Nesne İşaretçileri

Binnur Kurt
kurt@ce.itu.edu.tr

Bilgisayar Mühendisliği Bölümü
İstanbul Teknik Üniversitesi

Sınıf Yapısı

4 Kalıtım

Çok Şekillilik

Nesne İşaretçileri

İşaretçiler, veri değil, verinin yerleşik bulunduğu bellek gözünün adresini taşırlar. İşaretçiler basit tipte değişkenlere işaret edebildikleri gibi, bir nesneye de işaret edebilirler. İşaretçiler kullanılmadan önce uygun başlangıç değeri atanmalıdır:

- new operatörü

İşletim sisteminden uygun miktarda bellek alanı alır. Döndürdüğü değer bu alanın başlangıç adresidir. Eğer işlem başarısız olursa 0 (NULL) döndürür.

Nesne işaretçilerinde new kullanıldığında yukarıdakine ek olarak nesnenin kurucu fonksiyonu çalıştırılır. Böylece nesne yaratılırken başlangıç değerleri atanmış olur.

Nesne İşaretçileri

- delete operatörü

Belleğin verimli ve etkin kullanımı için, new operatörünün kullanımına karşılık olarak, bellek alanı, kullanımı bittiğinde işletim sistemine delete operatörü ile geri verilmelidir.

new ile aşağıdaki biçimde bir nesne dizisi için bellek alındığında

```
int * ptr = new int[10];
```

delete ile

```
delete [ ] ptr;
```

şeklinde işletim siteminde geri verilmelidir. İşaretçi önündeki "[]" kullanılmaz ise sadece dizinin ilk elemanı için bellek alanı geri verilir.

```
class person {
    char *name;
  public:
    person();
    void setName(char *);
    void printName()
    {
       cout << "Name is:" << name<<endl;
    }
    ~person(){
        cout << "Destructor" << endl;
        delete[] name;}
};

person::person()
{
   cout << "Constructor" << endl;
   name = new char;
   name = '\0';
}
```

```
void person::setName(char *n)
{
  delete[] name;
  name = new char[strlen(n)+1];
  strcpy(name, n);
}
```

```
void main()
{
   person* persPtr = new person[3];
   persPtr->setName("Person1");
   (persPtr+1)->setName("Person2");
   (persPtr+2)->setName("Person3");
   persPtr->printName();
   (persPtr+1)->printName();
   (persPtr+2)->printName();
   delete [ ] persPtr;
}  // end main()
```



```
void main()
{
   person* persPtr = new person[3];
   persPtr->setName("Ahmet");  persPtr[0].setName("Ahmet") ;
   (persPtr+1)->setName("Ayse");
   (persPtr+2)->setName("Ali");  *(persPtr+2).setName("Ali") ;
   persPtr->printName();
   (persPtr+1)->printName();
   (persPtr+2)->printName();
   delete [ ] persPtr;
}  // end main()
```

Nesne Bağlantılı Listeleri

Bir sınıf kendi tipinden bir nesneye işaretçi içerebilir. Bu işaretçi kullanılarak bir nesne zinciri (bağlantılı liste) kurulabilir.

```
class Teacher
{
  friend class TeacherList ;
  char *name;
  int age,numberOfStudents;
  Teacher * next;
 public:
  Teacher(char *, int, int);
  void print();
  char *getName(){return name;}
  ~Teacher() {
    cout <<" Destructor of teacher" ;
    delete []name;
   }
};
```

```
/* linked list for teachers */
class TeacherList
{
   Teacher *head;
 public:
   TeacherList(){ head=0L; }
   char append(char*,int,int);
   char remove(char*);
   void print();
};
```



```
class Teacher
{
 private:
  friend class TeacherList ;
  char *name;
  int age,numberOfStudents;
  Teacher * next;
 public:
  Teacher(char *, int, int);
  void print();
  inline char *getName(){return name;}
  ~Teacher()
  {
       cout << "\nDestructor of " << name ;
       delete []name;
   }
};
```

```
Teacher::Teacher(char *n,int a,int nos) : age(a),
numberOfStudents(nos)
{
  name = new char[strlen(n)+1] ;
  strcpy(name,n) ;
  next = 0L ;
}
```



```
class TeacherList
{
 private:
   Teacher *head;
 public:
   TeacherList(){ head=0L; }
   ~TeacherList() ;
   char append(char*,int,int);
   char append(Teacher*);
   char remove(char*);
   void print();
} ;
```

```
char TeacherList::append(Teacher *t)
{
    if (head==0L)
    {
      head = t ;
    }
    else
    {
      Teacher *p = head;
      while (p->next!=0L)
      {
        p = p->next ;
      }
      p->next = t ;
    }
    return 1 ;
}
```



```
char TeacherList::append(char *n,int a,int nos)
{
     append(new Teacher(n,a,nos)) ;
}
```

```
char TeacherList::remove(char *n) {
    if (head==0L)
      return 0;
   else {
      Teacher *p=head,*q=head;
      while (p!=0L) {
        if (strcmp(p->name,n)==0) {
          if (head==p) head=p->next ;
          else         q->next = p->next ;
          delete p ;
          return 1;
        }
        q = p ;
        p = p->next ;
      }
    }
    return 0 ;
}
```

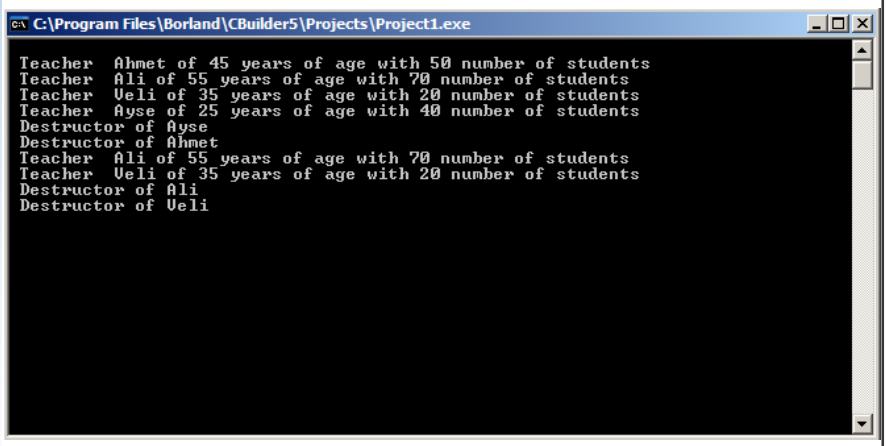


```
void TeacherList::print()
{
   Teacher *p=head;
   while (p!=0L)
   {
     cout << "\n Teacher  " << p->name ;
     cout << " of " << p->age << " years of age" ;
     cout << " with " << p->numberOfStudents <<
                 " number of students";
     p = p->next ;
   }
}
```

```
TeacherList::~TeacherList()
{
    if (head==0L) return ;
    Teacher *p=head,*q;
    while (p!=0L)
    {
      q = p->next ;
      delete p ;
      p = q ;
    }
}
```



```
int main(int argc, char* argv[]) {
  {
   TeacherList tl ;
   tl.append("Ahmet",45,50) ;
   tl.append("Ali",55,70) ;
   tl.append("Veli",35,20) ;
   tl.append("Ayse",25,40) ;
   tl.print() ;
   tl.remove("Ayse") ;
   tl.remove("Ahmet") ;
   tl.print() ;
  }
   return 0;
}
```



İşaretçiler ve Kalıtım

Eğer bir sınıf temel bir sınıftan türetilmiş ise, türetilmiş sınıftan bir işaretçiye herhangi bir tip dönüşümü gerekmeksizin temel sınıftan bir işaretçi atanabilir.Temel sınıfın işaretçisi türetilmiş sınıfa işaretçi olabilir. Tersi bir dönüşüm, tip dönüşümü gerektirir.

Örneğin, teacher nesnesine bir işaretçi teacher ve principal nesnelerine işaret edebilir. principal ile teacher arasında “is a” ilişkisi vardır : principal is a teacher. Ancak tersi her zaman doğru değildir : her teacher bir principal olmayabilir.

```
class Base
{
};
class Derived : public Base
{
};
Derived d;
Base *bp = &d;          // implicit conversion
Derived *dp = bp;       // error Base is not Derived
dp = (Derived *) bp;    // explicit conversion
```

Eğer bir sınıf kalıtım ile private olarak temel sınıftan türetilirse, bu durumda tip dönüşümü yapılamaz. Çünkü temel sınıfın public üyeleri temel sınıfa ait işaretçiler tarafından erişilebilir. Ancak türetilmiş sınıftan nesneler yada işaretçiler temel sınıf üyelerine erişemezler.

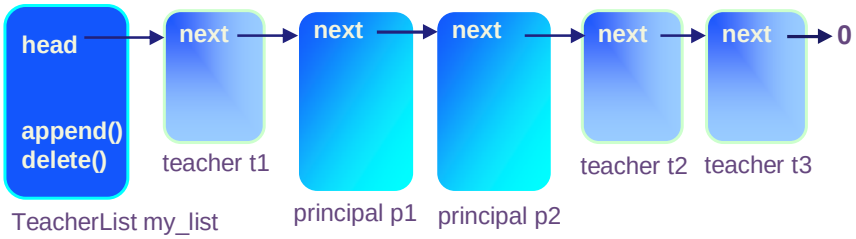
```
class Base {
    int m1;
  public:
    int m2;
};
class Derived : private Base { // m2 is not a public member of Derived
};

Derived d;
d.m2=5;              // error   m2 is private member of Derived
Base *bp=&d;         // error private base
bp->m2=5;            // ok
bp = (Base*)&d;      // ok: explicit conversion
bp->m2=5;            // ok
```


Heterojen Listeler

İşaretçiler ve kalıtım kullanılarak, heterojen bağlantılı listeler oluşturulabilir. Temel sınıfa işaret eden nesnelerden oluşan liste, bu temel sınıftan türetilmiş tüm sınıflara ait nesneler içerebilir. Heterojen listeleri daha sonra çok şekillilik konusunda tekrar inceleyeceğiz.

Örnek: öğretmenler ve müdürler listesi



```
class Principal : public Teacher
{
  private:
    friend class TeacherList ;
     int numberOfTeachers;
  public:
    Principal(char *, int, int,int);
    ...
};
```

TeacherList sınıfında herhangi bir değişiklik yapmamıza gerek yok !