자료구조 HW1

B935394 컴퓨터공학과 장준희

September 19, 2020

1 개념 설명

1.1 클래스와 객체

클래스는 사용자 정의 타입으로 멤버(멤버 변수, 멤버 함수 등...)의 집합으로 구성된다. 생성, 복사, 소멸 의미를 정의할 수 있으며 객체에 대해서는 . 포인터에 대해서는 ->로 멤버에 접근할 수 있다. 연산자 오버로딩을 통해 연산자는 클래스에 대해 정의될 수 있다. 대게 public 멤버는 클래스의 인터페이스를 제공하고, private 멤버는 구현 세부사항을 제공한다. **객체**는 어떤 타입의 값을 보관하는 약간의 메모리이다.

1.2 연산자 오버로딩

연산자 오버로딩은 피연산자에 따라 다른 연산을 하도록 동일한 연산자를 중복해서 작성하는 것이다.
returnType operator operatorSymbol(list of variable) 의 형태로 선언한다.

2 코드 설명

코드를 첨부하고 설명은 주석형태로 달기보다는 뒤에 따로 마련해두었습니다.

2.1 코드

mystr.cpp:

```
#include <iostream>
#include <cstring>
#include "mystr.h"
using namespace std;

Mystring::Mystring(const char* str = "default") {
    len = strlen(str);
    string = new char[len + 1];
    strcpy(string, str);
}

bool Mystring::operator ==(const Mystring& str) {
    if (strcmp(this->string , str.string) == 0) return true;
    else return false;
}


Mystring& Mystring::operator =(const Mystring& str) {
    if (this->string != NULL)
        delete[] this->string;
    this->len = str.len;
    this->string = new char[this->len + 1];
    strcpy(this->string, str.string);
    return *this;
}
```

```
Mystring Mystring ::operator *(const Mystring& str) {
    int i,j;
    int length = this->len + str.len + 1;
    char *tempchar = new char[length];

    if (this->len >= str.len) {
        for (i = 0; i < str.len; ++i) {
            tempchar[2*i] = this->string[i];
            tempchar[2*i + 1] = str.string[i];
            j = 2*(i+1);
        }
        for (; i < this->len; ++i) {
            tempchar[j++] = this->string[i];
        }
        Mystring tempstring = tempchar;
        delete[] tempchar;
        *this = tempstring;
        return *this;
    }
    else {
        for (i = 0; i < this->len; ++i) {
            tempchar[2 * i] = this->string[i];
            tempchar[2 * i + 1] = str.string[i];
            j = 2 * (i + 1);
        }
        for (; i < str.len; ++i) {
            tempchar[j++] =str.string[i];
        }
        Mystring tempstring = tempchar;
        delete[] tempchar;
        *this = tempstring;

        return *this;
    }
}

Mystring Mystring::operator +(const Mystring &str) {
    int length = this->len + str.len + 1;
    char* tempchar = new char[length];
    strcpy(tempchar, this->string);
    strcat(tempchar, str.string);
    Mystring tempstring(tempchar);
    delete[] tempchar;
    return tempstring;
}

ostream& operator <<(ostream& out, const Mystring& str) {
    out << str.string;
```

```
    return out;
}
Mystring::~Mystring() { delete[] string; }
```

2.2 설명

- **==연산자**
  cstring의 strcmp을 이용하여 두 문자열을 비교해주었다. 0이면 참을 그 외의 값에는 거짓을 반환한다.

- **=연산자**
  대입을 해준다. 다만 이때 원래의 객체가 NULL이 아니라면, 그 값을 반환하고 새로이 동적할당하여 주어진 값(문자열의 길이, 문자열) 을 대입하도록 했다. 생성자에서 많은 참조를 하였다.

- ***연산자**
  어찌되었든 두 문자열의 문자를 다 가져가기 때문에 임시로 그 둘의 길이를 합친 크기의 메모리 임시 문자열 변수 tempchar에 동적할당해주었다.
  if문은 피연산자의 두 문자열의 길이로 비교를 하였다. 만약에 앞의 피연산자가 더 길다면 if블록의 내용을, 뒤의 피연산자가 더 길다면 else블록의 내용을 따라가도록 하였다. 길이의 차이를 제외하면 내용은 비슷한데, 짧은 문자열의 끝에 도달할 때까지 for문을 돌려, 번갈아가며 문자를 대입시켰다. 그리고 남은 문자열을 다시 for문을 통해 뒤에 대입했다. 이 때 i가 앞의 for문에서의 값을 가지고 있어야했기 때문에 for문 밖에 선언하였다. tempchar를 이용하여 임시 Mystring 객체 tempstring을 생성하였고, return하였다.

- **+연산자**
  연산자에서와 마찬가지로, 길이를 합해 그 크기만큼을 동적할당 시키고, strcpy로 앞의 피연산자를 대입한 후 뒤에 strcat을 이용해 뒤의 피연산자를 붙였다. tempchar를 이용해 tempstring을 생성하고 tempchar는 반환, tempstring을 return 하였다.

- **<<연산자**
  out에 Mystring객체의 문자열을 주고, 그 값을 반환하였다.