(Лаборатори №8)

Э.Багабанди

ХШУИС - ийн ПХ хөтөлбөрийн

3-р түвшний оюутан, 19B1NUM0700@num.edu.mn

1. **ОРШИЛ/УДИРТГАЛ**

Уг лабораторийн хүрээнд өгөгдсөн дасгалуудыг гүйцэтгэж, жижиг хэмжээний програм хөгжүүлэхэд ашиглагдсан C++ хэлний онолын ойлголт /удамшилын харьцаа, бүрдэл харьцаа, хориглолт/ - уудыг судалж, эзэмшсэн болно.

2. **ЗОРИЛГО**

C++ хэлний удамшилын харьцаа, бүрдэл харьцаа, тэдгээрийн хэрэгжүүлэлт, бүрдэл харьцааны constraint буюу хориглолтын талаар судалж, үүнийг ашиглан лабораторт өгөгдсөн дасгалуудыг гүйцэтгэн жижиг хэмжээний хэрэгжүүлэлт хийх.

3. **ОНОЛЫН СУДАЛГАА**

3.1. Удамшил ямар харьцаа үүсгэдэг вэ?

- Удамшил гэдэг нь нэгэнт бий болсон зүйлийг дахин хэрэглэхтэй холбоотой ойлголт бөгөөд туршиж бэлэн болгосон классын кодыг өөр системд дахин хэрэглэх буюу эргэн хэрэглэгдэх чадвар, чадамжтай болгодог. Өөрөөр хэлбэл бэлэн байгаа классаас шинэ класс үүсгэх арга технологи гэж хэлж болох бөгөөд энэ үед шинээр үүсч буй класс нь өмнөх классынхаа шинжийг өвлөн авна.
- Өөрөөс нь класс үүсч байгаа классыг үндсэн(дээд/эх) класс, шинээр үүсч байгааг нь удамших, удамшсан(дэд/охин/хүүхэд) класс гэнэ.
- Удамшиж үүсэх класс нь эх классынхаа шинж, аргаас өвлөж авахаас гадна өөрийн гэсэн шинжтэй, өөрөөр хэлбэл эх класст байхгүй гишүүдтэй байж болно. Эх классын гишүүн функцийг удамших класс дотор дахин тодорхойлж болох ба ингэхэд эх классын эх код заавал байх албагүй юм.
- Удамшил нь класс хооронд “**is a**” буюу “**- төрлийнх, бол**” гэсэн харьцаа үүсгэдэг. Өөрөөр хэлбэл, суудлын машин нь машин төрлийнх гэвэл суудлын машин нь машинаас удамшсан гэж үзэх юм.

3.2. Объект хандлагат программчлалд хэрхэн хэрэгжүүлдэг вэ?

- **Классыг <u>public</u> горимоор удамшуулж үүсгэх** үеийн удамших классын тодорхойлолт:

```
class derived_class: public based_class
{
        data_members;
        members_function;
}
```

- Эх класс <u>public</u> горимоор удамших тохиолдолд түүний <u>public</u> гишүүд удамшиж үүсэх классын <u>public</u> гишүүд болно. Иймээс удамших классын функц ба объект эх классынхаа public гишүүд рүү хандана. Харин эх класс <u>public</u> горимоор удамших үед түүний <u>private</u> гишүүд удамших хэдий ч удамших классын гишүүд эх классынхаа

private гишүүд рүү хандаж чадахгүй. Мөн эх класс public горимоор удамших үед түүний protected гишүүд удамших классынхаа protected гишүүд болох тул уг гишүүд рүү удамших классын функцээс хандаж болно.

- **Классыг private горимоор удамшуулж үүсгэх** үеийн удамших классын тодорхойлолт:

```
class derived_class: private based_class
{
        data_members;
        members_function;
}
```

- Эх класс private горимоор удамших тохиолдолд түүний public болон protected гишүүд удамшиж үүсэх классын private гишүүд болно. Иймд удамших классын объект эх классынхаа гишүүд рүү хандаж чадахгүй. Харин функц нь классынхаа private гишүүд рүү хандаж болдог тул эх классынхаа public, protected гишүүд рүү удамших классын функцээс хандана.
- **Классыг protected горимоор удамшуулж үүсгэх** үеийн удамших классын тодорхойлолт:

```
class derived_class: protected based_class
{
        data_members;
        members_function;
}
```

- Эх класс protected горимоор удамших тохиолдолд түүний public болон protected гишүүд удамшиж үүсэх классын protected гишүүд болно. Иймд уг гишүүд рүү удамших классын функцээс хандаж болно. Харин эх класс protected горимоор удамших үед түүний private гишүүд удамших хэдий ч удамших классын гишүүд эх классынхаа private гишүүд рүү хандаж чадахгүй.
- **Удамших горимын хүснэгт:**

Эх классын гишүүд	Удамших горим		
	Public	private	protected
public	Public	private	protected
private	X	X	X
protected	Protected	private	protected

3.3. Бүрдэл харьцаа гэж юу вэ?

- Бүрдэл харьцаа гэдэг нь өөр классын объект тухайн классын гишүүн өгөгдөл болж, ашиглагдахын хэлнэ.
- Бүрдэл харьцаа нь класс хооронд “has a” буюу “-тай[3]” харьцаа үүсгэдэг. Өөрөөр хэлбэл машин нь мотортой гэвэл машин болон мотор нь хоорондоо бүрдэл харьцаатай бөгөөд мотор нь машин гэх классын гишүүн өгөгдөл болох юм.

3.4. Объект хандлагат программчлалд хэрхэн хэрэгжүүлдэг вэ?

- Бүрдэл харьцааг объект хандлагат програмчлалд дараах байдлаар хэрэгжүүлнэ.

```
class Motor {
        member_data
        member_function
}
class Wheel {
        member_data
        member_function
}
class Car {
        Motor motor;
        Wheel wh[4];
        …
        member_function
}
```

3.4. Бүрдэл харьцааны хориглолт (constraint) гэж юу вэ?

- Бүрдэл харьцааны хориглолт буюу constraint гэдэг нь өөр нэг классын объект тухайн классын гишүүн өгөгдөл болж, ашиглахад үед тухайн класст уг гишүүн өгөгдөл объект хэдэн ширхэг хамгийн багадаа, хамгийн ихдээ хэд байхыг заасан хязгаарлалты хэлнэ. Тухайлбал, машин мотор хоёр нь хоорондоо бүрдэл харьцаатай буюу машин нь мотортой байг. Энэ үед тухайн 2 – ийн бүрдэл харьцааны хориглолт 1..n гэж үзвэл машин хамгийн багадаа нэг мотортой, хамгийн ихдээ хэдэн ч мотортой байна гэдгийг хэлэх юм.

4. **ХЭРЭГЖҮҮЛЭЛТ**

```cpp
//1. Дараах зурагт харуулсан класс диаграмын дагуу классуудаг байгуул.
//6. 0..1, 0..n болон 1, 1..n харьцаануудыг зөв программчилсан байх шаардлагатай.
class Person{
    private:
        char*name;
        char*SSName;
        int age;
    public:
        Person(){
           name = new char(1);
           strcpy(name, "");
           SSName = new char(1);
           strcpy(SSName, "");
           age = 0;
        }

        ~Person(){
            delete name;
            delete SSName;
        }

        Person(char* name, char* SSName, int age){
             this->name = new char(strlen(name) + 1);
             strcpy(this->name, name);
             this->SSName = new char(strlen(SSName) + 1);
             strcpy(this->SSName, SSName);
             this->age = age;
        }

        char* getName(){
            return name;
        }

        char* getSSName(){
            return SSName;
        }

        int getAge(){
            return age;
        }

        void setName(char* name){
            delete this->name;
            this->name = new char(strlen(name) + 1);
            strcpy(this->name, name);
        }

        void setSSName(char* SSName){
            delete this->SSName;
            this->SSName = new char(strlen(SSName) + 1);
            strcpy(this->SSName, SSName);
        }

        void setAge(int age){
            this->age = age;
        }

        virtual void printData(){
            cout<<"Name: "<<name<<endl;
            cout<<"SSName: "<<SSName<<endl;
            cout<<"Age: "<<age<<endl;
        }
};

class Spouse:public Person{
    private:
        char* anniversaryData;
    public:
        Spouse():Person(){
            anniversaryData = new char(1);
            strcpy(anniversaryData, "");
        }

        Spouse(char* name, char* SSName, int age, char* anniversaryData):Person(name,
SSName, age){
            this->anniversaryData = new char(strlen(anniversaryData) + 1);
            strcpy(this->anniversaryData, anniversaryData);
        }

        ~Spouse(){
            delete anniversaryData;
        }

        char* getAnniversaryData(){
```

```
91              return anniversaryData;
92          }
93
94          void setAnniversaryData(char* anniData){
95              delete anniversaryData;
96              anniversaryData = new char(strlen(anniData) + 1);
97              strcpy(anniversaryData, anniData);
98          }
99
100         void copyFunc(Spouse &sp){
101             delete anniversaryData;
102             setName(sp.getName());
103             setSSName(sp.getSSName());
104             setAge(sp.getAge());
105             anniversaryData = new char(strlen(sp.anniversaryData) + 1);
106             strcpy(anniversaryData, sp.anniversaryData);
107         }
108
109         void printData(){
110             cout<<"Name: "<<getName()<<endl;
111             cout<<"SSName: "<<getSSName()<<endl;
112             cout<<"Age: "<<getAge()<<endl;
113             cout<<"AnniversaryData: "<<anniversaryData<<endl;
114         }
115 };
116
117 class Division{
118     private:
119         char* divisionName;
120     public:
121         Division(){
122             divisionName = new char(1);
123             strcpy(divisionName, "");
124         }
125
126         Division(char* divisionName){
127             this->divisionName = new char(strlen(divisionName) + 1);
128             strcpy(this->divisionName, divisionName);
129         }
130
131         ~Division(){
132             delete divisionName;
133         }
134
135         char* getDivisionName(){
136             return divisionName;
137         }
138
139         void setDivisionName(char* divisionName){
140             delete this->divisionName;
141             this->divisionName = new char(strlen(divisionName) + 1);
142             strcpy(this->divisionName, divisionName);
143         }
144
145         void copyFunc(Division &d){
146             delete divisionName;
147             divisionName = new char(strlen(d.divisionName) + 1);
148             strcpy(divisionName, d.divisionName);
149         }
150         void printData(){
151             cout<<"Division name: "<<divisionName<<endl;
152         }
153 };
154
155 class JobDescription{
156     private:
157         char* description;
158     public:
159         JobDescription(){
160             description = new char(1);
161             strcpy(description, "");
162         }
163
164         JobDescription(char* description){
165             this->description = new char(strlen(description) + 1);
166             strcpy(this->description, description);
167         }
168
169         ~JobDescription(){
170             delete description;
171         }
172
173         char* getDescription(){
174             return description;
```

```
175         }
176
177         void setDescription(char* description){
178             delete this->description;
179             this->description = new char(strlen(description) + 1);
180             strcpy(this->description, description);
181         }
182
183         void printData(){
184             cout<<"Description: "<<description<<endl;
185         }
186 };
187
188 class Child: public Person{
189     private:
190         char* favoriteToy;
191     public:
192         Child():Person(){
193             favoriteToy = new char(1);
194             strcpy(favoriteToy, "");
195         }
196
197         Child(char* name, char* SSName, int age, char*favoriteToy):Person(name, SSName,
    age){
198             this->favoriteToy = new char(strlen(favoriteToy) + 1);
199             strcpy(this->favoriteToy, favoriteToy);
200         }
201
202         ~Child(){
203             delete favoriteToy;
204         }
205
206         char* getFavoriteToy(){
207             return favoriteToy;
208         }
209
210         void setFavoriteToy(char* favoriteToy){
211             delete this->favoriteToy;
212             this->favoriteToy = new char(strlen(favoriteToy) + 1);
213             strcpy(this->favoriteToy, favoriteToy);
214         }
215
216         void printData(){
217             cout<<"Name: "<<getName()<<endl;
218             cout<<"SSName: "<<getSSName()<<endl;
219             cout<<"Age: "<<getAge()<<endl;
220             cout<<"FavoriteToy: "<<favoriteToy<<endl;
221         }
222 };
223
224 class Employee:public Person{
225     private:
226         char* companyID;
227         char* title;
228         char* startDate;
229         Spouse sp;
230         Division div;
231         vector<JobDescription>jobDesc_vec;
232         vector<Child>ch_vec;
233         void printJobDesc(){
234             cout<<"\nJob description: "<<endl;
235             for(int i = 0; i < getLenJD(); i++){
236                 jobDesc_vec[i].printData();
237             }
238         }
239
240         void printChildren(){
241             cout<<"\nChildren: "<<endl;
242             for(int i = 0; i < getNumChild(); i++){
243                 ch_vec[i].printData();
244             }
245         }
246     public:
247         Employee():Person(){
248             companyID = new char(4);
249             strcpy(companyID, "cID");
250             title = new char(6);
251             strcpy(title, "title");
252             startDate = new char(1);
253             strcpy(startDate, "");
254             Division d;
255             div.copyFunc(d);
256             JobDescription jbd;
257             jobDesc_vec.push_back(jbd);
```

```
258         }
259
260         Employee(char* name, char* SSName, int age, char* companyID, char* title, char*
    startDate, Division &d, JobDescription &jbd):Person(name, SSName, age){
261             this->companyID = new char(strlen(companyID) + 1);
262             strcpy(this->companyID, companyID);
263             this->title = new char(strlen(title) + 1);
264             strcpy(this->title, title);
265             this->startDate = new char(strlen(startDate) + 1);
266             strcpy(this->startDate, startDate);
267             div.copyFunc(d);
268             jobDesc_vec.push_back(jbd);
269         }
270
271         ~Employee(){
272             delete companyID;
273             delete title;
274             delete startDate;
275         }
276
277         char* getCompanyID(){
278             return companyID;
279         }
280
281         char* getTitle(){
282             return title;
283         }
284
285         char* getStartDate(){
286             return startDate;
287         }
288
289         void setCompanyID(char* companyID){
290             delete this->companyID;
291             this->companyID = new char(strlen(companyID) + 1);
292             strcpy(this->companyID, companyID);
293         }
294
295         void setTitle(char *title){
296             delete this->title;
297             this->title = new char(strlen(title) + 1);
298             strcpy(this->title, title);
299         }
300
301         void setStartDate(char* startDate){
302             delete this->startDate;
303             this->startDate = new char(strlen(startDate) + 1);
304             strcpy(this->startDate, startDate);
305         }
306
307         void setDivision(Division &d){
308             div.copyFunc(d);
309         }
310
311         Division* getDivision(){
312             return &div;
313         }
314
315         void setSpouse(Spouse &s){
316             sp.copyFunc(s);
317         }
318
319         Spouse* getSpouse(){
320             return &sp;
321         }
322
323         void addJobDsec(JobDescription &j){
324             jobDesc_vec.push_back(j);
325         }
326
327         int getLenJD(){
328             return jobDesc_vec.size();
329         }
330
331         JobDescription* getJobDesc(int idx){
332             if(0 <= idx && idx < getLenJD())
333                 return &jobDesc_vec[idx];
334             else return NULL;
335         }
336
337         void insertJobDesc(int idx, JobDescription &j){
338             if(0 <= idx && idx < getLenJD()){
339                     jobDesc_vec.insert(jobDesc_vec.begin() + idx, j);
340             }
```

```
341             else cout<<"Wrong position."<<endl;
342         }
343
344         void eraseJobDesc(int idx){
345             if(0 <= idx && idx < getLenJD()){
346                 if(getLenJD() == 1)
347                     cout<<"Not able erase."<<endl;
348                 else{
349                    jobDesc_vec.erase(jobDesc_vec.begin() + idx);
350                 }
351             }else{
352                 cout<<"Wrong position."<<endl;
353             }
354         }
355
356         void popJobDesc(){
357             if(getLenJD() == 1) cout<<"Not able erase."<<endl;
358             else jobDesc_vec.pop_back();
359         }
360
361         void addChild(Child &c){
362             ch_vec.push_back(c);
363         }
364
365         int getNumChild(){
366             return ch_vec.size();
367         }
368
369         Child* getChild(int idx){
370             if(0 <= idx && idx < getNumChild())
371                 return &ch_vec[idx];
372             else return NULL;
373         }
374
375         void insertChild(int idx, Child &c){
376             if(0 <= idx && idx < getNumChild()){
377                     ch_vec.insert(ch_vec.begin() + idx, c);
378             }
379             else cout<<"Wrong position."<<endl;
380         }
381
382         void eraseChild(int idx){
383             if(0 <= idx && idx < getNumChild()){
384                 ch_vec.erase(ch_vec.begin() + idx);
385             }else{
386                 cout<<"Wrong position."<<endl;
387             }
388         }
389
390         void popChild(){
391             ch_vec.pop_back();
392         }
393
394         void printData(){
395             cout<<"\n\nName: "<<getName()<<endl;
396             cout<<"SSName: "<<getSSName()<<endl;
397             cout<<"Age: "<<getAge()<<endl;
398             cout<<"Company ID: "<<companyID<<endl;
399             cout<<"Title: "<<title<<endl;
400             cout<<"Startdate: "<<startDate<<endl;
401             cout<<"Division: "<<endl;
402             div.printData();
403             cout<<"Spouse: "<<endl;
404             sp.printData();
405             printJobDesc();
406             printChildren();
407         }
408 };
```

```cpp
int main(){
    //2. Division болон JobDescription классуудын хэд хэдэн объект байгуул.
    Division div[3];
    JobDescription jd[3];
    int i;
    for(i = 0; i < 3; i++){
        char dName[10];
        cout<<"\nEnter division name in "<<i + 1<<" element"<<endl;
        cout<<"Name: ";
        cin>>dName;
        div[i].setDivisionName(dName);
        char desc[10];
        cout<<"\nEnter job description in "<<i + 1<<" element"<<endl;
        cout<<"Description: ";
        cin>>desc;
        jd[i].setDescription(desc);
    }

    //3. Employee классын хэд хэдэн объект байгуулж тус бүрийн Division, JobDescription -ийг авааж өг.
    Employee a("Bagaa", "MT01252114", 20, "Sync2325", "Sync Systems LLC", "2021.08.18", div[0], jd[0]);
    a.addJobDsec(jd[1]);
    a.addJobDsec(jd[2]);
    a.addJobDsec(jd[3]);
    Employee b("Galaa", "TL89251214", 32, "Malo2325", "Malo LLC", "2011.06.11", div[1], jd[0]);
    b.addJobDsec(jd[1]);
    b.addJobDsec(jd[2]);
    b.addJobDsec(jd[0]);
    b.addJobDsec(jd[1]);
    Employee c("Bat", "OJ98262214", 23, "Tbo2325", "TBO LLC", "2016.07.06", div[3], jd[0]);
    c.addJobDsec(jd[1]);
    c.addJobDsec(jd[3]);

    //4. Employee классын объект тус бүрд Spouse, Child - уудыг тохируулж өг
    Spouse as, bs("Otgoo", "DB91212435", 30, "2013.01.08"), cs;
    Child ch[3];
    for(i = 0; i < 1; i++){
        cout<<"Child: "<<endl;
        char name[10], ssname[10], age, ft[10];
        cout<<"Name: ";
        cin>>name;
        ch[i].setName(name);
        cout<<"SSName: ";
        cin>>ssname;
        ch[i].setSSName(ssname);
        cout<<"Age: ";
        cin>>age;
        ch[i].setAge(age);
        cout<<"Favorite toy: ";
        cin>>ft;
        ch[i].setFavoriteToy(ft);
    }
    a.setSpouse(as);
    b.setSpouse(bs);
    b.addChild(ch[0]);
    b.addChild(ch[1]);
    b.addChild(ch[2]);
    c.setSpouse(cs);
    c.addChild(ch[0]);

    //5. Employee классын объект тус бүрийн бүх мэдээллийг хэвлэ.
    a.printData();
    b.printData();
    c.printData();
    return 0;
}
```

5. **ДҮГНЭЛТ**

Классыг удамшуулж үүсгэснээр өмнө нь бичсэн код болон програмыг дахин хэрэглэх, бичиглэл багасах, гишүүн өгөгдөл болон функцийг дундаа хэрэглэх гэх зэрэг давуу тал боломжууд бүрддэг. Мөн класс хооронд бүрдэл харьцаа үүсгэснээр өөр нэг классын объектыг өөр класс ашиглах боломж бүрддэг бөгөөд энэ нь гишүүн өгөгдлийн төвөгтэй зохион байгуулалтыг багасгаж өгдөг. Классууд бүрдэл харьцаагаар холбогдсон үед бүрдэл харьцааны хориглолтын тавих буюу тухайн класст өгөгдлөөр орсон классын объектуудын байж болох хэмжээг зааж өгөх шаардлагатай. Ингэснээр тухайн класс заавал нэг байх эсвэл олон байх гишүүн өгөгдлийг санамсаргүйгээр устгаж, хоосон болгох зэрэг гэнэтийн үйлдлүүдээс сэргийлэх боломж бүрддэг.

6. **АШИГЛАСАН МАТЕРИАЛ**

1. Объект хандлагат технологийн C++ програмчлал, Ж.Пүрэв, 2008, Улаанбаатар.

7. **ХАВСРАЛ**

```cpp
#include <iostream>
#include <vector>
#include <math.h>
#include <string.h>

using namespace std;

//1. Дараах зурагт харуулсан класс диаграмын дагуу классуудаг байгуул.
//6. 0..1, 0..n болон 1, 1..n харьцаануудыг зөв программчилсан байх шаардлагатай.
class Person{
    private:
        char*name;
        char*SSName;
        int age;
    public:
        Person(){
           name = new char(1);
           strcpy(name, "");
           SSName = new char(1);
           strcpy(SSName, "");
           age = 0;
        }

        ~Person(){
            delete name;
            delete SSName;
        }

        Person(char* name, char* SSName, int age){
             this->name = new char(strlen(name) + 1);
             strcpy(this->name, name);
             this->SSName = new char(strlen(SSName) + 1);
             strcpy(this->SSName, SSName);
             this->age = age;
        }

        char* getName(){
            return name;
        }

        char* getSSName(){
            return SSName;
        }

        int getAge(){
            return age;
        }

        void setName(char* name){
            delete this->name;
            this->name = new char(strlen(name) + 1);
            strcpy(this->name, name);
        }

        void setSSName(char* SSName){
            delete this->SSName;
            this->SSName = new char(strlen(SSName) + 1);
            strcpy(this->SSName, SSName);
        }

        void setAge(int age){
            this->age = age;
        }

        virtual void printData(){
            cout<<"Name: "<<name<<endl;
            cout<<"SSName: "<<SSName<<endl;
            cout<<"Age: "<<age<<endl;
        }
};

class Spouse:public Person{
    private:
        char* anniversaryData;
    public:
        Spouse():Person(){
            anniversaryData = new char(1);
            strcpy(anniversaryData, "");
        }

        Spouse(char* name, char* SSName, int age, char* anniversaryData):Person(name,
SSName, age){
            this->anniversaryData = new char(strlen(anniversaryData) + 1);
            strcpy(this->anniversaryData, anniversaryData);
```

```
84          }
85
86          ~Spouse(){
87              delete anniversaryData;
88          }
89
90          char* getAnniversaryData(){
91              return anniversaryData;
92          }
93
94          void setAnniversaryData(char* anniData){
95              delete anniversaryData;
96              anniversaryData = new char(strlen(anniData) + 1);
97              strcpy(anniversaryData, anniData);
98          }
99
100         void copyFunc(Spouse &sp){
101             delete anniversaryData;
102             setName(sp.getName());
103             setSSName(sp.getSSName());
104             setAge(sp.getAge());
105             anniversaryData = new char(strlen(sp.anniversaryData) + 1);
106             strcpy(anniversaryData, sp.anniversaryData);
107         }
108
109         void printData(){
110             cout<<"Name: "<<getName()<<endl;
111             cout<<"SSName: "<<getSSName()<<endl;
112             cout<<"Age: "<<getAge()<<endl;
113             cout<<"AnniversaryData: "<<anniversaryData<<endl;
114         }
115 };
116
117 class Division{
118     private:
119         char* divisionName;
120     public:
121         Division(){
122             divisionName = new char(1);
123             strcpy(divisionName, "");
124         }
125
126         Division(char* divisionName){
127             this->divisionName = new char(strlen(divisionName) + 1);
128             strcpy(this->divisionName, divisionName);
129         }
130
131         ~Division(){
132             delete divisionName;
133         }
134
135         char* getDivisionName(){
136             return divisionName;
137         }
138
139         void setDivisionName(char* divisionName){
140             delete this->divisionName;
141             this->divisionName = new char(strlen(divisionName) + 1);
142             strcpy(this->divisionName, divisionName);
143         }
144
145         void copyFunc(Division &d){
146             delete divisionName;
147             divisionName = new char(strlen(d.divisionName) + 1);
148             strcpy(divisionName, d.divisionName);
149         }
150         void printData(){
151             cout<<"Division name: "<<divisionName<<endl;
152         }
153 };
154
155 class JobDescription{
156     private:
157         char* description;
158     public:
159         JobDescription(){
160             description = new char(1);
161             strcpy(description, "");
162         }
163
164         JobDescription(char* description){
165             this->description = new char(strlen(description) + 1);
166             strcpy(this->description, description);
167         }
```

```
168
169         ~JobDescription(){
170             delete description;
171         }
172
173         char* getDescription(){
174             return description;
175         }
176
177         void setDescription(char* description){
178             delete this->description;
179             this->description = new char(strlen(description) + 1);
180             strcpy(this->description, description);
181         }
182
183         void printData(){
184             cout<<"Description: "<<description<<endl;
185         }
186 };
187
188 class Child: public Person{
189     private:
190         char* favoriteToy;
191     public:
192         Child():Person(){
193             favoriteToy = new char(1);
194             strcpy(favoriteToy, "");
195         }
196
197         Child(char* name, char* SSName, int age, char*favoriteToy):Person(name, SSName,
    age){
198             this->favoriteToy = new char(strlen(favoriteToy) + 1);
199             strcpy(this->favoriteToy, favoriteToy);
200         }
201
202         ~Child(){
203             delete favoriteToy;
204         }
205
206         char* getFavoriteToy(){
207             return favoriteToy;
208         }
209
210         void setFavoriteToy(char* favoriteToy){
211             delete this->favoriteToy;
212             this->favoriteToy = new char(strlen(favoriteToy) + 1);
213             strcpy(this->favoriteToy, favoriteToy);
214         }
215
216         void printData(){
217             cout<<"Name: "<<getName()<<endl;
218             cout<<"SSName: "<<getSSName()<<endl;
219             cout<<"Age: "<<getAge()<<endl;
220             cout<<"FavoriteToy: "<<favoriteToy<<endl;
221         }
222 };
223
224 class Employee:public Person{
225     private:
226         char* companyID;
227         char* title;
228         char* startDate;
229         Spouse sp;
230         Division div;
231         vector<JobDescription>jobDesc_vec;
232         vector<Child>ch_vec;
233         void printJobDesc(){
234             cout<<"\nJob description: "<<endl;
235             for(int i = 0; i < getLenJD(); i++){
236                 jobDesc_vec[i].printData();
237             }
238         }
239
240         void printChildren(){
241             cout<<"\nChildren: "<<endl;
242             for(int i = 0; i < getNumChild(); i++){
243                 ch_vec[i].printData();
244             }
245         }
246     public:
247         Employee():Person(){
248             companyID = new char(4);
249             strcpy(companyID, "cID");
250             title = new char(6);
```

```
251             strcpy(title, "title");
252             startDate = new char(1);
253             strcpy(startDate, "");
254             Division d;
255             div.copyFunc(d);
256             JobDescription jbd;
257             jobDesc_vec.push_back(jbd);
258         }
259 
260         Employee(char* name, char* SSName, int age, char* companyID, char* title, char*
    startDate, Division &d, JobDescription &jbd):Person(name, SSName, age){
261             this->companyID = new char(strlen(companyID) + 1);
262             strcpy(this->companyID, companyID);
263             this->title = new char(strlen(title) + 1);
264             strcpy(this->title, title);
265             this->startDate = new char(strlen(startDate) + 1);
266             strcpy(this->startDate, startDate);
267             div.copyFunc(d);
268             jobDesc_vec.push_back(jbd);
269         }
270 
271         ~Employee(){
272             delete companyID;
273             delete title;
274             delete startDate;
275         }
276 
277         char* getCompanyID(){
278             return companyID;
279         }
280 
281         char* getTitle(){
282             return title;
283         }
284 
285         char* getStartDate(){
286             return startDate;
287         }
288 
289         void setCompanyID(char* companyID){
290             delete this->companyID;
291             this->companyID = new char(strlen(companyID) + 1);
292             strcpy(this->companyID, companyID);
293         }
294 
295         void setTitle(char *title){
296             delete this->title;
297             this->title = new char(strlen(title) + 1);
298             strcpy(this->title, title);
299         }
300 
301         void setStartDate(char* startDate){
302             delete this->startDate;
303             this->startDate = new char(strlen(startDate) + 1);
304             strcpy(this->startDate, startDate);
305         }
306 
307         void setDivision(Division &d){
308             div.copyFunc(d);
309         }
310 
311         Division* getDivision(){
312             return &div;
313         }
314 
315         void setSpouse(Spouse &s){
316             sp.copyFunc(s);
317         }
318 
319         Spouse* getSpouse(){
320             return &sp;
321         }
322 
323         void addJobDsec(JobDescription &j){
324             jobDesc_vec.push_back(j);
325         }
326 
327         int getLenJD(){
328             return jobDesc_vec.size();
329         }
330 
331         JobDescription* getJobDesc(int idx){
332             if(0 <= idx && idx < getLenJD())
333                 return &jobDesc_vec[idx];
```

```
            else return NULL;
        }

        void insertJobDesc(int idx, JobDescription &j){
            if(0 <= idx && idx < getLenJD()){
                    jobDesc_vec.insert(jobDesc_vec.begin() + idx, j);
            }
            else cout<<"Wrong position."<<endl;
        }

        void eraseJobDesc(int idx){
            if(0 <= idx && idx < getLenJD()){
                if(getLenJD() == 1)
                    cout<<"Not able erase."<<endl;
                else{
                   jobDesc_vec.erase(jobDesc_vec.begin() + idx);
                }
            }else{
                cout<<"Wrong position."<<endl;
            }
        }

        void popJobDesc(){
            if(getLenJD() == 1) cout<<"Not able erase."<<endl;
            else jobDesc_vec.pop_back();
        }

        void addChild(Child &c){
            ch_vec.push_back(c);
        }

        int getNumChild(){
            return ch_vec.size();
        }

        Child* getChild(int idx){
            if(0 <= idx && idx < getNumChild())
                return &ch_vec[idx];
            else return NULL;
        }

        void insertChild(int idx, Child &c){
            if(0 <= idx && idx < getNumChild()){
                    ch_vec.insert(ch_vec.begin() + idx, c);
            }
            else cout<<"Wrong position."<<endl;
        }

        void eraseChild(int idx){
            if(0 <= idx && idx < getNumChild()){
                ch_vec.erase(ch_vec.begin() + idx);
            }else{
                cout<<"Wrong position."<<endl;
            }
        }

        void popChild(){
            ch_vec.pop_back();
        }

        void printData(){
            cout<<"\n\nName: "<<getName()<<endl;
            cout<<"SSName: "<<getSSName()<<endl;
            cout<<"Age: "<<getAge()<<endl;
            cout<<"Company ID: "<<companyID<<endl;
            cout<<"Title: "<<title<<endl;
            cout<<"Startdate: "<<startDate<<endl;
            cout<<"Division: "<<endl;
            div.printData();
            cout<<"Spouse: "<<endl;
            sp.printData();
            printJobDesc();
            printChildren();
        }
};

int main(){
    //2. Division болон JobDescription классуудын хэд хэдэн объект байгуул.
    Division div[3];
    JobDescription jd[3];
    int i;
    for(i = 0; i < 3; i++){
        char dName[10];
        cout<<"\nEnter division name in "<<i + 1<<" element"<<endl;
```

```
418             cout<<"Name: ";
419             cin>>dName;
420             div[i].setDivisionName(dName);
421             char desc[10];
422             cout<<"\nEnter job description in "<<i + 1<<" element"<<endl;
423             cout<<"Description: ";
424             cin>>desc;
425             jd[i].setDescription(desc);
426         }
427
428         //3. Employee классын хэл хэлэн объект байгуулж тус бүрийн Division, JobDescription
    -ийг зааж өг.
429         Employee a("Bagaa", "MT01252114", 20, "Sync2325", "Sync Systems LLC", "2021.08.18",
    div[0], jd[0]);
430         a.addJobDsec(jd[1]);
431         a.addJobDsec(jd[2]);
432         a.addJobDsec(jd[3]);
433         Employee b("Galaa", "TL89251214", 32, "Malo2325", "Malo LLC", "2011.06.11", div[1],
    jd[0]);
434         b.addJobDsec(jd[1]);
435         b.addJobDsec(jd[2]);
436         b.addJobDsec(jd[0]);
437         b.addJobDsec(jd[1]);
438         Employee c("Bat", "OJ98262214", 23, "Tbo2325", "TBO LLC", "2016.07.06", div[3], jd[0]);
439         c.addJobDsec(jd[1]);
440         c.addJobDsec(jd[3]);
441
442         //4. Employee классын объект тус бүрд Spouse, Child – уудыг тохируулж өг
443         Spouse as, bs("Otgoo", "DB91212435", 30, "2013.01.08"), cs;
444         Child ch[3];
445         for(i = 0; i < 1; i++){
446             cout<<"Child: "<<endl;
447             char name[10], ssname[10], age, ft[10];
448             cout<<"Name: ";
449             cin>>name;
450             ch[i].setName(name);
451             cout<<"SSName: ";
452             cin>>ssname;
453             ch[i].setSSName(ssname);
454             cout<<"Age: ";
455             cin>>age;
456             ch[i].setAge(age);
457             cout<<"Favorite toy: ";
458             cin>>ft;
459             ch[i].setFavoriteToy(ft);
460         }
461         a.setSpouse(as);
462         b.setSpouse(bs);
463         b.addChild(ch[0]);
464         b.addChild(ch[1]);
465         b.addChild(ch[2]);
466         c.setSpouse(cs);
467         c.addChild(ch[0]);
468
469         //5. Employee классын объект тус бүрийн бүх мэдээллийг хэвлэ.
470         a.printData();
471         b.printData();
472         c.printData();
473         return 0;
474     }
475
476
```