자료구조실습

배열과 문자열

목차

KWANGWOON UNIVERSITY

KWANGWOON UNIVERSITY

배열과 문자열

배열

KWANGWOON UNIVERSITY

- 배열 (Array)
  - 인덱스와 그 인덱스에 대응하는 같은 종류의 데이터들로 이루어진 자료구조.
  - 가장 기초적인 자료구조로, 대부분의 프로그래밍 언어에서 사용할 수 있음.
    - 고급 프로그래밍 언어들은 다차원 배열을 지원함.
  - 배열은 크게 정적 배열과 동적 배열이 있음.

- 배열의 원소 (Element)
  - 일반적으로 배열에는 데이터들이 순차적으로 저장됨.
    - 데이터의 번호(인덱스)가 배열의 시작점부터 데이터가 저장되어 있는 상대적인 위치임.
  - 배열의 이름과 인덱스를 이용하여, 특정 위치의 원소에 접근할 수 있음.

arr[0]	1
arr[1]	1
arr[2]	2
arr[3]	3
arr[4]	5
arr[5]	8
arr[6]	13
arr[7]	21
arr[8]	34
arr[9]	55
arr[10]	89
arr[11]	144

정적 배열

KWANGWOON UNIVERSITY

□ 정적 배열 (Static Array)
- 크기가 고정되어 있는 배열.
- 배열에 저장할 수 있는 데이터의 개수가 고정되어 있음.

□ 장점
- 한번 할당하면, 메모리를 추가적으로 할당하거나 해제하지 않아도 됨.
- 특정 위치의 데이터에 접근하는 속도가 빠름.

□ 단점
- 크기가 고정되어 있기 때문에, 필요에 따라 크기를 조절할 수 없음.
- 일반적으로 스택(Stack) 영역에 할당되기 때문에, 최대 크기가 제약됨.

동적 배열

KWANGWOON UNIVERSITY

□ 동적 배열 (Dynamic Array)
- ■ 크기가 고정되지 않은 배열.

□ 장점
- ■ 필요에 따라 배열의 크기를 조절할 수 있음.

□ 단점
- ■ 필요에 따라 메모리를 추가적으로 할당하기 때문에, 부가적인 연산이 추가됨.

문자열

KWANGWOON UNIVERSITY

- 문자열 (String)
  - 문자의 순차 수열을 나타내는 자료구조.
  - 일반적으로, 문자 인코딩과 관련된 문자를 대표하는 일련의 자료값을 저장함.

- Binary String
  - 텍스트로 표시되지 않아도 되는 문자열.

- Null String
  - 아무것도 저장되어 있지 않은 문자열.

- 문자열 리터럴 (String literal)
  - 접두사가 없는 큰따옴표로 구분하는 문자열.
  - 예시) "Hello World!"

KWANGWOON UNIVERSITY

C++ 의 배열

array 컨테이너

KWANGWOON UNIVERSITY

- 헤더 파일 및 템플릿

```
#include <array>

template <typename T, std::size_t N> class array;
```

- 설명
  - C++ STL에 포함되어 있는 정적 배열을 표현하는 컨테이너.
  - C++ 11에 추가됨.
- 인자
  - T : 데이터의 자료형
  - N : 데이터의 개수
- 선언 및 초기화 예시
  - std::array<int, 5> arr; : 1차원 정수형 배열 선언
  - std::array<int, 5> arr = {1, 2, 3, 4, 5}; : 각 원소 초기화
  - std::array<int, 5> arr = {0}; : 모든 배열의 원소를 0으로 초기화
  - std::array<int, 5> arr = {1, 2}; : 지정한 원소 이외의 모든 원소를 0으로 초기화
  - std::array<std::array<int, 5>, 5> arr = {0}; : 2차원 정수형 배열 선언

array 컨테이너 (cont`d)

KWANGWOON UNIVERSITY

- 멤버 함수 (Iterators)
  - iterator begin() noexcept;
    - 배열의 첫번째 원소를 가리키는 반복자를 반환함.
  - iterator end() noexcept;
    - 배열의 마지막 원소의 다음을 가리키는 반복자를 반환함.
  - reverse_iterator rbegin() noexcept;
    - 배열을 역으로 했을 때, 그 첫번째 원소를 가리키는 역방향 반복자를 반환함.
  - reverse_iterator rend() noexcept;
    - 배열을 역으로 했을 때, 그 마지막 원소의 다음을 가리키는 역방향 반복자를 반환함.

array 컨테이너 (cont`d)

KWANGWOON UNIVERSITY

- 멤버 함수 (Capacity)
  - constexpr size_type size() noexcept;
    - 배열의 크기를 반환함.
  - constexpr bool empty() noexcept;
    - 배열이 비어있음의 여부를 반환함.

array 컨테이너 (cont`d)

KWANGWOON UNIVERSITY

- 멤버 함수 (Element access)
  - reference operator[](size_type n);
    - 배열의 n번째 원소를 반환함.
  - reference operator at(size_type n);
    - 배열의 n번째 원소를 반환함.
  - reference front();
    - 배열의 첫번째 원소를 반환함.
  - reference back();
    - 배열의 마지막 원소를 반환함.
  - value_type* data() noexcept;
    - 배열을 포인터 타입으로 반환함.

array 컨테이너 (cont`d)

KWANGWOON UNIVERSITY

- 멤버 함수 (Modifiers)
  - void fill(const value_type& val);
    - 배열의 모든 원소를 val로 변경함.
  - void swap(array& arr) noexcept;
    - 배열의 모든 원소를 배열 arr과 교환함.

array 컨테이너 예시 - 1차원 배열

KWANGWOON UNIVERSITY

```cpp
#include <array>
#include <iostream>

using namespace std;

int main() {
    array<int, 8> arr1;
    array<int, 8> arr2 = {0};
    array<int, 8> arr3 = {1, 2, 3, 4};
    array<int, 8> arr4 = {1, 2, 3, 4, 5, 6, 7, 8};

    cout << "elements of arr1: ";
    array<int, 8>::iterator iter;
    for (iter = arr1.begin(); iter != arr1.end(); ++iter) {
        cout << *iter << " ";
    }
    cout << endl;

    cout << "elements of arr2: ";
    for (size_t i = 0; i < arr2.size(); ++i) { cout << arr2[i] << " "; }
    cout << endl;

    cout << "elements of arr3: ";
    for (size_t i = 0; i < arr3.size(); ++i) {
        cout << arr3.at(i) << " ";
    }
    cout << endl;

    cout << "elements of arr4 (reverse): ";
    array<int, 8>::reverse_iterator riter;
    for (riter = arr4.rbegin(); riter != arr4.rend(); ++riter) {
        cout << *riter << " ";
    }
    cout << endl;

    return 0;
}
```

array 컨테이너 예시 - 1차원 배열 (cont`d)

KWANGWOON UNIVERSITY

□ 컴파일 결과

```
elements of arr1: -2115200080 32765 4199666 0 2 0 4200381 0
elements of arr2: 0 0 0 0 0 0 0 0
elements of arr3: 1 2 3 4 0 0 0 0
elements of arr4 (reverse): 8 7 6 5 4 3 2 1
```

array 컨테이너 예시 - 2차원 배열

KWANGWOON UNIVERSITY

```cpp
#include <array>
#include <iostream>

using namespace std;

int main() {
    array<array<int, 8>, 4> arr = {0};
    array<array<int, 8>, 4>::iterator row;
    array<int, 8>::iterator col;

    cout << "elements of arr" << endl;
    for (row = arr.begin(); row != arr.end(); ++row) {
        for (col = (*row).begin(); col != (*row).end(); ++col) {
            cout << *col << " ";
        }
        cout << endl;
    }
    cout << endl;

    int i = 1;
    for (row = arr.begin(); row != arr.end(); ++row) {
        (*row).fill(i++);
    }

    cout << "value of arr[0][2]: ";
    cout << (arr.front())[2] << endl;

    cout << "value of arr[1][3]: ";
    cout << (arr.at(1)).at(3) << endl;

    cout << "value of arr[2][2]: ";
    cout << arr[2][2] << endl;

    return 0;
}
```

array 컨테이너 예시 - 2차원 배열 (cont`d)

KWANGWOON UNIVERSITY

□ 컴파일 결과

```
elements of arr
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

elements of arr (after fill)
1 1 1 1 1 1 1 1
2 2 2 2 2 2 2 2
3 3 3 3 3 3 3 3
4 4 4 4 4 4 4 4

value of arr[0][2]: 1
value of arr[1][3]: 2
value of arr[2][2]: 3
```

Array 구현

```cpp
#include <iostream>
#include <iterator>
#include <initializer_list>
#include <stdexcept>

template <std::size_t SIZE>
class Array {
private:
    int data[SIZE];
    using iterator = int*;

public:
    Array();
    Array(std::initializer_list<int> init);

    int& operator[](std::size_t index);

    std::size_t size() const;

    iterator begin();
    iterator end();
};

template <std::size_t SIZE>
Array<SIZE>::Array() {
    for (std::size_t i = 0; i < SIZE; ++i) {
        data[i] = int();
    }
}
```

```cpp
template <std::size_t SIZE>
Array<SIZE>::Array(std::initializer_list<int> init) {
    if (init.size() > SIZE) {
        throw std::out_of_range("Initializer list exceeds array size");
    }
    std::size_t i = 0;
    for (const auto& elem : init) {
        data[i++] = elem;
    }
    for (; i < SIZE; ++i) {
        data[i] = int();
    }
}

template <std::size_t SIZE>
int& Array<SIZE>::operator[](std::size_t index) {
    if (index ≥ SIZE) {
        throw std::out_of_range("Index out of range");
    }
    return data[index];
}

template <std::size_t SIZE>
std::size_t Array<SIZE>::size() const {
    return SIZE;
}

template <std::size_t SIZE>
typename Array<SIZE>::iterator Array<SIZE>::begin() {
    return data;
}

template <std::size_t SIZE>
typename Array<SIZE>::iterator Array<SIZE>::end() {
    return data + SIZE;
}
```

```cpp
int main() {
    Array<8> arr1;
    Array<8> arr2 = { 0 };
    Array<8> arr3 = { 1,2,3,4 };
    Array<8> arr4 = { 1, 2, 3, 4, 5, 6, 7, 8 };

    std::cout << "elements of arr1: ";
    for (auto it = arr1.begin(); it ≠ arr1.end(); ++it) {
        std::cout << *it << " ";
    }
    std::cout << std::endl;

    std::cout << "elements of arr2: ";
    for (auto it = arr2.begin(); it ≠ arr2.end(); ++it) {
        std::cout << *it << " ";
    }
    std::cout << std::endl;

    std::cout << "elements of arr3: ";
    for (auto it = arr3.begin(); it ≠ arr3.end(); ++it) {
        std::cout << *it << " ";
    }
    std::cout << std::endl;

    std::cout << "elements of arr4: ";
    for (auto it = arr4.begin(); it ≠ arr4.end(); ++it) {
        std::cout << *it << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

Array 구현

KWANGWOON UNIVERSITY

□ 컴파일 결과

```
elements of arr1: 0 0 0 0 0 0 0 0
elements of arr2: 0 0 0 0 0 0 0 0
elements of arr3: 1 2 3 4 0 0 0 0
elements of arr4: 1 2 3 4 5 6 7 8
```

KWANGWOON UNIVERSITY

C++ 의 문자열

string 컨테이너

KWANGWOON UNIVERSITY

□ 헤더 파일 및 템플릿

```
#include <string>

typedef std::basic_string<char> string;
```

□ 설명

■ C++ STL에 포함되어 있는 문자열을 표현하는 컨테이너.

□ 인자

■ 없음.

□ 선언 및 초기화 예시

■ std::string str; : 문자열 선언

■ std::string str = **"Hello World"**; : 문자열 초기화

string 컨테이너 (cont`d)

KWANGWOON UNIVERSITY

- 멤버 함수 (Iterators)
  - iterator begin() noexcept;
    - 문자열의 첫번째 문자를 가리키는 반복자를 반환함.
  - iterator end() noexcept;
    - 문자열의 마지막 문자의 다음을 가리키는 반복자를 반환함.
  - reverse_iterator rbegin() noexcept;
    - 문자열을 역으로 했을 때, 그 첫번째 문자를 가리키는 역방향 반복자를 반환함.
  - reverse_iterator rend() noexcept;
    - 문자열을 역으로 했을 때, 그 마지막 문자의 다음을 가리키는 역방향 반복자를 반환함.

string 컨테이너 (cont`d)

KWANGWOON UNIVERSITY

- 멤버 함수 (Capacity)
  - void resize(size_type n);
    - 문자열의 크기를 n으로 변경하고, 모든 원소를 '₩0'으로 초기화 함.
      - 크기가 증가하는 경우에, 추가된 원소를 '₩0'으로 초기화 함.
      - 크기가 감소하는 경우에, 기존의 원소는 유지함.
  - void shrink_to_fit();
    - 배열의 용량을 크기에 맞춰 감소함.
  - size_type size() const noexcept;
    - 문자열의 크기를 반환함.
  - size_type length() const noexcept;
    - 문자열의 길이를 반환함.
  - size_type capacity() const noexcept;
    - 배열의 용량을 반환함.
  - void clear() noexcept;
    - 문자열의 모든 문자를 삭제함. (길이가 0이 됨.)
  - bool empty() noexcept const;
    - 배열이 비어있음의 여부를 반환함.

string 컨테이너 (cont`d)

KWANGWOON UNIVERSITY

- 멤버 함수 (Element access)
  - char& operator[](size_type n);
    - 문자열의 n번째 문자를 반환함.
  - char& at(size_type n);
    - 문자열의 n번째 문자를 반환함.
  - char& front();
    - 문자열의 첫번째 문자를 반환함.
  - char& back();
    - 문자열의 마지막 문자를 반환함.

string 컨테이너 (cont`d)

KWANGWOON UNIVERSITY

- 멤버 함수 (Modifiers)
  - string& assign(const string& str);
    - 문자열을 str로 대체하고, 그 문자열을 반환함.
  - string& operator+=(const string& str);
    - 문자열의 뒤에 str을 추가하고, 그 문자열을 반환함.
  - string& append(const string& str);
    - 문자열의 뒤에 str을 추가하고, 그 문자열을 반환함.
  - void swap(string& str) noexcept;
    - 문자열을 문자열 str과 교환함.
  - void push_back(char c);
    - 문자열의 마지막에 문자 c를 추가함.
  - void pop_back();
    - 문자열의 마지막 원소를 삭제함.
  - string& replace(std::size_t pos, std::size_t len, const string& str);
    - 배열의 지정한 위치(pos)부터 len만큼의 문자를 str로 대체하고, 그 문자열을 반환함.

string 컨테이너 (cont`d)

KWANGWOON UNIVERSITY

- 멤버 함수 (Modifiers) (cont`d)
  - string& insert(std::size_t pos, const string& str);
    - 문자열의 지정한 위치(pos)에 str을 추가하고, 그 문자열을 반환함.
  - string& erase(std::size_t pos, std::size_t len);
    - 문자열의 지정한 위치(pos)에서 len만큼 문자를 삭제하고, 그 문자열을 반환함.

string 컨테이너 (cont`d)

KWANGWOON UNIVERSITY

- 멤버 함수 (Operations) (cont`d)
  - const char* c_str() noexcept;
    - 문자열을 C 스타일의 char형 배열로 변환하여 반환함.
  - const char* data() const noexcept;
    - 문자열을 char형 배열로 반환함.
  - std::size_t copy(char* str, std::size_t len, std::size_t index = 0) const;
    - 문자열에 지정한 위치(index)에서 len만큼 문자를 C 스타일의 char형 배열을 복사하고, 복사된 크기를 반환함.
  - std::size_t find(const string& str, std::size_t pos) const;
    - 문자열에서 지정한 위치(index) 이후에 문자열 str과 일치하는 문자열의 시작 위치를 반환함.
      - 일치하는 문자열이 없는 경우에, string::npos 를 반환함.
  - int compare(const string& str) const;
    - 문자열과 문자열 str을 비교하고, 그 결과를 반환함.
      - 일치하는 경우에, 0을 반환함.
      - 일치하지 않은 경우에, 0이 아닌 값을 반환함.
  - string substr(std::size_t pos, std::size_t len = pos);
    - 문자열의 지정한 위치(pos)에서 len만큼의 일부 문자열을 반환함.

string 컨테이너 예시

KWANGWOON UNIVERSITY

```cpp
#include <iostream>
#include <string>
#include <vector>

using namespace std;

int main() {
    string str1 = "Hello World!";
    cout << "str1: " << str1 << endl;

    string::reverse_iterator rit;
    cout << "str1 (reverse): ";
    for (rit = str1.rbegin(); rit != str1.rend(); ++rit) { cout << *rit; }
    cout << endl << endl;

    string str2 = "Bonjour!";
    cout << "str2: " << str2 << endl << endl;

    str1.swap(str2);

    cout << "--- After swap ---" << endl;
    cout << "str1: " << str1 << endl;
    cout << "str2: " << str2 << endl;

    return 0;
}
```

string 컨테이너 예시 (cont`d)

KWANGWOON UNIVERSITY

□ 컴파일 결과

```
str1: Hello World!
str1 (reverse): !dlroW olleH

str2: Bonjour!

--- After swap ---
str1: Bonjour!
str2: Hello World!
```

String 구현

```
#include <iostream>
#include <stdexcept>

class String {
private:
    char* data;
    size_t size;

public:
    String();
    String(const char* str);
    ~String();

    String& operator=(const String& other);

    size_t length() const;

    void append(const String& other);
    size_t copy(char* str, size_t len, size_t index = 0) const;

    char& operator[](size_t index);
    const char& operator[](size_t index) const;

    void swap(String& other) noexcept;

    friend std::ostream& operator<<(std::ostream& os, const String& str);
};


String::String() : data(new char[1] {'\0'}), size(0) {}
```

```
String::String(const char* str) {
    size_t len = 0;
    while (str[len] != '\0') {
        ++len;
    }

    size = len;
    data = new char[len + 1];
    for (size_t i = 0; i < len; ++i) {
        data[i] = str[i];
    }
    data[len] = '\0';
}

String::~String() {
    delete[] data;
}

String& String::operator=(const String& other) {
    if (this == &other) {
        return *this;
    }

    delete[] data;
    data = new char[other.length() + 1];
    size = other.size;
    size = other.size;
    for (size_t i = 0; i < size; ++i) {
        data[i] = other.data[i];
    }
    data[size] = '\0';
    return *this;
}

size_t String::length() const {
    return size;
}
```

String 구현

KWANGWOON UNIVERSITY

```
void String::append(const String& other) {
    size_t newLength = size + other.length();
    char* newData = new char[newLength + 1];

    for (size_t i = 0; i < size; ++i) {
        newData[i] = data[i];
    }

    for (size_t j = 0; j < other.length(); ++j) {
        newData[size + j] = other[j];
    }

    newData[newLength] = '\0';

    delete[] data;
    data = newData;
    size = newLength;
}

size_t String::copy(char* str, size_t len, size_t index) const {
    if (index ≥ size) {
        return 0;
    }

    size_t copyLength = (index + len ≤ size) ? len : (size - index);

    for (size_t i = 0; i < copyLength; ++i) {
        str[i] = data[index + i];
    }

    return copyLength;
}

char& String::operator[](size_t index) {
    if (index ≥ size) {
        throw std::out_of_range("Index out of range");
    }
    return data[index];
}
```

```
const char& String::operator[](size_t index) const {
    if (index ≥ size) {
        throw std::out_of_range("Index out of range");
    }
    return data[index];
}

void String::swap(String& other) noexcept {
    char* tempData = data;
    data = other.data;
    other.data = tempData;

    size_t tempUsed = size;
    size = other.size;
    other.size = tempUsed;
}

std::ostream& operator<<(std::ostream& os, const String& str) {
    os << str.data;
    return os;
}

int main() {
    String str1 = "Hello World!";
    std::cout << "str1: " << str1 << std::endl;

    String str2 = "Bonjour!";
    std::cout << "str2: " << str2 << std::endl << std::endl;

    str1.swap(str2);
    std::cout << "---After swap---" << std::endl;
    std::cout << "str1: " << str1 << std::endl;
    std::cout << "str2: " << str2 << std::endl;

    return 0;
}
```

String 구현

KWANGWOON UNIVERSITY

□ 컴파일 결과

```
str1: Hello World!
str2: Bonjour!

---After swap---
str1: Bonjour!
str2: Hello World!
```