Data Structure Mid-term examination (2018.10.30)

1. Programming life cycle 8단계를 제시하고, 개발될 시스템의 기능과 개발비용이 결정되는 단계를 제시하고 보다 구체적으로 설명하라. (10점)

Problem analysis-Requirement definition-design-implementation-testing-delibery-operation – maintenance

Requirement Definition: 문제해결을 위해서 어떤 기능들이 어떠한 조건하에서 동작하는 지를 정의하는 것으로 시스템의 기능과 동작범위가 이 단계에서 결정됨.

1. 객체지향 프로그래밍(OOP)이 절차지향 프로그래밍에 비해서 재사용성이 높은 이유를 설명하라. (10점)

자료를 구분하고 자료별로 표현방법과 사용방법을 하나로 묶어서 객체로 표현한다. 자료와 자료는 분명히 구분되기 때문에 독립적인 모듈을 구성할 수 있어서 재사용성이 높다.

1. 일반적으로 재귀함수는 반복함수에 비해서 overhead가 크다. 그럼에도 재귀함수를 채택하는 가장 큰 이유는 무엇인가? 그리고 재귀함수를 사용할 경우 발생하는 overhead의 요인은 무엇인가? (10점)

문제에 따라서 재귀함수가 보다 직관적이고 짧은 코드를 생성하는 경우가 있다. Overhead는 activation record 생성 및 관리 때문에 발생한다. 이를 줄이기 위해서는 activation record size를 줄이고 stack에 쌓이는 깊이를 줄여야 한다.

1. class에서 Information hiding이 어떻게 이루어지는 지를 설명하고 Information hiding으로 인한 효과를 설명하라. (10점)

자료를 private로 선언하고 public 멤버함수를 통해서 통제된 자료 검색을 허용한다. 객체의 상세를 사용자와 분리함으로써 정보를 보호하고 사용자가 시스템의 전체적인 구조 파악을 쉬게 하고 보다 안정적이고 유연한 프로그램을 생성한다.

1. 아래 수식의 복잡도를 Big-O 표기법으로 표현하라. (과정이 없으면 0점 처리) (10점)

F(N) = 1+ N\*1000\*400 + (N+(N-1)+(N-2)+,,,2+1)\*10000+N\*N/100

= 1+400000N+ 5000\*(N2-N)+0.01N2= 1+395000N+ 5000.01N2=O(N2)

1. **[프로그램 과제]** 아래 정의된 class들을 바탕으로 다음 물음에 답하라.

|  |  |  |  |
| --- | --- | --- | --- |
| class appType {  public:  void FindNDispGenrUsingMlist();  void FindNDispGenrUsingGlist();  private:  listType<mType> mList;  listType<gType> gList;  } | class mType {  public:  void DisplayInfo();  void SetMName(string name);  void GetGenrName(string& n);  private:  string name;  } | class gType {  public:  ?? DetailInfo(??);  void DisplayInfo();  void GetGenreName(string& name);  void SetGenreName(string name);  private:  string genrName;  listType<SimpleType> smList;  } | template<class itemType>  class listType {  public:  void Find(itemType& item);  int GetNext(itemType& item);  void ResetList();  private:  NodeType<itemType>\* listStart;  } |
| [mType] DisplayInfo(): 자세한 음악정보 화면출력, SetMName(): 음악객체의 이름을 set, GetGenrName(n): 음악이 속한 장르 이름을 리턴  [gType] GetGenrName(): 장르 이름을 리턴, SetGenrName(): 장르이름을 set, DisplayInfo(): 장르 정보를 화면에 출력  [listType] FindNDisplay(item): item에 저장된 이름으로 레코드를 검색하여 자세한 정보를 화면에 출력  \*\* [SimpleType]에 곡명을 리턴하는 GetSmName(string& name)이 정의되었다고 가정한다. | | | |

* 1. FindNDispGenrUsingMlist() 를 구현하라. 키보드로부터 장르명을 입력 받고, mList에서 해당 장르에 속하는 모든 곡의 자세한 정보를 화면에 출력한다. (10점)

Void appType :: FindNDisplayGenreUsingMlist(){

mType mItem; string name, sTemp;

cout<<”장르로 이름을 입력🡪”; cin >> name;

mList.ResetList();

while(mList.GetNextItem(mItem)) {

mItem.GetGenrName(sTemp);

if (sTemp==name) mItem.Display(); }

}

* 1. A와 동일한 기능을 수행하는 FindNDispGenrUsingGlist() 는 gList 와 mList 를 함께 이용한다. 키보드로부터 장르명을 입력 받고, gList에서 해당 장르를 찾아 DetailInfo() 함수를 호출한다. DetailInfo() 를 통해 해당 장르에서 smList 의 복사본이나 포인터를 받고, 이를 이용하여 mList 를 검색해 곡의 자세한 정보를 화면에 출력한다. 이 때, 필요한 DetailInfo() 함수를 구현하고 이것과 기존 멤버함수들을 이용하여 FindNDispGenrUsingGlist() 를 구현한다. (20점)

listType<SimpleType>\* gType::DetailInfo(){ return msList;}

or

void gType::DetailInfo(listType<SimpleType>& temList) { tempList=msList;} //복사생성자 정의 필수

void appType::FindNDispGenrUsingGlist(){

// 장르 이름을 입력받아 해당 장르 객체를 gList에서 찾는다.

gType gItem; string name;

cout<<”장르로 이름을 입력🡪”; cin >> name;

gItem.SetGenrName(name);

gList.Find(gItem); // 항상 존재한다고 가정

// 찾은 장르 객체에 소속된 곡 리스트를 가져온다

listType<SimpleType>\* smList;

smList=gItem.DetailInfo();

// 곡 리스트를 스캔하며 소속된 곡의 자세한 정보를 mList에서 찾아 화면 출력

smList->ResetList();

SimpleType sItem; String mName; mType mItem;

while (smList->GetNextItem(sItem)){

sItem.GetSmName(mName);

mItem.SetMName(mName);

mList.Find(mItem);

mItem.DisplayInfo();

}

}

* 1. B 방법을 사용하는 이유와 이 방법이 어떤 조건하에서 유리한 지를 설명하라. (10점)

전체 리스트를 스캔하지 않고도 특정장르에 속한 곡들을 찾을 수 있다. 곡명을 이용하여 mList에서 찾아야 하기 때문에 Find함수가 binary search처럼 빨라야 유리하다. 소속된 곡 수가 M일때 계산량은 Find()가 N에 비례하면 MN logN에 비래하면 MLogM에 비례한다.

* 1. B를 구현하기 위한 방법 중 하나는 상속을 이용하는 것이다. B를 구현하기 위한 base class, appType class, gType class를 정의하고 FindNDispGenrUsingGlist() 함수를 구현하라. (10+보너스 10점)

Class baseType {

Protected:

static listType<mType> mList;

static listType<gType> gList;

Public: }

listType<mType> baseType::mList;

listType<gType> baseType::gLst;

Class appType::public baseType {

Public: voidFindNDisplayGenreUsingGlist();

)

Class gType::public baseType {

String genreName;

listType<SimpleType> smList;

public:

void DisplayDetailInfo(); }

void appType:: FindNDispGenrUsingGlist(){

// 장르 이름을 입력받아 해당 장르 객체를 gList에서 찾는다.

gType gItem; string name;

cout<<”장르로 이름을 입력🡪”; cin >> name;

gItem.SetGenrName(name);

gList.Find(gItem); // 항상 존재한다고 가정

gItem.DisplayDetailInfo();

}

Void gType:DisplayDetailInfo(){

// 곡 리스트를 스캔하며 소속된 곡의 자세한 정보를 mList에서 찾아 화면 출력

smList.ResetList();

SimpleType sItem; String mName; mType mItem;

while (smList.GetNextItem(sItem)){

sItem.GetSmName(mName);

mItem.SetMName(mName);

mList.Find(mItem);

mItem.DisplayInfo();

}}