Programmazione II

Analisi del Problema

Il testo richiede la progettazione e l'implementazione di una struttura dati (ADT) che consenta di:

- Creare N oggetti Cittadino con i vari tipi (Studente, Lavoratore, Pensionato).
- Inserire questi oggetti dentro una struttura dati
- Visualizzarli in ordine crescente di età
- Ricercare i cittadini che hanno età minore o uguale a quella inserita, stamparli e visualizzare il numero di studenti, lavoratori e pensionati.

Inoltre è richiesto che:

- La creazione di uno studente, lavoratore o pensionato avvenga in modo casuale.
- La creazione degli attributi per nome, cognome, età, città, stipendio, ecc.. avvenga anche in modo casuale.
- La struttura dati venga visualizzata in modo crescente.

Considerazione Progettuali

Tenendo conto che il tipo Cittadino dovrà essere scelto casualmente. Si corre all'uso di uno switch case e una funzione rand. Questo permetterà la scelta casuale del tipo di Cittadino.

```
int choice=rand()%3;

//PUNTO A
switch(choice){
    case 0:{
    Studente s(nomi[(rand()%30)],cognomi[(rand()%30)],((rand()%19)+6),cities[(rand()%30)],((rand()%100+1) * 100),"Studente");
    list.insert(s);
    cout<<"Cittadino(Studente) caricato nella lista #"<<ss+++<<" :"<<s<<endl;
    i++;
    break;
    }
    case 1:{
    Lavoratore l(nomi[(rand()%30)],cognomi[(rand()%30)],((rand()%52)+18),cities[(rand()%30)],((rand()%100+1) * 100),"Lavoratore");
    list.insert(l);
    cout<<"Cittadino(Lavoratore) caricato nella lista #"<<ll+++<<" :"<<l<<endl;
    i++;
    break;}

    case 2:{
    Pensionato p(nomi[(rand()%30)],cognomi[(rand()%30)],((rand()%25)+65),cities[(rand()%30)],((rand()%100+1) * 100),"Pensionato");
    list.insert(p);
    cout<<"Cittadino(Pensionato) caricato nella lista #"<<pp+++<<" :"<<p<<endl;
    i++;
    break;  }
}
```

Tramite la creazione di tre array statici denominati "nomi", "cognomi" e "cities", è stato possibile assegnare casualmente i valori degli attributi di ciascun tipo di Cittadino.

```
string nomi[30]={"Nino","Lapo","Divo","Lupo","Brando","Giusto","Sante","Elio","Attilio","Erenia","Fosco","Ferruccio","Fermo","Faber","Severo",
string cognomi[30]={"Ferraro", "Carpenteri", "Muratori", "Moltisanti", "Faro", "Gallo", "Tinè","Stanco", "Bella","Battiato","Morello","Sorge",
string cities[30]={"Augusta","Catania","Bolzano","Milano","Enna","Bari","Palermo","Belpasso","Fano","Agira","Piacenza","Portofino","Riccione",
```

La caratteristica che contraddistingue gli uni dagli altri è l'età, infatti:

• Gli studenti hanno un'età da 6 a 25;

• I lavoratori hanno un'età da 18 a 70;
• I pensionati hanno un'età da 65 a 90.

La struttura che utilizzata è stata una Lista Doppiamente Linkata, poiché agevolava la richiesta di visualizzare gli oggetti estraendoli dalla struttura dati in ordine di età, permettendo di percorrere la struttura dati in entrambi i versi. Ne consegue che le operazioni di inserimento e ricerca avvengono in tempo O(n).

L'ordinamento della lista è stato introdotto in fase di inserimento usando l'overload degli operatori di comparazione con l'età come parametro di confronto.
Ciò ha reso possibile ottenere la lista ordinata senza ricorrere ad alcun algoritmo di ordinamento (rispettando le linee guida del progetto).

```
bool operator > (const Cittadino&c)const{
    return this->eta>c.eta;
}

bool operator >= (const Cittadino&c)const{
    return this->eta>=c.eta;
}

bool operator < (const Cittadino&c)const{
    return this->eta<c.eta;
}

bool operator <= (const Cittadino&c)const{
    return this->eta<=c.eta;
}

bool operator == (const Cittadino&c)const{
    return this->eta==c.eta;
}
```

Per poter effetuare la ricerca dei cittadini che hanno età minore o uguale al valore dell'età inserita in fase di compilazione è stata implementata una funzione chiamata "search".

```
void search(int e){
    NodeDL<Cittadino> *cur= this->head;
    int s1=0,l1=0,p1=0;
    while(cur!=NULL && cur->getValue().getEta()<=e){
       cout<<"Cittadino Trovato!"<< " ";
       cout<<*cur<<endl;
       if(cur->getValue().getStatus()=="Lavoratore"){
           l1++;
       }
       else if(cur->getValue().getStatus()=="Studente"){
           s1++;
       }
       else if (cur->getValue().getStatus()=="Pensionato"){
           p1++;
       }
        cur=cur->getNext();
    }
    cout<<"Totale-->  Studenti: "<<s1<<" Lavoratori: "<<l1<<" Pensionati: "<<p1<<endl;

    }
```

CLASSI UTILIZZATE

- Cittadino: classe che modella il Cittadino.

```
Cittadino(string nome, string cognome , int eta, string city, double stipendio,string status):
nome(nome),cognome(cognome),eta(eta),city(city),stipendio(stipendio),status(status) {}
```

1. Implementati i metodi getter e setter.
2. Overload della redirezione dell'output.

```
friend ostream& operator<<(ostream& out, const Cittadino& c){
    out << c.nome << ", " << c.cognome << ", " << c.eta << ", " << c.city << ", " << c.stipendio<< ", "<<c.status;
    return out;
}
```

3. Overload degli operatori di comparazione logica

- Studente: sottoclasse della classe Cittadino che modella lo Studente.

```
Studente(string nome, string cognome, int eta, string city, double stipendio,string status) :
Cittadino(nome, cognome, eta, city,0.0,"Studente"){}
Studente():Cittadino("","",0,"",0.0,"Studente"){}
```

Si è pensato di settare nel costruttore, lo stipendio dello studente a 0, poiché essendo studente non ha un lavoro e non ha introiti.

- Lavoratore: sottoclasse della classe Cittadino che modella il Lavoratore.

```
Lavoratore(string nome, string cognome, int eta, string city, double stipendio,string status) :
Cittadino(nome, cognome, eta, city,stipendio,"Lavoratore"){}
Lavoratore():Cittadino("","",0,"",0.0,"Lavoratore"){}
```

- Pensionato: sottoclasse della classe Cittadino che modella il Pensionato.

```
Pensionato(string nome, string cognome, int eta, string city, double stipendio,string status) :
Cittadino(nome, cognome, eta, city,stipendio,"Pensionato"){}
Pensionato():Cittadino("","",0,"",0.0,"Pensionato"){}
```

- NodeDL: è una classe template che rappresenta un nodo con successore e predecessore.

```
NodeDL(T value, NodeDL<T> *prev , NodeDL<T> *next): value(value),prev(prev),next(next){}
NodeDL(T value) : NodeDL(value, NULL,NULL){}
```

- ListDL: è una classe template che rappresenta una lista doppiamente linkata, con puntatori testa e coda.

```
ListDL() : ListDL(true){}
ListDL(bool ascend):head(NULL), tail(NULL), length(0),ascend(ascend){}
```

- Cittadinolista: è una sottoclasse della classe ListDL.

```
Cittadinolista():ListDL<Cittadino>(true){}
Cittadinolista(bool ascend):ListDL<Cittadino>(ascend){}
```