(5) C++ string 클래스 사용법 한번에 끝내기

사용자 ldgeao99 2019. 4. 19. 14:25

참고 : http://yotop93.blogspot.com/2015/04/string.html

C++에서 문자열을 다룰 수 있는 두 가지 방법

 # C-스트링

- C언어에서 사용해오던 전통적인 문자열로, '₩0'으로 끝나는 char타입의 배열을 취급하는 방법이다.

```
char s[100];
scanf("%s", s);
```

 # string 클래스

- C++ 표준 라이브러리에서 제공하는 클래스로 문자열의 크기를 동적으로 변경 가능하며, Python의 문자열 방식과 비슷하다.

```
string str = "";

getline(cin, str);    // 문자열 입력받기
cout << str;

str.append("you");    // 문자열 뒤에 문자열 추가하기
cout << str;
cout << str[0];
```

string 클래스 객체의 입/출력

 # string 객체의 입/출력

- cin, getline / cout을 이용하여 입출력을 할 수 있다.(scanf, printf 사용불가)

```
string str;

cin >> str;             // 스페이스가 들어간 문자열의 경우 첫번째 것만 입력받으므로 주의해야함.
cout << str;

getline(cin, str);      // 스페이스가 있어도 한줄을 통째로 입력받음. getline(cin, str, '\n')와 같음.
cout << str;

getline(cin, str, 'a'); // a 문자 앞까지만 입력받음 "bgafeq" -> bg
cout << str;
```

string 클래스 객체의 여러가지 생성 방법

 # string 객체 생성 방법

```
// 빈 문자열을 가지도록 생성하는 방식
string str;

// 생성자에 초기값을 넘겨주는 방식
string str = "abcde";
string str("abcde");

// C-스트링과의 호환을 보여주는 방식
char s[] = {'a', 'b', 'c', 'd', 'e'};
string str(s);

// 문자열 내용을 복사한 새로운 객체 생성 방식
string str = "abcde";
string str2(str);       // string 객체의 문자열 내용을 복사하여 새로운 객체를 생성함(주소복사X)

// new, delete를 이용한 동적할당 방식1
string *p = new string();       // new 를 이용하여 객체를 생성하려면 포인터변수로 선언해야한다.
p->append(" Great!!");          // 포인터변수 p가 가리키고 있는 객체의 함수 append 사용
cout << *p << endl;             // 포인터 변수가 가리키고 있는 객체의 값을 가져오려면 '*'을 붙여줘야 한다.
delete p;                       // 메모리 반환


// new, delete를 이용한 동적할당 방식2
```

```
string *p = new string("C++");  // 위의 방식에서 여기만 다름
p->append(" Great!!");
cout << *p << endl;
delete p;

// new, delete를 이용한 동적할당 방식3
string str = "abcd"
string *p = new string(str);    // 위의 방식에서 여기만 다름
p->append(" Great!!");
cout << *p << endl;
delete p;
```

string 클래스 객체끼리의 비교연산

문자열이 같은지 비교

- '==' 연산자를 이용한다.

```
string str1 = "abcde";
string str2 = "abcde";
string str3 = "abcff";

cout << (str1 == str2) << " " << (str1 == str3) << endl; // true(0) fasle(1) 출력
```

사전순서 비교

- '< or >' 연산자를 이용한다.

- 맨 앞의 문자뿐만 비교하는게 아니라 전체 문자를 비교해본 결과가 나옴

```
string str1 = "abcde";
string str2 = "ccccc";

cout << (str1 < str2) << " " << (str1 > str2) << endl; // true(1) false(0) 출력
```

위 두가지 행위를 compare 함수로도 할 수 있음

- 비효율적이므로 위의 방식을 사용하자.

```
int result;
string name1 = "appaa";
string name2 = "appbb";

result = name1.compare(name2); // 문자열이 같으면 0을, 매개 객체보다 앞순이면 -1을, 매개 객체보다 뒷순이면

if(result == 0)
    cout << "두 문자열은 같음" << endl;

else if(result < 0)
    cout << "name1이 앞에옴" << endl;

else if(result > 0)
    cout << "name2이 앞에옴" << endl;

cout << result;
```

string 클래스 객체의 배열 사용법

string 객체를 한 문자씩 접근하기

```
string str = "abcde";

cout << str[0];      // 'a' 출력
```

string 객체 배열을 한 객체씩 접근하기

```
string str[5];

string[0] = "abcde";
string[1] = "fghij";

cout << str[0];      // "abcde"출력
```

string 클래스의 자주 사용되는 메소드

length(), size(), capacity()

- 문자열 길이 반환

- 문자열 길이 반환

- 사용중인 메모리 크기 반환

```
string str = "012345";

cout << str.length() << endl;    // 6 문자열길이 반환
cout << str.size() << endl;      // 6 문자열길이 반환(length, size는 동작결과가 동일함)
cout << str.capacity() << endl;  // 22 이 객체가 사용중인 메모리 크기(가변)
```

 # append() 또는 '+'

- 문자열 연결
- append()의 경우 인자로 'c' 같이 char 형을 사용할 수 없다. "c" 는 가능함. 'c'를 더해주고 싶은 경우 + 연산을 이용하자.

```
string str = "aaa";

str.append("bbb");
cout << str << endl;   // "aaabbb"

str += "ccc";
cout << str << endl;   // "aaabbbccc"
```

 # insert()

- 문자열 삽입

```
string str = "012345";

str.insert(2,"bbb"); // index가 2인 위치에 있는 문자 앞에 삽입함.
cout << str << endl; // "01bbb2345"
```

 # replace()

- 문자열 대체

```
string str = "012345";
```

```
str.replace(2,3,"bbb"); // index가 2인 위치에 있는 문자부터 ~ 3개의 문자를 "bbb"로 대체함
cout << str << endl;    // "01bbb5"
```

 # erase()

- 부분 지우기

```
string str = "012345";

str.erase(1, 4);       // index 1 ~ 4 인 부분을 부분적으로 지움

cout << str << endl;   // "05"
```

 # clear()

- 전체 지우기

```
string str = "012345";

str.clear();           // 저장되어 있는 문자열을 모두 지움

cout << str << endl;   // ""
```

 # substr()

- 부분 문자 반환받기

```
string str = "012345";

cout << str.substr(2) << endl;    // "2345"  index 2의 위치부터 ~ 끝까지의 문자를 반환함
cout << str.substr(2,3) << endl;  // "234"   index 2의 위치부터 3개의 문자를 반환함
```

 # find()

- 문자가 존재하는 경우, 해당 위치의 index 반환받기

- 문자열이 존재하는 경우, 문자열이 시작되는 index 반환받기

- 존재하지 않으면 -1을 반환받음

```
string str = "kkk abc aaa";
int result;

result = str.find("f");
cout << result << endl;  // -1 : "f" 가 존재하지 않으므로.

result = str.find("aac");
cout << result << endl;  // -1 : "aac" 가 존재하지 않으므로.

result = str.find("aaa");
cout << result << endl;  //  8 : "aaa" 가 시작되는 인덱스가 8이므로.

result = str.find("kkk", 4);
cout << result << endl;  //  -1 : 인덱스 4의 위치부터 "kkk" 를 찾는데 존재하지 않으므로.
```

 # [] 혹은 at()

- 해당 인덱스의 문자 한개를 char 형으로 반환함.

```
string str = "012345";
char c;

c = str[1];
cout << c << endl;

c = str.at(1); // 위와 동일한 표현
cout << c << endl;
```

 # stoi() 또는 to_string()

- string -> int 변환을 수행하거나 ('s'tring 'to' 'i'nt 의 약자 stoi), int -> string 변환을 수행함.

- stoi() 함수의 경우 인자는 반드시 string형을 넘겨주어야 함.(str[0]은 char형이므로 불가)
- 비주얼 c++2008 이하는 atoi(str.c_str()) 이걸 써야함

```
// string의 문자열 전체를 숫자로 변환하는 경우
string str = "2000";
int a = stoi(str);

// string의 문자 한개를 숫자로 변환하는 경우
string str = "2000";
```

```
string temp;
temp = str[0];          // index를 포함한 값인 경우 여기처럼 선언과 할당을 분리시켜줘야 한다.
int b = stoi(temp);

int a = 12;
string str;
str = to_string(a);
```

 # toupper() 또는 tolower()

- 소문자를 대문자로 변환(단, 하나의 문자씩만 변환이 가능함)

- 소문자를 대문자로 변환(단, 하나의 문자씩만 변환이 가능함)

```
string str = "abcde";
str[2] = toupper(str[2]);
cout << str;                // "abCde"

string str = "abcde";
str[2] = tolower(str[2]);
cout << str;                // "ABcDE"
```

 # isdigit()

- 문자가 숫자인지 아닌지 판별해줌

```
string str = "1ABCDE";

cout << isdigit(str[0]) << endl; // 1(true)
cout << isdigit(str[2]) << endl; // 0(false)
```

 # isalpha()

- 문자가 영어인지 아닌지 판별해줌

```
string str = "1ABCDE";

cout << isalpha(str[0]) << endl; // 0(false)
cout << isalpha(str[2]) << endl; // 1(true)
```

 # empty()

- 문자열이 비어있나 판별해줌

```
string str1 = "";
cout << str1.empty() << endl; // 1(true)

string str2 = "abcde";
cout << str2.empty() << endl; // 0(false)
```

 # swap()

- 스트링 문자열을 서로 교환함

```
string str1 = "aaa";
string str2 = "ccc";

str1.swap(str2);

cout << str1 << endl; // "ccc"
cout << str2 << endl; // "aaa"
```

 # pop_back() 또는 push_back()

- 맨 뒤의 문자를 pop하거나, 맨 뒤에 push 함.

```
string str = "abc";

str.pop_back();
cout << str << endl; // "ab"

str.push_back('c');
cout << str << endl; // "abc"
```

 # front() 또는 back()

- 맨 뒤의 문자를 반환하거나, 맨 뒤의 문자를 반환함.

```
string str = "abc";

cout << str.front() << endl; // "a"
cout << str.back() << endl;  // "e"
```