Laborator IX

DERIVAREA CLASELOR. POLIMORFISM.

Operațiunea de derivare este unul din cele mai importante mecanisme proprii OOP și asigură, pe lângă reutilizare, premizele apariției polimorfismului.

În operația de derivare membrii clasei de bază sunt moșteniți în clasa derivată. Derivarea se poate face public sau private, definind astfel protecția în clasa derivată.

Derivare public

Membrii *private* din clasa de bază nu sunt accesibili direct prin intermediul obiectelor de tipul clasă derivată. Ei rămân accesibili prin intermediul funcțiilor membru declarate *public* în clasa de bază. Membrii *protected* din clasa de bază sunt accesibili direct prin intermediul obiectelor de tipul clasă derivată, rămânând *protected*. Membrii *public* din clasa de bază sunt accesibili direct prin intermediul obiectelor de tipul clasă derivată, devenind *public*.

Derivare private

Membrii *private* din clasa de bază nu sunt accesibili direct prin intermediul obiectelor de tipul clasă derivată. Ei rămân accesibili prin intermediul funcțiilor membru declarate *public* în clasa de bază. Membrii *protected* din clasa de bază sunt accesibili direct prin intermediul obiectelor de tipul clasă derivată, rămânând *private*. Membrii *public* din clasa de bază sunt accesibili direct prin intermediul obiectelor de tipul clasă derivată, devenind *private*

Membri virtuali și polimorfism

Implementarea polimorfismului în OOP se face cu ajutorul funcțiilor virtuale. Astfel, o funcție virtuală a clasei de bază este înlocuită de aceeași funcție definită în clasa derivată.

O funcție poate fi pur virtuală, în acest caz clasa a cărei membru este devine abstractă. Nu pot exista obiecte de tip clasă abstractă. Clasele abstracte sunt clase de bază pentru derivare.

Unui pointer la obiect clasa de bază i se poate atribui valoarea adresei unui obiect dintr-o clasă derivată la orice nivel din acea clasă de bază. Folosind acest pointer pentru accesul funcțiilor membru virtuale se apelează funcția din clasa derivată și nu cea din clasa de bază. Dacă funcția nu este virtuală, chiar dacă este supradefinită în clasa de derivată, va fi apelată varianta din clasa de bază.

.

1. ***Problema 1***

Codul sursă prezentat mai jos este un exemplu (didactic) de derivare a unor clase pe baza unei ierarhii administrative dintr-o firmă virtuală. Plecând de la acest exemplu veţi dezvolta o aplicaţie mai complexă care va pune în valoare aspecte legate şi de reutilizarea codului folosind compunerea.

```
/*************************************************
angajat.cpp
*************************************************/

#include <iostream>
#include <conio.h>
using namespace std;

class Angajat                  // clasa Angajat
   {
   private:
      enum { LUNG=30 };          // lungimea maxima a numelui
      char nume[LUNG];           // numele angajatului
      unsigned long numar;     //  ID-ul angajatului
   public:
      void intro_date()
         {
         cout << "\n   Introduceti numele: "; cin >> nume;
         cout << "   Introduceti ID-ul angajatului : ";      cin >> numar;
         }
      void afis_date()
         {
         cout << "\n   Nume = " << nume;
         cout << "\n   Numar  = " << numar;
         }
   };
class Manager : public Angajat     // clasa Manager
   {
   private:
      enum { LUNG=40 };          // lungimea maxima a titlului
      char titlu[LUNG];          // "vice-presedinte" etc.
      int nr_masini;              // numarul de masini
   public:
      void intro_date()
         {
         Angajat::intro_date();
         cout << "   Introduceti titlul: ";         cin >> titlu;
         cout << "   Introduceti numarul de masini: "; cin >> nr_masini  ;
         }
      void afis_date()
         {
         Angajat::afis_date();
         cout << "\n   Titlu = " << titlu;
         cout << "\n   Numarul de masini  = " << nr_masini ;
         }
   };
class Cercetator : public Angajat      // clasa Cercetator
   {
   private:
      int pub;      // numbar de publicatii
   public:
      void intro_date()
         {
         Angajat::intro_date();
         cout << "   Introduceti numarul de publicatii: "; cin >> pub;
         }
      void afis_date()
```

```
        {
        Angajat::afis_date();
        cout << "\n   Numarul de publicatii = " << pub;
        }
   };
class Muncitor : public Angajat        // clasa Muncitor
   {
   };

int main()
   {
   Manager m1, m2;
   Cercetator c1;
   Muncitor l1;
   cout << endl;
   cout << "\nIntroduceti date pt. manager 1";     // date pentru
   m1.intro_date();                               // diversi angajati
   cout << "\nIntroduceti date pt. manager 2";
   m2.intro_date();
   cout << "\n Introduceti date pt. cercetator 1";
   c1.intro_date();
   cout << "\nIntroduceti date pt.muncitor 1";
   l1.intro_date();
   cout << "\nDate despre manager 1";           // afisare date
   m1.afis_date();                             //  angajati
   cout << "\nDate despre manager 2";
   m2.afis_date();
   cout << "\nDate despre cercetator 1";
   c1.afis_date();
   cout << "\nDate despre muncitor 1";
   l1.afis_date();
   cout<<"\nApasati o tasta...";
   _getch();
   return 0;
   }
```

1.Rulaţi exemplul şi explicaţi sintaxa apelurilor marcate.

2. Explicaţi de ce nu există cod pentru constructor, constructor de copiere şi destructor.

3. Verificaţi modul de acces şa datele private ale unei clase derivate public (de exemplu, puteţi accesa direct din clasa *Manager* data membru *nume*, moştenită de la clasa *Angajat* ?)

4. Modificaţi adecvat codul astfel încât:

- în locul tabloului de caractere pentru nume să apară un pointer către char şi zona de memmorie să se aloce dinamic (evident va trebui să scrieţi cod pentru constructor, constructor de copiere şi destructor);

- după ce în prealabil aţi definit o clasă *Data* (cu componentele private *zi, luna, an* de tip întreg şi funcţiile membru publice *void introData(), void afisData(), int get_an()* , introduceţi ca dată membru în clasa Angajat, data naşterii.

5. Proiectaţi o funcţie care, plecând de la data curentă să afişeze vârsta în ani a unui membru oarecare al firmei (de ce tip va fi paramatrul de apel?). Afişaţi din funcţie datele angajatului (din raţiuni didactice!). Explicaţi comportatea programului şi găsiţi soluţii astfel încât apelul funcţiei să funcţioneze corect (în ceea ce priveşte afişarea datelor componente) pentru orice tip de angajat.

2. ***Problema 2***

Codul sursă de mai jos ilustrează printr-un exemplu simplu mecaismele polimorfismului. Studiați-l cu atenție.

```
/*************************************************
figuri.hpp
*************************************************/

#ifndef _FIGURI_H
#define _FIGURI_H

// declaratie clasa figura

class Figura
   {
   private:
      int xCo, yCo;             // coordonate figura
      int dim;                  // dimensiune  figura
   protected:
      int getx() const { return xCo; }
      int gety() const { return yCo; }
      int getz() const { return dim; }
      void jos() const;         // prototip
   public:                      // constructor 3-arg
      Figura(int x, int y, int s) : xCo(x), yCo(y), dim(s)
         {  }
      virtual void deseneaza() const = 0;  // functie pur virtuala
   };

// declaratie clasa patrat, derivata din figura

class Patrat : public Figura     // patrat
   {
   public:                      //  constructor 3-arg
      Patrat(int x, int y, int s) : Figura(x, y, s)
         {  }
      void deseneaza() const;        // prototip
   };

// declaratie clasa triunghi, derivata din figura

class Triunghi : public Figura                  // triunghi
   {
   public:
      Triunghi(int x, int y, int s) : Figura(x, y, s)
         {  }
      void deseneaza() const;
   };

// declaratie clasa bol, derivata din figura

class Bol : public Figura       // triunghi cu varful in jos
   {
   public:
      Bol(int x, int y, int s) : Figura(x, y, s)
         {  }
      void deseneaza() const;
   };


#endif
```

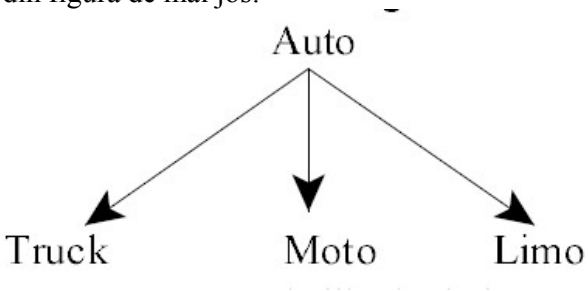
```
/*************************************************
figuri.cpp
*************************************************/

#include <conio.h>
#include <iostream>
#include "figuri.hpp"

using namespace std;

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

void Figura::jos() const        // muta cursorul in jos pana la inceputul figurii
   {
   for(int y=0; y<yCo; y++)
      cout << endl;
   }
////////////////////////////////////////////////////////////////

void Patrat::deseneaza() const       // deseneaza un patrat
   {
   Figura::jos();                // pozitioneaza y la inceputul figurii
   for(int y=0; y<getz(); y++)  // deplaseaza y in jos pt. a desena figura
      {
      int x;
      for(x=1; x<getx(); x++)   // spatii pana la inceputul figurii
         cout << ' ';
      for(x=0; x<getz(); x++)   // deseneaza o linie de X-uri
         cout << 'X';
      cout << endl;
      }
   }
////////////////////////////////////////////////////////////////

void Triunghi::deseneaza() const                    // deseneaza triunghi
   {
   Figura::jos();
   for(int y=0; y<getz(); y++)
      {
      int x;
      for(x=0; x < getx()-y+1; x++)
         cout << ' ';
      for(x=0; x<2*y+1; x++)
         cout << 'X';
      cout << endl;
      }
   }
////////////////////////////////////////////////////////////////

void Bol::deseneaza() const         // deseneaza triunghi cu varful in jos
   {
   Figura::jos();
   for(int y=0; y<getz(); y++)
      {
      int x;
      for(x=0; x < getx()-(getz()-y)+2; x++)
         cout << ' ';
      for(x=0; x < 2*(getz()-y)-1; x++)
         cout << 'X';
      cout << endl;
      }
   }
////////////////////////////////////////////////////////////////
```

```
/***********************************************
main.cpp
***********************************************/

#include <iostream>
#include <conio.h>
#include "figuri.hpp"

using namespace std;

int main()
   {
   const int N = 3;    // nr. de figuri

   Figura* tabpfig[N] = { &Bol(10, 0, 3), &Patrat(20, 1, 5),&Triunghi(30, 1, 7) };
   cout << endl << endl;
   for(int j=0; j<N; j++)   // afiseaza cele 3 figuri
      tabpfig[j]->deseneaza();
   cout<<"\nApasati o tasta...";
    _getch();
       return 0;
   }
```

1.Se pot instanția obiecte de tip Figura? (Justificați)
2. Funcționează programul dacă tabpfig este tablou de figuri?
3. Completați programul și cu alte figuri geometrice.

3. ***Problema 3***

Următorul program urmărește punerea în evidență a conceptului de derivare și a polimorfismului. Structura de derivare este cea din figura de mai jos:




```
/***********************************************
auto.hpp
***********************************************/
/*********************************************
Clasa Auto: clasa de baza pur virtuala pentru derivarea claselor
care modeleaza tipuri de autovehicule.
Nu pot exista obiecte de tipul Auto !!!
*********************************************/

#include <iostream>
#include <conio.h>

#ifndef _AUTO_INCLUDED_
#define _AUTO_INCLUDED_
using namespace std;
enum Fuel //Tip de combustibil ...
{
    Motorina,
```

```
    CO90,
    CO98
};

enum State //Rezultatul unei actiuni ...
{
    Error,
    Success
};

enum Motor
{
    Oprit,
    Pornit,
};

class Auto //Clasa de baza ...
{
protected:
        int               Stare; //Starea in care se afla auto ...
        int               VitezaMaxima; //Viteza maxima in Km/h ...
        int               Viteza; //Viteza instantanee ...
        int               Capacitate; //Capacitatea cilindrica ...
        Fuel             TipCombustibil;
        int               CantitateCombustibil; //Cantitatea de combustibil in litri
        void Print(ostream &);

public:
        //Constructor implicit.
        Auto(void) : Stare(Oprit), VitezaMaxima(0), Viteza(0), Capacitate(0),
                          TipCombustibil(CO98), CantitateCombustibil(0) {};
        //Constructor de copiere.
        Auto(Auto const &A) : Stare(A.Stare), VitezaMaxima(A.VitezaMaxima), Viteza(A.Viteza),
                                          Capacitate(A.Capacitate),
TipCombustibil(A.TipCombustibil),
                                          CantitateCombustibil(A.CantitateCombustibil) {};
        virtual ~Auto() {};

        //Functie virtuala pentru initializare. Este utilizata in clasele derivate.
        virtual void SetUp(void);
        //Functie virtuala. Returneaza Error sau Success.
        virtual int Pornire(void);
        //Functie pur virtuala. Returneaza Error sau Success.
        //Parametrul reprezinta timpul in secunde cat dureaza accelerarea.
        virtual int Accelerare(int) = 0;
        //Functie pur virtuala. Returneaza Error sau Success.
        //Parametrul reprezinta timpul in ore cat dureaza croaziera.
        virtual int Croaziera(int) = 0;
};

#endif

/***********************************************
truck.hpp
***********************************************/
/*********************************************
Clasa Truck: Clasa derivata din clasa Auto.
*********************************************/

#include "auto.hpp"

#ifndef _TRUCK_INCLUDED_
#define _TRUCK_INCLUDED_
using namespace std;
class Truck : public Auto
```

```
{
protected:
        int             Remorca; //Numarul de remorci ...
public:
        Truck(void);
        Truck(Truck const &T) : Auto(T), Remorca(T.Remorca) {};
        ~Truck() {};

        virtual void SetUp(void);
        virtual int Pornire(void);
        virtual int Accelerare(int);
        virtual int Croaziera(int);
        friend ostream& operator <<(ostream &, Truck &);
};

#endif

/************************************************
moto.hpp
************************************************/
/*********************************************
Clasa Moto: Clasa derivata din clasa Auto.
*********************************************/

#include "auto.hpp"

#ifndef _MOTO_INCLUDED_
#define _MOTO_INCLUDED_
using namespace std;

class Moto : public Auto
{
public:
        Moto(void);
        Moto(Moto const &M) : Auto(M) {};
        ~Moto() {};

        virtual void SetUp(void);
        virtual int Pornire(void);
        virtual int Accelerare(int);
        virtual int Croaziera(int);
        friend ostream & operator <<(ostream &, Moto &);
};

#endif

/************************************************
limo.hpp
************************************************/
/*********************************************
Clasa Limo: Clasa derivata din clasa Auto.
*********************************************/

#include "auto.hpp"

#ifndef _LIMO_INCLUDED_
#define _LIMO_INCLUDED_
using namespace std;
class Limo : public Auto
{
protected:
        int             Locuri; //Numarul de locuri ...
public:
        Limo(void);
        Limo(Limo const &L) : Auto(L), Locuri(L.Locuri) {};
```

```
        ~Limo() {};

        virtual void SetUp(void);
        virtual int Pornire(void);
        virtual int Accelerare(int);
        virtual int Croaziera(int);
        friend ostream & operator <<(ostream &, Limo &);
};

#endif

/*************************************************
auto.cpp
*************************************************/
/*************************************************
Clasa Auto: clasa de baza pur virtuala pentru derivarea claselor care modeleaza tipuri de
autovehicule. Nu pot exista obiecte de tipul Auto !!!
*************************************************/
#include <iostream>
#include "auto.hpp"
using namespace std;
//Functie virtuala pentru initializare. Este utilizata in clasele derivate.
 void Auto::SetUp(void)
{
    cout << "Viteza initiala de croaziera: ";
    cin >> Viteza;
    cout << "Numarul de litri de combustibil la alimentare: ";
    cin >> CantitateCombustibil;
}

 int Auto::Pornire(void)
{
    if(Stare == Pornit)
    {
        cout << "Deja pornit ..." << endl;
        return Success;
    }
    if(CantitateCombustibil <= 0)
    {
        cout << "Rezervor gol !!!" << endl;
        return Error;
    }
    Stare = Pornit;
    return Success;
}

void Auto::Print(ostream &Out)
{
    switch(Stare)
    {
        case Pornit:
                Out << "Pornit" << endl;
                break;
        case Oprit:
                Out << "Oprit" << endl;
                break;
    }
    Out << "Viteza maxima: " << VitezaMaxima << "km/h" << endl;
    Out << "Viteza de croaziera: " << Viteza << "km/h" << endl;
    Out << "Capacitate cilindrica: " << Capacitate << "cmc" << endl;
    Out << "Tip combustibil: ";
    switch(TipCombustibil)
    {
        case Motorina:
                Out << "Motorina" << endl;
```

```
                break;
        case CO90:
                Out << "Benzina cu CO 90" << endl;
                break;
        case CO98:
                Out << "Benzina cu CO 98" << endl;
                break;
    }
    cout << "Cantitatea de combustibil din rezervor: " << CantitateCombustibil << " l." << endl;
}


/************************************************
truck.cpp
************************************************/
/***********************************************
Clasa Truck: Clasa derivata din clasa Auto.
***********************************************/
#include <iostream>
#include "truck.hpp"
using namespace std;
Truck::Truck()
{
    SetUp();
}

 void Truck::SetUp(void)
{
    cout << "Truck :" << endl;
    VitezaMaxima = 120;
    Capacitate = 4000;
    TipCombustibil = Motorina;
    Auto::SetUp(); //Apel la functia din clasa de baza ...
    cout << "Numar remorci: ";
    cin >> Remorca;
    if(Remorca < 0)
        Remorca = 0;
}

 int Truck::Pornire(void)
{
    cout << "Truck :" << endl << "Incerc sa pornesc ..." << endl << "BRRrrrr" << endl;
    if(Auto::Pornire())
    {
        cout << "Brm Brm Brm" << endl;
        return Success;
    }
    cout << "Chhh Chhh" << endl;
    return Error;
}

 int Truck::Accelerare(int Timp)
{
    cout << "Truck :";
    if(Stare == Pornit && CantitateCombustibil > 0)
    {
        cout << "Accelerez .........." << endl;
        if((Viteza += 5 * Timp) > VitezaMaxima)
                Viteza = VitezaMaxima;
        return Success;
    }
    cout << "SSSSssss ...." << endl;
    Stare = Oprit;
    return Error;
}
```

```
 int Truck::Croaziera(int Timp)
{
    cout << "Truck :" << endl;
    if(Stare == Oprit)
    {
        cout << "SSSSssss ...." << endl;
        return Error;
    }
    for(int Contor = 0; Contor < Timp; Contor++)
    {
        if(CantitateCombustibil > 0)
                cout << "Brm Brm Brm" << endl;
        else
        {
            cout << "Rezervor gol !!!" << endl;
            CantitateCombustibil = 0;
            return Error;
        }
        CantitateCombustibil -= 10 + 2 * Remorca;
    }
    return Success;
}

ostream & operator <<(ostream &Out, Truck &T)
{
    Out << "Truck :" << endl;
    T.Print(Out);
    cout << "Numar de remorci: " << T.Remorca << endl;
    return Out;
}

/************************************************
moto.cpp
************************************************/
/*********************************************
Clasa Moto: Clasa derivata din clasa Auto.
*********************************************/
#include <iostream>
#include "moto.hpp"
using namespace std;
Moto::Moto()
{
    SetUp();
}

 void Moto::SetUp(void)
{
    cout << "Moto" << endl;
    VitezaMaxima = 160;
    Capacitate = 1400;
    TipCombustibil = CO90;
    Auto::SetUp(); //Apel la functia din clasa de baza ...
}

 int Moto::Pornire(void)
{
    cout << "Moto :" << endl << "Incerc sa pornesc" << endl << "BRRrrrr BRRrrrr" << endl;
    if(Auto::Pornire())
    {
        cout << "Brm Tic Tic Tic Brm Tic Tic Tic Brm Tic Tic Tic" << endl;
        return Success;
    }
    cout << "Chhh Chhh" << endl;
    return Error;
}
```

```
 int Moto::Accelerare(int Timp)
{
    cout << "Moto:";
    if(Stare == Pornit && CantitateCombustibil > 0)
    {
        cout << "Accelerez ........." << endl;
        if((Viteza += 20 * Timp) > VitezaMaxima)
                Viteza = VitezaMaxima;
        return Success;
    }
    cout << "Moto : SSSSssss ...." << endl;
    Stare = Oprit;
    return Error;
}

 int Moto::Croaziera(int Timp)
{
    cout << "Moto :" << endl;
    if(Stare == Oprit)
    {
        cout << "SSSSssss ...." << endl;
        return Error;
    }
    for(int Contor = 0; Contor < Timp; Contor++)
    {
        if(CantitateCombustibil > 0)
                cout << "RRRRRrrrrrrrrrrrrrrr" << endl;
        else
        {
            cout << "Rezervor gol !!!" << endl;
            CantitateCombustibil = 0;
            return Error;
        }
        CantitateCombustibil -= 3;
    }
    return Success;
}

ostream & operator <<(ostream &Out, Moto &M)
{
    Out << "Moto :" << endl;
    M.Print(Out);
    return Out;
}

/************************************************
limo.cpp
************************************************/
/***********************************************
Clasa Limo: Clasa derivata din clasa Auto.
***********************************************/
#include <iostream>
#include "limo.hpp"
using namespace std;
Limo::Limo()
{
    SetUp();
}

 void Limo::SetUp(void)
{
    cout << "Limo" << endl;
    VitezaMaxima = 220;
    Capacitate = 1600;
    TipCombustibil = CO98;
```

```
    Locuri = 5;
    Auto::SetUp(); //Apel la functia din clasa de baza ...
}

 int Limo::Pornire(void)
{
    cout << "Limo :" << endl << "Incerc sa pornesc" << endl << "Bzzzz" << endl;
    if(Auto::Pornire())
    {
        cout << "TicTicTicTicTic" << endl;
        return Success;
    }
    cout << "Chhh Chhh" << endl;
    return Error;
}

 int Limo::Accelerare(int Timp)
{
    cout << "Limo:";
    if(Stare == Pornit && CantitateCombustibil > 0)
    {
        cout << "Accelerez ........." << endl;
        if((Viteza += 40 * Timp) > VitezaMaxima)
                Viteza = VitezaMaxima;
        return Success;
    }
    cout << "Limo : SSSSssss ...." << endl;
    Stare = Oprit;
    return Error;
}

 int Limo::Croaziera(int Timp)
{
    cout << "Limo :" << endl;
    if(Stare == Oprit)
    {
        cout << "SSSSssss ...." << endl;
        return Error;
    }
    for(int Contor = 0; Contor < Timp; Contor++)
    {
        if(CantitateCombustibil > 0)
                cout << "ZZZZZZZzzzzzzzzzzzzzzz" << endl;
        else
        {
            cout << "Rezervor gol !!!" << endl;
            CantitateCombustibil = 0;
            return Error;
        }
        CantitateCombustibil -= 7;
    }
    return Success;
}

ostream & operator <<(ostream &Out, Limo &L)
{
    Out << "Limo :" << endl;
    L.Print(Out);
    cout << "Numar de locuri: " << L.Locuri << endl;
    return Out;  }

/*********************************************
main.cpp
*********************************************/
#include "truck.hpp"
```

```
#include "moto.hpp"
#include "limo.hpp"
#include <iostream>
using namespace std;
int main(void)
{
    Auto *Automobile[3]; //Vector de pointeri la obiecte de tipul Auto ...
    Truck T;
    Moto M;
    Limo L;
    Automobile[0] = &T;
    Automobile[1] = &M;
    Automobile[2] = &L;
    int Contor;
    int Timp;
    for(Contor = 0; Contor < 3; Contor++)
    {
        if(!Automobile[Contor]->Pornire())
            cout << "Probleme cu automobilul " << Contor << endl;
        getch();
    }
    cout << "Timpul de accelerare ? (secunde) ";
    cin >> Timp;
    for(Contor = 0; Contor < 3; Contor++)
    {
        if(!Automobile[Contor]->Accelerare(Timp))
            cout << "Probleme cu automobilul " << Contor << endl;
        getch();
    }
    cout << T;
    getch();
    cout << M;
    getch();
    cout << L;
    getch();
    cout << "Timpul de croaziera ? (ore) ";
    cin >> Timp;
    for(Contor = 0; Contor < 3; Contor++)
    {
        if(!Automobile[Contor]->Croaziera(Timp))
            cout << "Probleme cu automobilul " << Contor << endl;
        getch();
    }
    cout << T;
    getch();
    cout << M;
    getch();
    cout << L;
    getch();

    /*
    //Utilizare gresita a pointerului la obiect de tip clasa de baza caruia i s-a atribuit adresa
    //unui obiect din clasa derivata.
    for(Contor = 0; Contor < 3; Contor++)
    {
        cout << *(Automobile[Contor]);
        getch();
    }
    */
       cout<<"\nApasati o tasta...";
    return 0;
}
```