자료구조및프로그래밍 HW7

C211171 최후락

2025 11 13

1 개요

Infix로 주어진 식을 Arraystack을 사용하여 Postfix로 변환한다.
post.in의 산술연산뿐 아니라 post2.in의 논리연산도 가능하게 구현한다.

1.Arraystack구현

2.post구현

3.실행결과

4.어려웠던 점

-코드에 대한 설명은 주석을 참고한다.

```cpp
template<typename T>
void ArrayStack<T>::push(const T& someItem)
{
    if (top >= (capacity - 1)) //배열이 꽉찼으면
    {
        int new_capacity = capacity + 3; //용량 3증가

        T* new_items = new T[new_capacity]; //새 배열 만들기

        for (int i = 0; i <= top; i++)
        {
            new_items[i] = items[i];
        }//전부복사

        delete[] items; //원래배열 없애기

        items = new_items;
        capacity = new_capacity;//용량증가
    }

    top++;
    items[top] = someItem; //아이템추가
}
```

Figure 1: push

2 Arraystack구현

2.1 push

Arraystack에서 push를 구현함에 있어 한가지 문제가 있었다. 배열이 고정된 크기 5로 제한되어 있어 만약 5를 넘는 상황이 오면 새로운 item을 push할 수 없었다. 이 문제를 해결하기 위해서 push를 할 때, 현재크기를 넘게 된다면 3만큼 더 큰 배열을 임시로 만든 뒤 기존의 배열을 복사하고 포인터를 옮기는 방식을 사용했다. 이를 위해서 int capacity; 를 private에 새로 추가했으며, 다음과 같이 push를 구현했다.

2.2 pop

스택이 비어있지 않은 경우에 한해서 top를 하나 줄여 구현했다.

```cpp
template <typename T>//빼기
bool ArrayStack<T>::pop() {
    bool result = false;
    if (is_empty())
    {
        throw PrecondViolatedExcep("Stack is empty!");
    }
    else
    {
        result = true;
        top--;
    }
    return result;

}
```

Figure 2: pop

2.3 peek

스택이 비어있지 않은 경우에 한해서 items[top]를 반환한다.

```
63  template <typename T>//맨위 보기
64  T ArrayStack<T>::peek() const
65  {
66      if (is_empty())
67      {
68          throw PrecondViolatedExcep("Stack is empty");
69      }//비었으면 오류
70
71      return items[top];
72  }
73
```

Figure 3: peek

3 post구현

post를 구현하기에 앞서, 기존에 작성되어 있는 코드를 이해하는데 상당히 많은 시간을 사용했다. 토큰의 개념과 이를 어떻게 만들어 놓았을지를 코드를 보며 하나씩 공부하는 느낌으로 분석했다. 처음보는 개념인 isp와 icp가 나와서 구현하기전 어떤 개념인지를 살펴보고 어떻게 구현할지 생각하는데도 많은 시간이 소요되었다.

3.1 GetInt

기존에 작성된 생성자를 보고 많이 참고해서 GetInt를 작성했다.

3.2 Twochar

작성요 작성요

3.3 isp

작성요

3.4 postfix

우리가 지금까지 짠 모든 코드는 이걸 위해서 존재했다.

```
8   //토큰의 생성자
9   Token::Token() {}
10
11  Token::Token(char c) : len(1), type(c) {
12      str = new char[1];
13      str[0] = c;
14  } // default type = c itself
15
16  Token::Token(char c, char c2, int ty) : len(2), type(ty) {
17      str = new char[2];
18      str[0] = c;
19      str[1] = c2;
20  }
21
22  Token::Token(char* arr, int l, int ty = ID) : len(1), type(ty) {
23      str = new char[len + 1];//deep copy준비
24      for (int i = 0; i < len; i++) str[i] = arr[i];//복사
25          str[len] = '\0';//마지막 널
26      if (type == NUM) {//숫자라면
27          ival = arr[0] - '0';//아스키0과 차이만큼이 진짜 숫자값이니까
28          for (int i = 1; i < len; i++) ival = ival * 10 + arr[i] - '0';//자릿수 맞춰주기 > 여기까지
29      } else if (type == ID)//식별자라면
30          ival = 0;//값이 없지
31      else
32      throw "must be ID or NUM";
33  }
```

Figure 4: token의 생성자

```
8   //토큰의 생성자
9   Token::Token() {}
10
11  Token::Token(char c) : len(1), type(c) {
12      str = new char[1];
13      str[0] = c;
14  } // default type = c itself
15
16  Token::Token(char c, char c2, int ty) : len(2), type(ty) {
17      str = new char[2];
18      str[0] = c;
19      str[1] = c2;
20  }
21
22  Token::Token(char* arr, int l, int ty = ID) : len(l), type(ty) {
23      str = new char[len + 1];//deep copy준비
24      for (int i = 0; i < len; i++) str[i] = arr[i];//복사
25          str[len] = '\0';//마지막 널
26      if (type == NUM) {//숫자라면
27          ival = arr[0] - '0';//아스키0과 차이만큼이 진짜 숫자값이니까
28          for (int i = 1; i < len; i++) ival = ival * 10 + arr[i] - '0';//자릿수 맞춰주기 > 여기까지
29      } else if (type == ID)//식별자라면
30          ival = 0;//값이 없지
31      else
32      throw "must be ID or NUM";
33  }
```

Figure 5: GetInt

```
97  bool TwoCharOp(Expression& e, Token& tok) {//두개짜리 연산자인지 확인한다.
98
99      if (e.pos + 1 >= e.len) {//문자열 끝인지 여부 확인
100         return false;
101     }
102
103     char c = e.str[e.pos];//지금위치
104     char c2 = e.str[e.pos + 1];//한 칸 다음
105     int op;
106
107     if (c == '<' && c2 == '=')//if써서 맞는거 찾아서 반환
108         op = LE;
109     else if (c == '>' && c2 == '=')
110         op = GE;
111     else if (c == '=' && c2 == '=')
112         op = EQ;
113     else if (c == '!' && c2 == '=')
114         op = NE;
115     else if (c == '&' && c2 == '&')
116         op = AND;
117     else if (c == '|' && c2 == '|')
118         op = OR;
119
120     else {
121         return false;
122     } // 맞는 두 글자 토큰이 아니면 false
123
124     tok = Token(c, c2, op);//묶어서 저장
125
126     e.pos += 2; //두칸이동
127
128     return true;
129 }
```

Figure 6: Twocharop

```
165  int icp(const Token& t) { // in-coming priority 우선순위 정하기
166      int ty = t.type;
167
168      switch (ty) {
169          case '(':
170              return 0;
171
172          case UMINUS:
173          case '!':
174              return 1;
175
176          case '*':
177          case '/':
178          case '%':
179              return 2;
180
181          case '+':
182          case '-':
183              return 3;
184
185          case '<':
186          case '>':
187          case LE:
188          case GE:
189              return 4;
190
191          case EQ:
192          case NE:
193              return 5;
194
195          case AND:
196              return 6;
197
198          case OR:
199              return 7;
200
201          case '=':
202              return 8;
203
204           case '#':
205              return 9;
206
207          default:
208              return -1;
209      }
210  }
```

Figure 7: icp

```
212 int isp(const Token& t)  // in-stack priority
213 {
214     int ty = t.type;
215
216     switch (ty) {
217
218         case '(':
219         case '#':
220         case '=':
221             return 9;
222
223         case UMINUS:
224         case '!':
225             return 2;
226
227         case '*':
228         case '/':
229         case '%':
230             return 2;
231
232         case '+':
233         case '-':
234             return 3;
235
236         case '<':
237         case '>':
238         case LE:
239         case GE:
240             return 4;
241
242         case EQ:
243         case NE:
244             return 5;
245
246         case AND:
247             return 6;
248
249         case OR:
250             return 7;
251
252         default:
253             return -1;
254     }
255 }
```

Figure 8: isp

```
212 int isp(const Token& t)  // in-stack priority
213 {
214     int ty = t.type;
215
216     switch (ty) {
217
218         case '(':
219         case '#':
220         case '=':
221             return 9;
222
223         case UMINUS:
224         case '!':
225             return 2;
226
227         case '*':
228         case '/':
229         case '%':
230             return 2;
231
232         case '+':
233         case '-':
234             return 3;
235
236         case '<':
237         case '>':
238         case LE:
239         case GE:
240             return 4;
241
242         case EQ:
243         case NE:
244             return 5;
245
246         case AND:
247             return 6;
248
249         case OR:
250             return 7;
251
252         default:
253             return -1;
254     }
255 }
```

Figure 9: postfix

```
C:\Users\hoora\Desktop\hw7>hw7.exe < post.in
C211171
A B * C *
A -u B + C - D +
A B -u * C +
A B + D * E F A D * + / + C +
```

Figure 10: post.in

```
C:\Users\hoora\Desktop\hw7>hw7.exe < post2.in
C211171
A B && C || E F > ! ||
A B C < C D > || ! && ! C E < ||
```

Figure 11: post2.in

4 실행결과

실행결과 학번과 함께 post.in, post2.in, post3.in, post4.in 모두 이상없이 출력된다.

```
C:\Users\hoora\Desktop\hw7>hw7.exe < post3.in
C211171
34 56 * 11 2 / +
1 2 3 * + 4 -u 2 * +
```

Figure 12: post3.in

```
C:\Users\hoora\Desktop\hw7>hw7.exe < post4.in
C211171
33 55 2 * +
an77 2 7 5 * + =
b 2 =
an77 b 2 * +
a 5 +
```

Figure 13: post4.in

5 어려웠던 점

새로운 코드를 구현하는 것은 늘 어려운 일이고 어떤 방식으로 구현할지 고민하는데 대부분의 시간을 사용한다. 하지만, 이번 과제 hw7에서는 기존에 작성되어 있는 코드를 분석하고 내가 어떤 것을 작성해야 하는지 파악하는데에 전체 시간의 상당부분을 할애했다. 스택을 구현함에 있어서는 기존에 어디선가 봤던 방식들이 떠올라서 비교적 짧은 시간내에 코드를 작성했지만, post는 그렇지 않았다.

강의록에서 infix를 postfix로 stack과 queue를 사용해 변환하는 코드를 봤던 기억을 토대로 작성해보려 했다. 이를 위해서 처음보는 개념인 Expression과 Token을 여기저기 뒤져가면서 공부했다. 처음보는 개념이라 상당한 시간이 소요되었다.

isp와 icp 또한 만만치 않았다. 우리가 기존에 사용하던 연산자간의 우선순위인 in stack priority, isp는 스위치문을 사용하여 케이스 분류를 하면 손쉽게 할 수 있었다. 하지만 isp와 비교할 또 다른 우선순위인 icp는 쉽지 않았다. 연산자 간에 우선순위도 비교를 하며 기존에 스택에 들어가 있는 isp와도 동시에 적절한 우선순위를 가져야했다. 여기서 처음보는 #이 나왔지만 이 연산자는 일종의 보호막이라고 생각하여 맨 뒤로 우선순위를 미뤄놓고 처리하니 어렵지 않게 9번으로 정할 수 있었다.

물론 코드를 작성하는데에도 어려움이 많았고 makefile을 통해서 실행할 때도 말썽이었다. 지난번 컴파일러가 잘 물리지 않아서 실행이 안되는 것은
****
** Visual Studio 2022 Developer Command Prompt v17.13.2
** Copyright (c) 2022 Microsoft Corporation
****
[vcvarsall.bat] Environment initialized for: 'x64'

C:\Program Files\Microsoft Visual Studio\2022\Community¿code .
을 사용하여 새로 띄워진 vscode창을 통해서 해결했다. 이번에는 이 새로 킨 vscode에서 make명령어를 인식하지 못했다. 여러가지 방법이 실패했고 최종적으로 make.exe의 실제 위치를 찾아서 환경변수에 새로 물려주는 방식으로 해결했다. 차라리 코드구현에서 막혔다면 공부한다는 생각이라도 했겠지만 make가 실행되지 않자 정말 다해놓고 막힌 기분이었다.