

|  |
| --- |
| Binary Tree (연결구조 방식) |
| **과제 5장 보고서(2번 - 연결구조 방식)** |



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **제 출 일** | **2014. 0** |  | **전 공** | **Business & CSE** |
| **과 목** | **자료구조론** |  | **학 번** | **20101215** |
| **담당교수** | **이 현 아** |  | **이 름** | **정 원 영** |

**1. Complete Binary Tree의 원리**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | **Preorder** | **Inorder** | | **Postorder** | | template <class T>  void BinaryTree<T>::Preorder(TreeNode<T>\* pNode){  if (pNode == NULL)  return;  cout << pNode->Data << " ";  Preorder(pNode->LeftChild);  Preorder(pNode->RightChild);    } | template <class T>  void BinaryTree<T>::Inorder(TreeNode<T>\* pNode){  if (pNode == NULL)  return;  Inorder(pNode->LeftChild);  cout << pNode->Data << " ";  Inorder(pNode->RightChild);  } | | template <class T>  void BinaryTree<T>::Postorder(TreeNode<T>\* pNode){  if (pNode == NULL)  return;  Postorder(pNode->LeftChild);  Postorder(pNode->RightChild);  cout << pNode->Data << " ";  } | | **Levelorder** | | | | | template <class T>  void BinaryTree<T>::Levelorder(TreeNode<T>\* pNode){  int height = GetHeight();  int i;  for (int i = 1; i <= height; i++)  {  PrintLevelorder(pNode, i);  }    } | | template <class T>  void BinaryTree<T>::PrintLevelorder(TreeNode<T>\* pNode, int level){    if (pNode == NULL)  return;  if (level == 1)  cout << pNode->Data << " ";  else if (level > 1)  {  PrintLevelorder(pNode->LeftChild, level - 1);  PrintLevelorder(pNode->RightChild, level - 1);  }  } | |   **자료구조론 교제에 있는 원리를 적용해본 결과** |

**2. 코드 결과**

|  |
| --- |
| /\*  금오공과대학교  과제: 방학(Binary Tree) 과제 - 1번(연결구조 구현)  학과: Business & CSE  학번: 20101215  이름: 정원영  \*/  #include <iostream>  #include <math.h>  #include <string>  #include "binarytree.h"  using namespace std;  // 1. Visitor 구축  const int INVEST = 0; // 조사  const int MAKE = 1; // 생성  const char StrToCh(const string D);  /\* 완전 이진트리 갯수 계산 \*/  int GetBinaryCount(int Num){  int i = 0;  int Binary = 0;  int Result = 0;  // 원리 = 2^0 + 2^1 + ... + 2^n;  while (1)  {  Binary = (int)pow(2, (double)i);  if ((Binary + Result) > Num)  break;  else  Result += Binary;  i++;  }  return Result;  }  /\* 데이터 값 트리 구축 \*/  BinaryTree<int>\* TreeCreate(Visit<int>\* \_pVisit, int \_UserSize){  // 태스트 구간  VisitNode<int>\* nextVisit = \_pVisit->GetHeadNode();  VisitNode<int>\* prevVisit = nextVisit;  BinaryTree<int>\* NewTree = new BinaryTree<int>();  TreeNode<int>\* NewNode, \*TopNode = NULL;  TreeNode<int>\* NextNode;  int Count = 1;  int BinaryCount = GetBinaryCount(\_UserSize);  int GetSize = 0;  int GetPosition = LEFT; // 초기 왼쪽 방향  bool First = true;  // 루트 생성  NewNode = NewTree->CreateNode(nextVisit->data);  NewTree->SetRootNode(NewNode);  NextNode = NewTree->GetRootNode();  TopNode = NewTree->GetRootNode();  // 노드 생성  nextVisit = nextVisit->link;  while (nextVisit != NULL)  {  int Decision;  GetSize = 0;  TopNode = NewTree->GetRootNode();  // 왼쪽 구간 구축  if (Count < BinaryCount)  {  // 위치 설정  while (GetSize < nextVisit->height)  {  Decision = NONE;  if (TopNode->LeftChild == NULL && TopNode->RightChild == NULL)  {  Decision = ALL;  } // End of if  if (Decision == NONE){  if (TopNode->LeftChild != NULL && TopNode->RightChild == NULL)  {  Decision = LEFT;  }  } // End of if  if (Decision == NONE){  if (TopNode->LeftChild != NULL && TopNode->RightChild != NULL)  {  Decision = RIGHT;  }  } // End of if  switch (Decision)  {  case LEFT:  TopNode = TopNode->LeftChild;  break;  case RIGHT:  TopNode = TopNode->RightChild;  break;  } // End of Switch  GetSize++;  } // End of While  } // End of if(1/2)    // 오른쪽 구간 구축  else{    // 위치 설정  while (GetSize < nextVisit->height)  {  Decision = NONE;  if (TopNode->LeftChild == NULL && TopNode->RightChild == NULL)  {  Decision = ALL;  } // End of if  if (Decision == NONE)  {  if (TopNode->LeftChild != NULL && TopNode->RightChild != NULL)  if ((GetSize + 1) == nextVisit->height)  Decision = LEFT;  else  Decision = RIGHT;  } // End of if  // 결정  switch (Decision)  {  case LEFT:  TopNode = TopNode->LeftChild;  break;  case RIGHT:  TopNode = TopNode->RightChild;  break;  } // End of Switch  GetSize++;  } // End of While  } // End of if(2/2)  // 노드가 비어있을 때  if (TopNode == NULL)  {  NewNode = NewTree->CreateNode(nextVisit->data);  NewTree->Insert(LEFT, TopNode, NewNode);  }  // 왼쪽이 비어있을 때 삽입  else if (TopNode->LeftChild == NULL)  {  NewNode = NewTree->CreateNode(nextVisit->data);  NewTree->Insert(LEFT, TopNode, NewNode);  }  // 오른쪽이 비어있을 때 삽입  else if (TopNode->RightChild == NULL)  {  NewNode = NewTree->CreateNode(nextVisit->data);  NewTree->Insert(RIGHT, TopNode, NewNode);  }  Count++;  nextVisit = nextVisit->link;  }  return NewTree;  }  /\* GetProcess  1. 조사 단계(INVEST)  1-1. GetProcess(std::string D, int Type, int &pCount)  2. 생성 단계(MAKE)  2-1. GetProcess(std::string D, int Type, int &pCount, Visit<int>\* pVisit)  \*/  void GetProcess(std::string D, int Type, int \*pCount, Visit<int>\* pVisit = NULL){  bool First = false; // 처음여부 파악  int EndIDX = D.length();  int Height = 0;  int Number = -1; // 임시 숫자 보관  string NumSt;  string LevelSt;  // 생성 단계  if (Type == MAKE)  {  if (pVisit->GetHeadNode() == NULL)  First = true;  }  for (int StartIDX = 0; StartIDX < EndIDX; StartIDX++)  {  const char chData = StrToCh(D.substr(StartIDX, 1));  // 숫자일 때  if (isdigit(chData))  NumSt.push\_back(chData);  // 깊이 파악  if (chData == '-')  LevelSt.push\_back(chData);  // 공백일 때  if (chData == ' ' || (StartIDX + 1) == EndIDX)  {  switch (Type)  {  case INVEST:  // 숫자 배출(갯수 파악 목적)  if (!NumSt.empty())  {  NumSt.erase();  \*pCount = \*pCount + 1;  }  break;  case MAKE:  // 깊이 증가 판단  if (!LevelSt.empty())  {  Height++;  LevelSt.erase();  }  // 숫자 배출(갯수 파악 목적)  if (!NumSt.empty())  {  Number = atoi(NumSt.c\_str());  NumSt.erase();  if (Height == ROOT)  pVisit->InsertNode(pVisit->CreateNode(ROOT, Number, Height)); // 루트노드 생성  else{  if (((StartIDX + 1) == EndIDX) && Height == 1) // 마지막에 도달할 때  {  pVisit->InsertNode(pVisit->CreateNode(NODE, Number, pVisit->GetHeight() - Height)); // 일반노드 생성  }  else  {  pVisit->InsertNode(pVisit->CreateNode(NODE, Number, Height)); // 일반노드 생성  }  } // End of if  \*pCount = \*pCount + 1;  }  break;  default:  break;  } // End of Switch  } // End of if  } // End of For  // 첫 입력일 때  if (First == true)  pVisit->SetHeight(Height);  }  int main()  {  int UserSize;  int Count = 0; // 노드 수  Visit<int>\* pVisitor = new Visit<int>(); // Visitor 노드  BinaryTree<int>\* Tree;  cout << "노드 개수를 입력하세요 : ";  cin >> UserSize;  while (Count < UserSize)  {  int CurCount = 0;  string UserData;  std::getline(cin, UserData);  GetProcess(UserData, INVEST, &CurCount); // 1. 조사  if ((Count + CurCount) > UserSize)  cout << "노드 갯수 초과" << endl;  else  {  Count += CurCount;  CurCount = 0;  GetProcess(UserData, MAKE, &CurCount, pVisitor); // 1. 조사  }  }    Tree = TreeCreate(pVisitor, UserSize);  // 소멸 Visitor  delete pVisitor;  // VLR  cout << "preorder : ";  Tree->Preorder(Tree->GetRootNode());  cout << endl;  // LVR  cout << "inorder : ";  Tree->Inorder(Tree->GetRootNode());  cout << endl;  // LRV  cout << "postorder : ";  Tree->Postorder(Tree->GetRootNode());  cout << endl;  // height 기준 Traversal  cout << "level order : ";  Tree->Levelorder(Tree->GetRootNode());  cout << endl;  delete Tree;  }  const char StrToCh(const string D){  return D.at(0);  } |

Figure 1) Main.cpp

|  |
| --- |
| /\*  금오공과대학교  과제: 방학(Binary Tree) 과제 - 1번(연결구조 구현)  학과: Business & CSE  학번: 20101215  이름: 정원영  \*/  #ifndef \_BINARYTREE\_H\_  #define \_BINARYTREE\_H\_  #include <iostream>  #include <string>  // 1. 형식 구분  const int ROOT = 0; // 타입:루트  const int NODE = 1; // 타입:노드  // 2. 방향 구분(DECISION에 이용 - Tree 자료 구축에 사용)  const int LEFT = 0; // 왼쪽  const int RIGHT = 1; // 오른쪽  const int ALL = 2; // 모두 널  const int NONE = 3; // 초기  // 3. Visit 설계  template <class T>  class VisitNode{  public:  int type;  int height;  T data;  VisitNode<T>\* link;  };  template <class T>  class Visit{  public:  void InsertNode(VisitNode<T>\* pNode);  VisitNode<T>\* CreateNode(int Type, T Element, int height);  VisitNode<T>\* GetTopNode();  VisitNode<T>\* GetHeadNode();  void SetHeight(int height);  int GetHeight();  Visit();  ~Visit();  private:  VisitNode<T>\* HeadNode;  VisitNode<T>\* TopNode;  int MaxHeight;  };  // 4. Tree 설계  template <class T>  class TreeNode{  public:  TreeNode<T>\* LeftChild;  T Data;  TreeNode<T>\* RightChild;  };  template <class T>  class BinaryTree{  public:  BinaryTree();  ~BinaryTree();  void DestoryTree(TreeNode<T>\* pNode);  void DestoryNode(TreeNode<T>\* pNode);  TreeNode<T>\* CreateNode(T Element);  void Insert(int Type, TreeNode<T>\* CenterNode, TreeNode<T>\* pNode);  TreeNode<T>\* GetRootNode();  void SetRootNode(TreeNode<T>\* pNode);  void Preorder(TreeNode<T>\* pNode);  void Inorder(TreeNode<T>\* pNode);  void Postorder(TreeNode<T>\* pNode);  void Levelorder(TreeNode<T>\* pNode);  void PrintLevelorder(TreeNode<T>\* pNode, int level);  int GetHeight();  private:  TreeNode<T>\* rootNode;  };  #include "visitor\_detail.h"  #include "binarytree\_detail.h"  #endif |

**Figure 2) Binarytree.h**

|  |
| --- |
| /\*  금오공과대학교  과제: 방학(Binary Tree) 과제 - 1번(연결구조 구현)  학과: Business & CSE  학번: 20101215  이름: 정원영  \*/  #ifndef \_BINARYTREE\_DETAIL\_H\_  #define \_BINARYTREE\_DETAIL\_H\_  #include "binarytree.h"  template <class T>  BinaryTree<T>::BinaryTree(){  rootNode = NULL;  }  template <class T>  BinaryTree<T>::~BinaryTree(){  DestoryTree(GetRootNode());  }  template <class T>  void BinaryTree<T>::DestoryTree(TreeNode<T>\* pNode){  if (pNode != NULL)  {  DestoryTree(pNode->LeftChild);  DestoryTree(pNode->RightChild);  DestoryNode(pNode);  }  }  template <class T>  void BinaryTree<T>::DestoryNode(TreeNode<T>\* pNode){  delete pNode;  }  template <class T>  void BinaryTree<T>::Insert(int Type, TreeNode<T>\* CenterNode, TreeNode<T>\* pNode){  switch (Type)  {  case LEFT:  CenterNode->LeftChild = pNode;  break;  case RIGHT:  CenterNode->RightChild = pNode;  break;  }  }  template <class T>  TreeNode<T>\* BinaryTree<T>::CreateNode(T Element){  TreeNode<T>\* newNode = new TreeNode<T>();  newNode->Data = Element;  newNode->LeftChild = NULL;  newNode->RightChild = NULL;  return newNode;  }  template <class T>  TreeNode<T>\* BinaryTree<T>::GetRootNode(){  return rootNode;  }  template <class T>  void BinaryTree<T>::SetRootNode(TreeNode<T>\* pNode){  if (rootNode == NULL)  rootNode = pNode;  }  template <class T>  void BinaryTree<T>::Preorder(TreeNode<T>\* pNode){  if (pNode == NULL)  return;  cout << pNode->Data << " ";  Preorder(pNode->LeftChild);  Preorder(pNode->RightChild);    }  template <class T>  void BinaryTree<T>::Inorder(TreeNode<T>\* pNode){  if (pNode == NULL)  return;  Inorder(pNode->LeftChild);  cout << pNode->Data << " ";  Inorder(pNode->RightChild);  }  template <class T>  void BinaryTree<T>::Postorder(TreeNode<T>\* pNode){  if (pNode == NULL)  return;  Postorder(pNode->LeftChild);  Postorder(pNode->RightChild);  cout << pNode->Data << " ";  }  template <class T>  void BinaryTree<T>::Levelorder(TreeNode<T>\* pNode){  int height = GetHeight();  int i;  for (int i = 1; i <= height; i++)  {  PrintLevelorder(pNode, i);  }    }  template <class T>  void BinaryTree<T>::PrintLevelorder(TreeNode<T>\* pNode, int level){    if (pNode == NULL)  return;  if (level == 1)  cout << pNode->Data << " ";  else if (level > 1)  {  PrintLevelorder(pNode->LeftChild, level - 1);  PrintLevelorder(pNode->RightChild, level - 1);  }  }  template <class T>  int BinaryTree<T>::GetHeight(){  int height = 0;  TreeNode<int> \*pNode = rootNode;  while (pNode != NULL)  {  height++;  pNode = pNode->LeftChild;  }  return height;  }  #endif |

**Figure 3) binarytree\_detail.h**

|  |
| --- |
| #ifndef \_VISITOR\_DETAIL\_H\_  #define \_VISITOR\_DETAIL\_H\_  #include "binarytree.h"  using namespace std;  template <class T>  void Visit<T>::SetHeight(int height){  MaxHeight = height;  }  template <class T>  int Visit<T>::GetHeight(){  return MaxHeight;  }  template <class T>  VisitNode<T>\* Visit<T>::CreateNode(int Type, T Element, int height){  VisitNode<T>\* pNode = new VisitNode<T>();  pNode->type = Type;  pNode->data = Element;  pNode->height = height;  pNode->link = NULL;  return pNode;  }  template <class T>  void Visit<T>::InsertNode(VisitNode<T>\* pNode){  if (TopNode != NULL){  TopNode->link = pNode;  TopNode = TopNode->link;  }  else  {  TopNode = pNode;  HeadNode = TopNode;  }  }  template <class T>  VisitNode<T>\* Visit<T>::GetTopNode(){  return TopNode;  }  template <class T>  VisitNode<T>\* Visit<T>::GetHeadNode(){  return HeadNode;  }  template <class T>  Visit<T>::Visit(){  HeadNode = NULL;  TopNode = NULL;  }  template <class T>  Visit<T>::~Visit(){  VisitNode<T>\* prevNode;  while (TopNode != NULL)  {  prevNode = TopNode;  TopNode = TopNode->link;  delete prevNode;  }  }  #endif |

**Figure 4) Visitor\_detail.h**