텍스트, 로고, 엠블럼, 폰트이(가) 표시된 사진

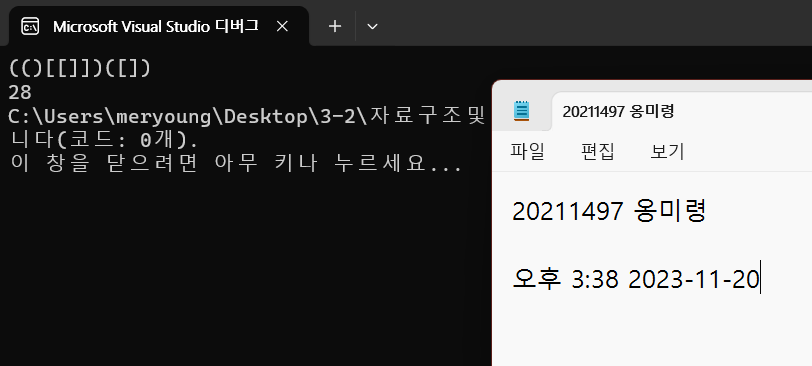
자동 생성된 설명

|  |  |
| --- | --- |
| 교과목 | **자료구조및알고리즘** |
| 분반 | **02** |
| 소속 | **전자공학부 지능전자공학전공** |
| 학번 | **20211497** |
| 이름 | **옹미령** |

1. 괄호의 값: (BOJ) BaekJoon 2504
2. 소스 코드

|  |
| --- |
| #include <iostream>  #include <stack>  #include <string>  using namespace std;  int main()  {  stack<char> S;// 문자를 저장하는 스택 선언  string str;  int ans = 0, temp = 1;// 결과값 및 임시 값 초기화  cin >> str;  for (int i = 0; i < str.length(); i++)  {  if (str[i] == '(')  {  temp \*= 2;// 여는 소괄호일 경우, temp 값을 2배로 증가시키고 스택에 추가  S.push('(');  }  else if (str[i] == '[')  {  temp \*= 3;  S.push('[');  }  else if (str[i] == ')')  {  //괄호가 잘 닫히지 않은 경우  if (S.empty() || S.top() != '(')  {  ans = 0; break;  }  // 짝이 맞는 소괄호일 경우, 결과값에 현재 temp 값을 더하고 temp를 나누기 2  if (str[i - 1] == '(') ans += temp;    temp /= 2;  S.pop();  }  else if (str[i] == ']')  {  //괄호가 잘 닫히지 않은 경우  if (S.empty() || S.top() != '[')  {  ans = 0; break;  }  if (str[i - 1] == '[') ans += temp;  temp /= 3;  S.pop();  }  }  if (!S.empty()) ans = 0;//괄호가 짝이 맞지 않는 경우  cout << ans;// 결과값 출력  return 0;  } |

1. 실행 결과



텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

1. 연구 조사

이 문제는 주어진 괄호 문자열에서 올바른 괄호 쌍의 개수를 계산하는 문제이다. 여는 괄호 '('와 '['에 대해 각각 2배와 3배로 가중치를 부여하고, 닫는 괄호가 나올 때 해당 괄호와 짝이 맞으면 결과값에 가중치를 더하고 가중치를 나누어 줄여가며 계산한다. 괄호쌍이 잘 닫혔을 때 최종 결과값, 그렇지 않으면 문제에 나온대로 0을 출력한다. 가중치를 부여하는 부분은 ans에 더하고 빼지는 변수 temp를 이용했다.

1. 스택 구현: 순차 자료구조
2. 소스 코드

|  |
| --- |
| #include <iostream>  #define stackMAXSIZE 10  using namespace std;  template<typename E>  class arrStack  {  public:  arrStack();  ~arrStack();  void push(const E& e); //데이터 삽입  void pop(); //데이터 삭제  E peek() const; //top 출력  int size() const; //스택 크기 출력  bool empty() const; //스택 빈 여부 확인  bool full() const; //스택 찬 여부 확인  void printStack() const;//스택 전체 출력  private:  E stack[stackMAXSIZE]; //스택 배열  int top; //스택의 탑(배열의 가장 큰 인덱스)  };  template<typename E>  arrStack<E>::arrStack() : top(-1) {}  template<typename E>  arrStack<E>::~arrStack() {}  template<typename E>  void arrStack<E>::push(const E& e)  {  if (full())  {  cout << "스택 포화 상태로 푸쉬 불가" << endl;  return;  }  stack[++top] = e;  }  template<typename E>  void arrStack<E>::pop()  {  if (empty())  {  cout << "스택이 비어 있음 " << endl;  return;  }  --top;  }  template<typename E>  E arrStack<E>::peek() const  {  if (empty()) return 0;  return stack[top];  }  template<typename E>  int arrStack<E>::size() const  {  if (empty()) return 0;  return (top + 1);  }  template<typename E>  bool arrStack<E>::empty() const  {  if (top == -1) return true;  else return false;  }  template<typename E>  bool arrStack<E>::full() const  {  if (top + 1 == stackMAXSIZE) return true;  else return false;  }  template<typename E>  void arrStack<E>::printStack() const  {  for (int i = 0; i <= top; i++) cout << stack[i] << " ";  }  int main()  {  int menu;  arrStack<int> st; //스택 클래스 선언  cout << "\n### 스택 구현: 1차원 배열 ###\n" << endl;  cout << "1) 데이터 삽입: PUSH" << endl;  cout << "2) 데이터 삭제: POP" << endl;  cout << "3) 전체 출력" << endl;  cout << "4) 프로그램 종료" << endl << endl;  while (true)  {  cout << "메뉴 선택 : "; cin >> menu;  switch (menu) //선택한 번호에 따라 기능 실행  {  case 1: //1) 데이터 삽입  int input;  cout << "삽입할 데이터 입력: "; cin >> input;  st.push(input); break;  case 2: //2) 데이터 삭제  st.pop(); break;  case 3: //3) 전체 출력  st.printStack(); cout << endl;  break;  case 4: //4) 프로그램 종료  return 0;  }  cout << endl;  }  return 0;  } |

1. 실행 결과

텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

1. 연구 조사

위의 코드의 CircularQueue 클래스는 1차원 배열을 사용하여 원형 큐를 구현하고, 큐의 기본 동작을 수행하는 멤버 함수이다.

front, rear, count 멤버 변수를 이용하여 큐의 상태를 관리하며, 이를 이용하여 삽입, 삭제, 출력 등의 연산을 한다.

또한 printQueue 함수는 큐의 현재 상태를 출력하며, size, empty, full 함수들은 각각 큐의 크기, 포화 및 공백 여부를 반환한다.

스택을 배열로 구현할 때 가장 중요한 것은 변수 top이다.

top이라는 변수는 스택 배열의 가장 큰 숫자를 가진 인덱스를 의미하는 변수로, 스택으로 치면 가장 위쪽의 원소를 가리키는 인덱스 변수라고 볼 수 있다.

스택의 삽입 시 ‘st[++top]=e’과 같이 전위 증가 연산자를 사용하여 top 변수를 증가시킨 이유는, 언제나 가장 나중에 추가된 원소를 가리키는 인덱스인 top 변수를 이용해 추가로 원소를 삽입할 때, top의 위치도 같이 바꾼 상태에서, 업데이트 된 top의 위치에 해당하는 자리에 새 원소를 추가해야 하기 때문이다.

1. 원형 큐 구현: 순차 자료 구조
2. 소스 코드

|  |
| --- |
| #include <iostream>  #define queueMAXSIZE 10  using namespace std;  template <typename E>  class CircularQueue  {  E queue[queueMAXSIZE];  int front, rear, count;  public:  CircularQueue();//생성자  ~CircularQueue();//소멸자  void push(const E& e);//삽입  E pop();//삭제  E printfront() const;//프론트 원소 출력  E printback() const; //큐 제일 뒤에 있는 값 반환  int size() const;//큐의 크기 출력  bool empty() const; //큐 빈 여부 확인  bool full() const; //큐 포화 여부 확인  void printQueue() const; //큐 배열 원소 출력  };  template <typename E>  CircularQueue <E> ::CircularQueue() : front(0), rear(0), count(0) {}  template <typename E>  CircularQueue<E>::~CircularQueue() {} // 소멸자  template <typename E>  void CircularQueue<E>::push(const E& e)  {  if (full())  {  cout << "큐 포화 상태로 푸쉬 불가" << endl;  return;  }  queue[rear] = e;  rear = (rear + 1) % queueMAXSIZE;  ++count;  }  template<typename E>  E CircularQueue<E>::pop()  {  if (empty())  {  cout << "큐가 비어 있음 " << endl;  return 0;  }  E pop\_data = queue[front];  front = (front + 1) % queueMAXSIZE;  --count;  return pop\_data;  }  template<typename E>  E CircularQueue<E>::printfront() const  {  if (empty())  {  cout << "큐가 비어있습니다." << endl;  return 0;  }  return queue[front];  }  template<typename E>  E CircularQueue<E>::printback() const  {  if (empty())  {  cout << "큐가 비어있습니다." << endl;  return 0;  }  return queue[(rear - 1 + queueMAXSIZE) % queueMAXSIZE];  }  template<typename E>  int CircularQueue<E>::size() const  {  return count;  }  template<typename E>  bool CircularQueue<E>::empty() const  {  return (count == 0);  }  template<typename E>  bool CircularQueue<E>::full() const  {  return (count == queueMAXSIZE);  }  template<typename E>  void CircularQueue<E>::printQueue() const  {  if (empty())  {  cout << "큐가 비어 있습니다" << endl;  return;  }  if (full())  {  cout << "큐가 꽉 찼습니다" << endl;  return;  }  int i = front;  for (int j = 0; j < count; j++)  {  cout << queue[i] << " ";  i = (i + 1) % queueMAXSIZE;  }  cout << endl;  }  int main()  {  CircularQueue<int> q; //큐 클래스 선언  int menu;  cout << endl << "### 큐 구현: 1차원 배열 ###" << endl << endl;  cout << "1) 데이터 삽입(enQueue):" << endl;  cout << "2) 데이터 삭제(deQueue):" << endl;  cout << "3) 전체 출력" << endl;  cout << "4) 프로그램 종료" << endl << endl;  while (true)  {  cout << "메뉴 선택: "; cin >> menu;  switch (menu)  {  case 1: //1) 데이터 삽입  int input;  cout << "삽입할 데이터 입력: "; cin >> input;  q.push(input);  break;  case 2: //2) 데이터 삭제  q.pop();  break;  case 3: //3) 데이터 출력  q.printQueue();  break;  case 4: //4) 프로그램 종료  return 0;  }  cout << endl;  }  return 0;  } |

1. 실행 결과

텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

1. 연구 조사

* count 변수를 사용하는 이유

원형 큐에 empty한 큐와 full한 큐를 판정할 때 front=rear+1이라는 조건문을 사용하여 판별하는데, 이러면 아래의 그림과 같이 empty와 full 두 상태의 구분이 어려우므로 count 함수를 이용해 empty와 full한 큐를 분별해야 한다.

도표, 원, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

1. 이진 트리의 순회: 비재귀적 용법
2. 소스 코드

|  |
| --- |
| #include <iostream>  #include <string>  #include <stack>  #include <queue>  using namespace std;  template <typename E>  class LinkedBTree;  // 클래스 DNode  template <typename E>  class DNode {  public:  DNode(const E& data);  private:  E \_\_data;  DNode<E>\* \_\_Llink;  DNode<E>\* \_\_Rlink;  template <typename T> friend class LinkedBTree;  };  template <typename E>  DNode<E>::DNode(const E& data) :  \_\_data(data), \_\_Llink(nullptr), \_\_Rlink(nullptr) {}  //클래스 LinkedBTree  template <typename E>  class LinkedBTree {  public:  LinkedBTree();  ~LinkedBTree();  DNode<E>\* makeLinkedBTree(const string str);  void Preorder(DNode<E>\* root) const;  void Inorder(DNode<E>\* root) const;  void Postorder(DNode<E>\* root) const;  void Levelorder(DNode<E>\* root) const;  private:  DNode<E>\* \_\_root;  };  //생성자  template <typename E>  LinkedBTree<E>::LinkedBTree() : \_\_root(nullptr)  { }  //소멸자  template <typename E>  LinkedBTree<E>::~LinkedBTree()  {  if (\_\_root == nullptr)  return;  DNode<E>\* tNode = nullptr;  queue<DNode<E>\*> Q;  Q.push(\_\_root);  while (!Q.empty()) {  tNode = Q.front();  Q.pop();  if (tNode->\_\_Llink) Q.push(tNode->\_\_Llink);  if (tNode->\_\_Rlink) Q.push(tNode->\_\_Rlink);  delete tNode;  }  }  //공백 여부 판단  int isOperator(int op)  {  return op == '+' || op == '-' || op == '\*' || op == '/';  }  //이진트리 생성  template <typename E>  DNode<E>\* LinkedBTree<E>::makeLinkedBTree(const string str)  {  stack<DNode<E>\*> S;  DNode<E>\* tNode;  for (int i = 0; i < str.length(); i++)  {  while (str[i] == ' ')  i++;  tNode = new DNode<E>(str[i]);  if (isOperator(str[i])) {  tNode->\_\_Rlink = S.top(); S.pop();  tNode->\_\_Llink = S.top(); S.pop();  }  S.push(tNode);  }  tNode = S.top();  S.pop();  // 루트 노드 설정  \_\_root = tNode;  return tNode;  }  // 깊이 우선 순회: 전위 순회  template <typename E>  void LinkedBTree<E>::Preorder(DNode<E>\* root) const  {  if (root == nullptr) return;  stack<DNode<E>\*> S;  S.push(root);  while (!S.empty())  {  DNode<E>\* tNode = S.top();  S.pop();  cout.width(3); cout << tNode->\_\_data;  if (tNode->\_\_Rlink) S.push(tNode->\_\_Rlink);  if (tNode->\_\_Llink) S.push(tNode->\_\_Llink);  }  cout << endl;  }  // 깊이 우선 순회: 중위 순회  template <typename E>  void LinkedBTree<E>::Inorder(DNode<E>\* root) const  {  if (root == nullptr) return;  stack<DNode<E>\*> S;  DNode<E>\* ptr = root;  while (ptr != nullptr || !S.empty())  {  while (ptr != nullptr)  {  S.push(ptr);  ptr = ptr->\_\_Llink;  }  ptr = S.top();  S.pop();  cout.width(3); cout << ptr->\_\_data;  ptr = ptr->\_\_Rlink;  }  cout << endl;  }  // 깊이 우선 순회: 후위 순회  template <typename E>  void LinkedBTree<E>::Postorder(DNode<E>\* root) const  {  if (root == nullptr) return;  stack<DNode<E>\*> S1, S2;  S1.push(root);  while (!S1.empty())  {  DNode<E>\* tNode = S1.top();  S1.pop();  S2.push(tNode);  if (tNode->\_\_Llink) S1.push(tNode->\_\_Llink);  if (tNode->\_\_Rlink) S1.push(tNode->\_\_Rlink);  }  while (!S2.empty())  {  DNode<E>\* tNode = S2.top();  S2.pop();  cout.width(3); cout << tNode->\_\_data;  }  cout << endl;  }  // 너비 우선 순회: 비재귀적 용법(큐 구조 활용)  template <typename E>  void LinkedBTree<E>::Levelorder(DNode<E>\* root) const  {  if (root == nullptr)  return;  queue<DNode<E>\*> Q;  Q.push(root);  while (!Q.empty()) {  DNode<E>\* tNode = Q.front();  Q.pop();  cout.width(3);  cout << tNode->\_\_data;  if (tNode->\_\_Llink) Q.push(tNode->\_\_Llink);  if (tNode->\_\_Rlink) Q.push(tNode->\_\_Rlink);  }  cout << endl;  }  int main(void)  {  string postfix;  cout << "트리를 구성할 후위 수식: ";  getline(cin, postfix);  // 이진 트리 생성 및 구성  LinkedBTree<char> BT;  DNode<char>\* root = BT.makeLinkedBTree(postfix);  // 깊이 우선 순회: 전위.중위.후위 순회  cout << "\nPreorder : "; BT.Preorder(root); // 전위 순회  cout << "Inorder : "; BT.Inorder(root); // 중위 순회  cout << "Postorder : "; BT.Postorder(root); // 후위 순회  // 너비 우선 순회  cout << "Levelorder : "; BT.Levelorder(root);  return 0;  } |

1. 실행 결과

텍스트, 스크린샷, 소프트웨어, 번호이(가) 표시된 사진

자동 생성된 설명

1. 연구 조사

위의 코드에서 DNode 클래스로 템플릿으로 정의된 이진트리의 노드를 나타냈으며,

makeLinkedBTree 함수로 주어진 후위 수식을 이용하여 이진트리를 생성하는 동작을 수행한다. 이때 스택을 활용하여 연산자와 피연산자를 처리하며 이진트리를 구성한다.

문제에 주어진 조건대로 큐로 구현한 levelorder를 제외한 preorder, inorder, postorder는 스택 구조로 구현하여 입력받은 후위 수식을 표현하였다. 이때 이진 트리의 순회 방식에 대한 구현은 스택이나 큐를 이용하여 비재귀적으로 수행되어 있다.

이 코드의 주요 알고리즘에 대해 간략히 나타낸 그림은 다음과 같다:

|  |  |  |
| --- | --- | --- |
| 깊이 우선 순회 | | |
| 전위 순회 | 중위 순회 | 후위 순회 |
| 루트 노드를 먼저 방문하고, 왼쪽 서브트리를 전위 순회한 뒤에 오른쪽 서브트리를 전위 순회한다. | 왼쪽 서브트리를 중위 순회한 뒤에 루트 노드를 방문하고, 오른쪽 서브트리를 중위 순회한다. | 왼쪽 서브트리를 후위 순회한 뒤에 오른쪽 서브트리를 후위 순회하고, 마지막으로 루트 노드를 방문한다. |
| 너비 우선 순회 | | |
| 루트 노드부터 시작하여 각 레벨 순서대로 좌에서 우로 노드를 방문한다. 이진 트리를 너비 우선으로 탐색하는 방법으로, 노드의 순회 순서는 레벨 단위로 좌에서 우로 이동하면서 출력한다. | | |

1. 힙 정렬
2. 소스 코드

|  |
| --- |
| #include <iostream>  #include <cstdlib>  #include <ctime>  #define SIZE 10  using namespace std;  // 함수 선언  template <typename E>  void HeapSort(E\* pArr, const int num);  template <typename E>  void buildHeap(E\* pArr, const int num);  template <typename E>  void percolateDown(E\* pArr, const int root, const int num);  template <typename E>  void SWAP(E\* pa, E\* pb);  template <typename E>  void PRINT(E\* pArr, const int num);  template <typename E>  void SWAP(E\* pa, E\* pb)  {  E temp;  temp = \*pa;  \*pa = \*pb;  \*pb = temp;  return;  }  template <typename E>  void PRINT(E\* pArr, const int num)  {  for (int i = 0; i < num; i++)  {  cout.width(3); cout << \*(pArr + i);  }  cout << endl;  return;  }  template <typename E>  void buildHeap(E\* pArr, const int num)  {  for (int i = num / 2; i >= 0; i--) percolateDown(pArr, i, num-1);  cout << "buildHeap: "; PRINT(pArr, SIZE);  }  template <typename E>  void percolateDown(E\* pArr, const int root, const int num)  {  int child = 2 \* root + 1; // 왼쪽 자식  int right = 2 \* root + 2; // 오른쪽 자식  if (child <= num)  {  if (right <= num && pArr[child] < pArr[right]) child = right;    if (pArr[root] < pArr[child])  {  SWAP(pArr + root, pArr + child);  percolateDown(pArr, child, num);  }  }  }  template <typename E>  void HeapSort(E\* pArr, const int num)  {  buildHeap(pArr, num);  for (int i = num - 1; i >= 0; i--)  {  SWAP(pArr + 0, pArr + i);  percolateDown(pArr, 0, i - 1);  }  }  int main()  {  int arr[SIZE] = {0};  //초기 난수 생성해 배열에 넣기  srand((unsigned int)time(NULL));  for (int i = 0; i < SIZE; i++) \*(arr + i) = rand() % 100;  cout << "정렬 전: "; PRINT(arr, SIZE);  HeapSort(arr, SIZE);  cout << "정렬 후: "; PRINT(arr, SIZE);  return 0;  } |

1. 실행 결과

텍스트, 소프트웨어, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

1. 보충 설명

위 코드는 생성한 난수 여러 개를 정렬 하지 않은 상태에서 처음에 출력,

HeapSort 함수에서 buildHeap 함수를 호출하여 주어진 배열을 최대 힙으로 만든 상태에서 출력, 최대 힙이 구성된 상태에서 HeapSort 함수를 호출하여 힙 정렬을 수행하고 정렬된 최종적인 결과 배열을 출력하는 코드이다.

|  |
| --- |
| // 함수 선언  template <typename E>  void HeapSort(E\* pArr, const int num);  template <typename E>  void buildHeap(E\* pArr, const int num);  template <typename E>  void percolateDown(E\* pArr, const int root, const int num);  template <typename E>  void SWAP(E\* pa, E\* pb);  template <typename E>  void PRINT(E\* pArr, const int num); |

위와 같은 함수의 선언부를 넣어주어야 하는 이유는 함수의 구현부에서 서로 순서가 엉킨 채로 다른 함수를 호출하고 있기 때문이다.

* 힙 정렬 알고리즘

힙은 완전 이진트리 기반의 트리형 자료구조로써 최댓값이나 최솟값을 찾아내기 위해 사용된다. 힙 정렬은 최대 힙 트리나 최소 힙 트리를 구성해 정렬을 하는 방법이다.

힙 정렬을 위한 과정은 다음과 같다.

* n개의 노드에 대한 완전 이진트리를 구성하고, 루트 노드로부터 부모노드, 왼쪽 자식 노드, 오른쪽 자식 노드 순으로 구성한다.
* 최대 힙(부모노드>자식노드인 트리)을 구성한다. 단말노드를 자식 노드로 가진 부모노트부터 구성하며 아래부터 루트까지 올라오며 순차적으로 만들어갈 수 있다.
* 가장 큰 수(루트에 위치)를 가장 끝의 노드와 맞바꾼다.
* 위의 과정을 반복한다.

단, 힙 정렬의 시간 복잡도는 O(nlogn)이다.