich vo huong X*Y = : "<< s;
   getch();  // hoặc system("pause") với #include
<stdlib.h>
}
```

3.5. Con trỏ this

- Con trỏ **this** được dùng để xác định địa chỉ của đối tượng dùng làm tham số ngầm định cho hàm thành phần. Như vậy có thể truy cập đến các thành phần của đối tượng gọi hàm thành phần gián tiếp thông qua **this**.
- **Ví dụ:**

```
void  diem :: nhap_diem( )
{
cout << "Nhap hoanh do va tung do cua diem: "
       cin >> x >> y;
}
```

- Các thuộc tính viết trong phương thức trên được hiểu là thuộc một đối tượng do con trỏ **this** trỏ tới.

- Như vậy phương thức **nhap_diem( )** có thể viết một cách tường minh như sau:

```
void  diem::nhap_diem()
{
   cout << "Nhap hoanh do va tung do cua diem: "
   cin  >>  this -> x  >>  this -> y;
}
```

3.5. Thành phần tĩnh

a. Dữ liệu tĩnh

- Là thành phần dữ liệu của lớp nhưng không gắn cụ thể với đối tượng nào
- Dùng chung cho toàn bộ lớp
- Các đối tượng của lớp đều dùng chung thành phần tĩnh này

❑ **Khai báo:**

static <kiểu dữ liệu> <tên thành phần>;

❑ **Ví dụ 3.5:**

Tạo thành phần dữ liệu đếm các phân số trong lớp Phanso.

```
#include <iostream.h>
#include <conio.h>
class   Phanso
{       int   ts, ms;
        static   int   dem;
```

```
public:
    Phanso(int   m = 0, int   n = 1)
        {
        ts = m;
        ms = n;
        dem++;
        }
    static  int   So_PS()
        {  return   dem;  }
};
//-------------
int   Phanso :: dem = 0;   //Khởi tạo giá trị
```

```
void   main()
{
   Phanso   p(1,2), q(3,4), r;
   clrscr();
   cout<<"So cac phan so la: " <<Phanso::So_PS();
   cout<<"So cac phan so la: " <<r.So_PS();
   getch();
}
```

- **Truy xuất thành phần dữ liệu tĩnh:**
  - Theo đối tượng, VD: r.So_PS();
  - Theo phương thức, VD: Phanso :: So_PS();

❑ **Chú ý:**

- Thành phần dữ liệu tĩnh tồn tại ngay khi chưa có đối tượng nào.
- Phải được khởi tạo trước khi đối tượng phát sinh
- Phải khởi tạo ngoài mọi hàm theo cú pháp:

<kiểu dl> <tên lớp>::<tên t/phần d/liệu> = <giá trị>;

- Ví dụ: int Phanso :: dem = 0;

b. Phương thức tĩnh

- Là hàm thành phần của lớp nhưng không gắn với đối tượng cụ thể nào
- Dùng để thao tác chung cho lớp
- Trong thân hàm không có đối tượng ẩn

❑ **Khai báo:**

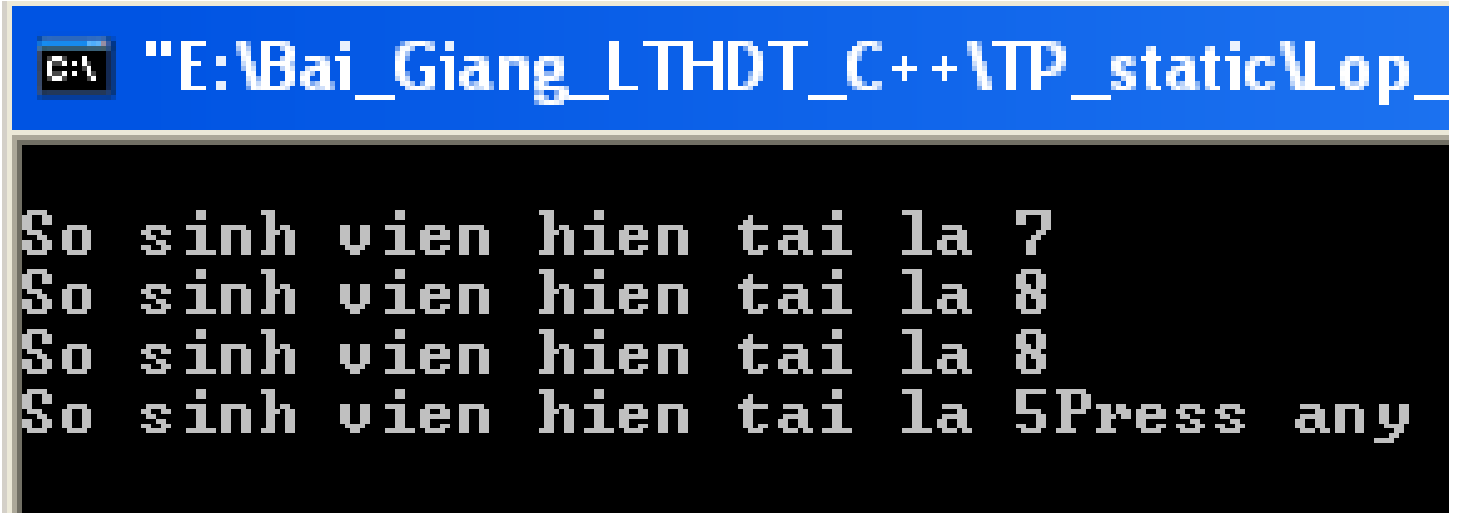
static <kiểu d/liệu trả về> <tên hàm> (tham số);

❑ Ví dụ 3.6:

Xây dựng lớp sinh viên có thành phần tĩnh là số sinh viên của lớp, phương thức tĩnh in ra số sinh viên hiện có.

```
#include <iostream.h>
class   lop_sv
{
   static  int  so_sv;
public:
   lop_sv() {so_sv++;}                  //Hàm tạo
   ~lop_sv() {so_sv--;}                 //Hàm hủy
   static void  in_so_sv()
{ cout<<"\nSo sinh vien hien tai la " << so_sv; }
};
int   lop_sv::so_sv = 0;
```

```
main()
{   lop_sv    a[5], b, c;
    a[0].in_so_sv();
    lop_sv  *d = new  lop_sv ;
    a[1].in_so_sv();  d -> in_so_sv();
    b.~lop_sv();  c.~lop_sv();  a[4].~lop_sv();
    a[2].in_so_sv();
}
```



3.6. Đối tượng hằng

a. Đối tượng hằng:

Một đối tượng có thể được khai báo là hằng bằng cách dùng từ khóa const.

- **Ví dụ:**

const diem d = diem(0, 0);

b. Phương thức hằng:

Là hàm thành phần của lớp nhưng không có khả năng thay đổi thành phần dữ liệu trong đối tượng.

- **Khai báo:**

<kiểu d/liệu trả về> <tên hàm> (tham số) const ;

- **Định nghĩa:**

<kiểu d/liệu trả về> <tên lớp>::<tên hàm> (tham số) const
{ //thân hàm }

- **VD:**

```
void Phanso::InPS()  const
     {   cout  <<  ts  <<   "/"  <<  ms;  }
```

3.7. Thành phần đối tượng

Thành phần đối tượng là thành phần dữ liệu của lớp có kiểu là một lớp khác.

❑ **Khai báo:**

<tên lớp> <tên thành phần dữ liệu> ;

❑ **Ví dụ:**

Thành phần *ngayvt* của lớp *hoadon* là đối tượng của lớp *ngay*.

```
class  ngay
{
private:
   int  ng, th, nm;
public:
   void  nhap_ngay();
   void  in_ngay();
};
```

```
class  hoadon
{
   private:
      char  mavt[5];
      char  tenvt[30];
      int  loai;
      ngay  ngayvt;
      float  soluong, dongia, thanhtien;
   public:
      void nhap_hd();
      void in_hd();
};
```

Bài tập chương 3

Bài 1. Xây dựng lớp phân số gồm các thành phần:

- Dữ liệu: tử số, mẫu số
- Phương thức: nhập, in, tối giản, so sánh nhỏ hơn giữa 2 phân số

Hàm main:

- Nhập mảng có n phân số (n<=10)
- Sắp xếp mảng phân số theo thứ tự giảm dần
- In mảng sau khi xếp

Bài 2. Xây dựng lớp số phức gồm các thành phần:

- Dữ liệu: phần thực, phần ảo
- Phương thức: nhập, in, tính trị tuyệt đối của số phức, tổng, hiệu 2 số phức.

Hàm main:

- Nhập 2 số phức
- Tính và in tổng, hiệu hai số phức

Bài tập chương 3

Bài 3. Xây dựng lớp vectơ gồm các thành phần:

- Dữ liệu: số phần tử, mảng các phần tử
- Phương thức: nhập, in, tổng 2 vectơ, tích vô hướng 2 vectơ

Hàm main:

- Nhập 2 vectơ a, b
- Tính và in a+b, a*b

Bài 4. Xây dựng lớp ma trận vuông gồm các thành phần:

- Dữ liệu: số hàng ma trận vuông, mảng các phần tử ma trận.
- Phương thức: nhập, in, kiểm tra ma trận có là đơn vị không.

Hàm main:

- Nhập ma trận vuông
- Thông báo có là ma trận đơn vị không
- In ma trận vuông

Bài tập chương 3

Bài 5. Xây dựng lớp sinh viên gồm các thành phần:

- Dữ liệu: họ tên, ngày sinh, giới tính, lớp ,điểm toán, lý, hóa, đtb
- Phương thức: nhập, in, tính điểm trung bình

Hàm main:

- Nhập danh sách sinh viên
- Sắp xếp theo điểm trung bình giảm dần
- In danh sách sau khi xếp

Bài 6. Xây dựng lớp hóa đơn gồm các thành phần:

- Dữ liệu: mã vật tư, tên vật tư, loại phiếu (nhập/xuất), ngày lập, khối lượng, đơn giá, thành tiền
- Phương thức: nhập, in hóa đơn.

Hàm main:

- Nhập danh sách hóa đơn
- Tính thành tiền cho các hóa đơn và in tổng thành tiền
- In danh sách hóa đơn sau khi sắp xếp theo số tiền giảm dần.