Lab 4

Vector

- Vectors are sequence containers representing arrays that can ch ange in size.
- Like arrays, vectors use contiguous storage locations for their elements.
- Vectors consume more memory in exchange for the ability to manage storage and grow dynamically in an efficient way.
- #import <vector>

Vector - Constructor

- `vector<int> v1;`
- `vector<int> v2(INITAL_SIZE, INITAL_VALUE);`
- `vector<int> v3(v2);`
- `vector<int> v4 {10, 20, 30}; // c++ 11~`

https://www.cplusplus.com/reference/vector/vector/vector/

Vector Examples

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

using namespace std;

void print_vector(vector<int> v) {
    cout << "[print_vector]: ";

    if (v.empty()) cout << "vector is empty";
    else for (int i = 0; i < v.size(); i++) {
        cout << v.at(i) << " ";
    }

    cout << endl;
}

void size_and_capacity(vector<int> v) {
    cout << "[size_and_capacity]: " << v.size() << " " << v.capacity
    () << endl;
}
```

- Define our helper functions
- print_vector() prints all elements in the vector.
- size_and_capacity() prints the size and capacity of the vector.
- We can access an element of vector like vector.at(idx) or vector[idx]

Vector Examples

```
21    int main() {
22        vector<int> v;
23
24        v.push_back(3);
25        v.push_back(2);
26        v.push_back(1);
27
28        print_vector(v);
29        size_and_capacity(v);
30
31        return 0;
32    }
```

```
❯ g++ vector_prac.cpp -o vector_prac && ./vector_prac
[print_vector]: 3 2 1
[size_and_capacity]: 3 3
```

- Let's add some value
- push_back(value) appends new value at the **end** of the array.
- Amortized constant time complexity
- How to insert in the **middle** of the arr ay? (later)

Vector Examples

```cpp
int main() {
    vector<int> v;

    v.push_back(3);
    v.push_back(2);
    v.push_back(1);

    print_vector(v);
    size_and_capacity(v);

    cout << v.front() << v.back() << v.at(1) << v[1] << endl;

    return 0;
}
```

```
❯ g++ vector_prac.cpp -o vector_prac && ./vector_prac
[print_vector]: 3 2 1
[size_and_capacity]: 3 3
3122
```

- Access some value
- front() accesses the first element
- back() accesses the last element
- operator [] and at(idx) allows to specify the position to access. (what's the difference?)
- Constant time complexity

Vector Examples

```cpp
int main() {
    vector<int> v;

    v.push_back(3);
    v.push_back(2);
    v.push_back(1);

    print_vector(v);
    size_and_capacity(v);

    v.pop_back();
    print_vector(v);
    size_and_capacity(v);

    v.clear();
    print_vector(v);
    size_and_capacity(v);

    return 0;
}
```

```
❯ g++ vector_prac.cpp -o vector_prac && ./vector_prac
[print_vector]: 3 2 1
[size_and_capacity]: 3 3
[print_vector]: 3 2
[size_and_capacity]: 2 2
[print_vector]: vector is empty
[size_and_capacity]: 0 0
```

- Delete some value
- pop_back() removes the last element
- clear() wipes all contents and make si ze 0
- Constant time complexity
- How to erase other elements? (Later)

Size and Capacity in vector

- size() returns **number of elements in the vector.**
- capacity() returns **how much space currently allocated**
- size() and capacity() can return **different value** because the vector expa nds to guarantee amortized constant time complexity.
- clear() reduces only the size of vector, not the capacity.

Size and Capacity in vector

```
21  int main() {
22      vector<int> v;
23
24      for (size_t i = 0; i < 16; i++) {
25          v.push_back(i);
26
27          cout << "size: " << v.size() << " capacity: " << v.capacity()
            << endl;
28      }
29
30      return 0;
31  }
```

```
❯ ./vector_prac
size: 1 capacity: 1
size: 2 capacity: 2
size: 3 capacity: 4
size: 4 capacity: 4
size: 5 capacity: 8
size: 6 capacity: 8
size: 7 capacity: 8
size: 8 capacity: 8
size: 9 capacity: 16
size: 10 capacity: 16
size: 11 capacity: 16
size: 12 capacity: 16
size: 13 capacity: 16
size: 14 capacity: 16
size: 15 capacity: 16
size: 16 capacity: 16
```

Vector Iterator

```
21  int main() {
22      vector<int> v;
23      vector<int>::iterator it;
24
25      v.push_back(0);
26      v.push_back(1);
27      v.push_back(2);
28      v.push_back(3);
29      v.push_back(4);
30
31      it = v.begin();
32      cout << *it << endl;
33
34      it++;
35      cout << *it << endl;
36
37      it = v.end();
38      it--;
39      cout << *it << endl;
40
41      return 0;
42  }
```

```
❯ g++ vector_prac.cpp -o vector_prac && ./vector_prac
0
1
4
```

- Iterator is an object that points to an element inside the container.
- Iterators have much more features/restrictions than pointers.
- Vector gives **random access iterator,** which supports arbitrary access. (operator ++, --, + , - +=, -=, [])

Vector Reverse Iterator

```
21  int main() {
22      vector<int> v;
23      vector<int>::reverse_iterator it;
24
25      v.push_back(0);
26      v.push_back(1);
27      v.push_back(2);
28      v.push_back(3);
29      v.push_back(4);
30
31      it = v.rbegin();
32      cout << *it << endl;
33
34      it++;
35      cout << *it << endl;
36
37      it = v.rend();
38      it--;
39      cout << *it << endl;
40
41      return 0;
42  }
```

```
❯ g++ vector_prac.cpp -o vector_prac && ./vector_prac
4
3
0
```

- **reverse_iterator** iterates backwards: increasing them moves them towards the beginning the container.
- Therefore, rend() returns the virtual element that precedes the first element of vector.

Vector Functions with iterator

```
21  int main() {
22      vector<int> v;
23      vector<int>::iterator it;
24
25      v.push_back(0), v.push_back(1), v.push_back(2), \
26      v.push_back(3), v.push_back(4);
27
28      it = v.begin();
29      it += 2;
30
31      print_vector(v);
32      v.erase(it);
33      print_vector(v);
34      v.insert(it, 10);
35      print_vector(v);
36
37      return 0;
38  }
```

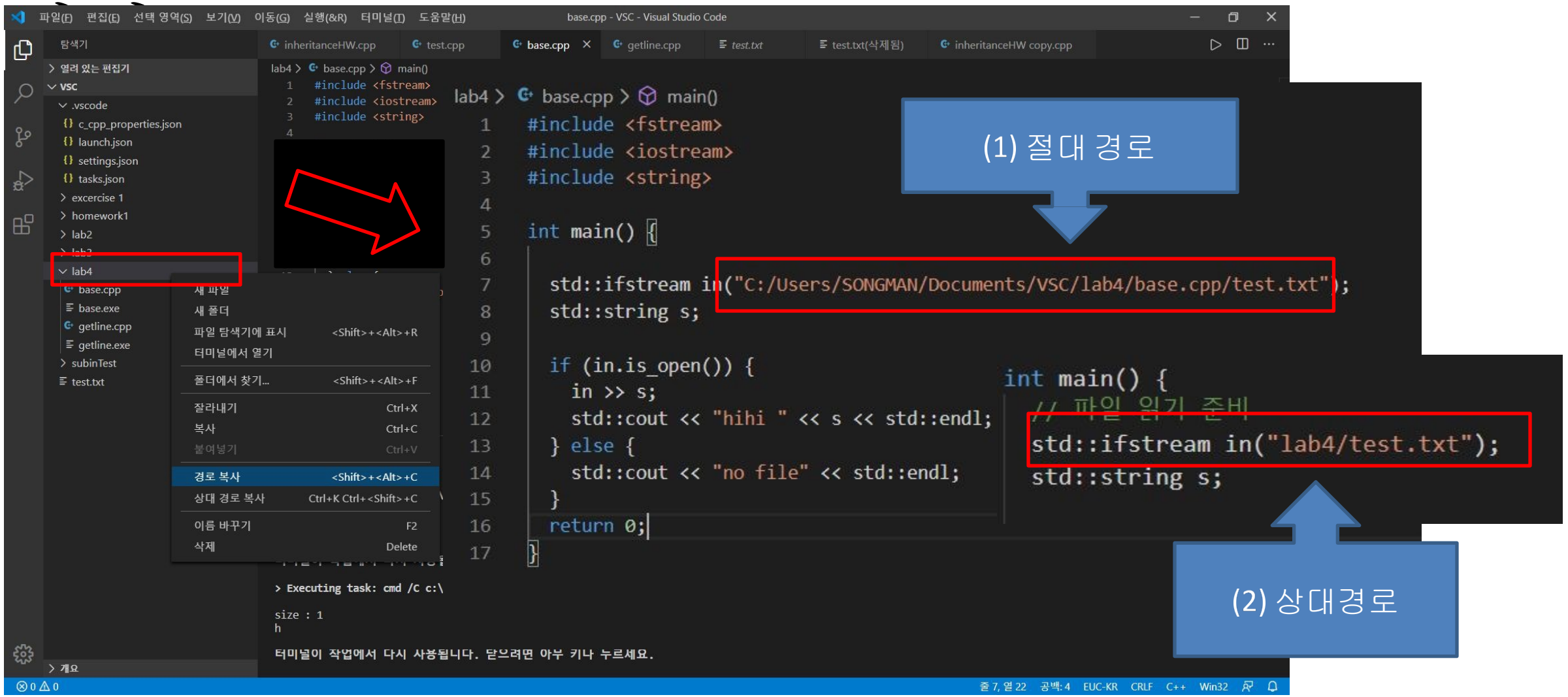
```
❯ g++ vector_prac.cpp -o vector_prac && ./vector_prac
[print_vector]: 0 1 2 3 4
[print_vector]: 0 1 3 4
[print_vector]: 0 1 10 3 4
```

- We can manipulate vector using iterators
- insert(iterator, value) inserts an element before the element at the specified positi on.
- erase(iterator) removes an element.
- erase(iterator, iterator) removes elements between within two iterators.

C++ File I/O

- C++ file I/O are easy to implement using the class of ofstream and ifstream in the **fstream** library.
- ofstream: File classes for write operations (derived from ostream)
- ifstream: File classes for read operations (derived from istream)
- **fstream**: File classes for both reads and writes (derived from iostream)

“test.txt” 경로


base.cpp - VSC - Visual Studio Code
inheritanceHW.cpp
test.cpp
base.cpp
getline.cpp
test.txt
test.txt(삭제됨)
inheritanceHW copy.cpp
lab4 > base.cpp > main()
#include <fstream>
#include <iostream>
#include <string>
int main() {
std::ifstream in("C:/Users/SONGMAN/Documents/VSC/lab4/base.cpp/test.txt");
std::string s;
if (in.is_open()) {
in >> s;
std::cout << "hihi " << s << std::endl;
} else {
std::cout << "no file" << std::endl;
}
return 0;
}
(1) 절대 경로
int main() {
// 파일 읽기 준비
std::ifstream in("lab4/test.txt");
std::string s;
(2) 상대경로
새 파일
새 폴더
파일 탐색기에 표시 <Shift>+<Alt>+R
터미널에서 열기
폴더에서 찾기... <Shift>+<Alt>+F
잘라내기 Ctrl+X
복사 Ctrl+C
붙여넣기 Ctrl+V
경로 복사 <Shift>+<Alt>+C
상대 경로 복사 Ctrl+K Ctrl+<Shift>+C
이름 바꾸기 F2
삭제 Delete
> Executing task: cmd /C c:\
size : 1
h
터미널이 작업에서 다시 사용됩니다. 닫으려면 아무 키나 누르세요.

C++ File I/O - INPUT

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

using namespace std;

int main() {
    ifstream in("test.txt");
    string s;

    if(in.is_open()) {
        in >> s;
        cout << "input: " << s << endl;
    } else {
        cout << "file not found!" << endl;
    }

    in.close();
    return 0;
}
```

function open:
open (filename, mode);
Where filename is a string representing the name of the file to be opened, and mode is an optional parameter with a combination of the following flags: (next slide)

```
~/Desktop/cplab4  9-file                                  base  01:32:50 AM
❯ echo 123123123 >> test.txt

~/Desktop/cplab4  9-file !1                               base  01:32:53 AM
❯ ./file_input
input: 123123123
```

ios::in	Open for input operations.
ios::out	Open for output operations.
ios::binary	Open in binary mode.
ios::app	All output operations are performed at the end of the file, appending the content to the current content of the file.
ios::trunc	If the file is opened for output operations and it already existed, its previous content is deleted and replaced by the new one.

ifstream dafult parameter

ofstream dafult parameter

All the flags can be combined using the bitwise operator OR (|).

```
ofstream myfile;
myfile.open ("example.bin", ios::out | ios::app | ios::binary);

//conduct the same opening operation
ofstream myfile ("example.bin", ios::out | ios::app | ios::binary);
```

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

int main() {
  // 파일 읽기 준비
  std::ifstream in("test.txt");
  std::string s;

  if (in.is_open()) {
    in >> s;
    std::cout << "input :" << s << std::endl;
  } else {
    std::cout << "can't find the file!" << std::endl;
  }

  in.close();
  in.open("other.txt");

  if (in.is_open()) {
    in >> s;
    std::cout << "input :: " << s << std::endl;
  } else {
    std::cout <<"can't find the file!" << std::endl;
  }

  return 0;
}
```

C++ File I/O – OPEN() & CLOSE()

in 객체를 다른 file로 재사용할 때는 이전에 꺼를 close() 반드시 해줘야 함.

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

int main() {
  // 파일 읽기 준비
  std::ifstream in("test.txt");
  std::string s;

  if (in.is_open()) {
    // 위치 지정자를 파일 끝으로 옮긴다.
    in.seekg(0, std::ios::end);

    // 그리고 그 위치를 읽는다. (파일의 크기)
    int size = in.tellg();

    // 그 크기의 문자열을 할당한다.
    s.resize(size);

    // 위치 지정자를 다시 파일 맨 앞으로 옮긴다.
    in.seekg(0, std::ios::beg);

    // 파일 전체 내용을 읽어서 문자열에 저장한다.
    in.read(&s[0], size);
    std::cout << s << std::endl;
  } else {
    std::cout << "can't find the file!" << std::endl;
  }

  return 0;
}
```

File 전체 읽기

seekg(offset, location) :
locatio기준으로 offse만큼 이동한 위치를 나타낸다.

tellg() :
커서전까지의 데이터 크기를 알려준다.

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

int main() {
  // 파일 읽기 준비
  std::ifstream in("test.txt");
  char buf[100];

  if (!in.is_open()) {
    std::cout << "can't find the file!" << std::endl;
    return 0;
  }

  while (in) {
    in.getline(buf, 100);
    std::cout << buf << std::endl;
  }

  return 0;
}
```

ifstream getline()

buffer size를 명시해줘야함

std:: getline()

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

using namespace std;

int main() {
    ifstream in("test.txt");
    string s;

    while(in) {
        getline(in, s);
        cout << s << endl;
    }

    return 0;
}
```

buffersize 없어도 됨

C++ File I/O - OUTPUT

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

using namespace std;

int main() {
    ofstream out("output.txt");
    string s;

    if(out.is_open()) {
        out << "hello world!";
    }

    return 0;
}
```

- File write

  ofstream object_name(“file_name");

```
❯ cat output.txt
hello world!%
```

OUTPUT – APPEND

```
1
2   #include <fstream>
3   #include <iostream>
4   #include <string>
5
6   int main() {
7     std::ofstream out("test.txt", std::ios::app);
8
9     out<< "append hello";
10    return 0;
11  }
```

Exercise

- **Pre-requisite knowledge** before Project 1
- String manipulation(Search, Split, Compare, …)
- No submit; just for your own benefit
- Your goal is to show the right result from a text file.
- This week’s task is done with cpp.

From a text file with one string line

- Go over to **Visual Studio Code**
- Make a new file titled “fileIOprac.txt”.
- In the text file, copy the following string and save it.

James,1992,10,23,180,male

- Get the string from the file (using ifstream)
- Save that string into a variable which data type is “char []”
- Accumulate your string to a vector<char *>.

Sample code (char [] split)

```cpp
#include <iostream>
#include <cstring>
#include <vector>
#include <fstream>
using namespace std;

int main() {
    char str[512];
    char *pch;
    const char* delimiter = ",";
    vector<char*> a;

    string filePath = "fileIOprac.txt";

    ifstream openFile(filePath.data());

    if(openFile.is_open()) {
        openFile.getline(str,512);
        openFile.close();
    }

    printf("Splitting string \"%s\" into tokens:\n", str);
    pch = strtok(str,delimiter);
    while(pch!=NULL) {
        printf("%s\n",pch);
        a.push_back(pch);
        pch = strtok(NULL,delimiter);
    }

    return 0;
}
}
```

reference: http://www.cplusplus.com/reference/cstring/strtok/

output:

```
PS C:\Users\DK\Desktop\cppExample\lab4example> .\fileexample.exe
Splitting string "James,1992,10,23,180,male" into tokens:
James
1992
10
23
180
male
```

Sample code continued

```
#include <iostream>
#include <cstring>
#include <vector>
#include <fstream>
using namespace std;

int main ()

{
        char str[] = "Hello,my-name/is.James";
        char title[] = "RTH";
        int i = 0;

        char * pch;
        const char* delimiter = ",./-";
        vector<char *> a;

        printf ("Splitting string \"%s\" into
        tokens:\n",str);  pch = strtok (str,delimiter);
        while (pch != NULL)
        {
        printf ("%d: %s\n",++i,pch);

         a.push_back(pch);
         pch = strtok (NULL,delimiter);
         }
         return 0;
 }
```

- You can also use multiple delimiters at the same time!

```
PS C:\Users\DK\Desktop\cppExample\lab4example> .\a.exe
Splitting string "Hello,my-name/is.James" into tokens:
1: Hello
2: my
3: name
4: is
5: James
```

You get the idea right?

Please make it your own because you will be using this skill doing your project 1!

Sample code (Split with string data type)

```cpp
#include <iostream>
#include <sstream>
#include <cstring>
#include <vector>
#include <fstream>

using namespace std;

int main()
{
        string filePath = "fileIOprac.txt";  string
        str;
        ifstream openFile(filePath.data());
        if(openFile.is_open()) {
                getline(openFile, str);
                openFile.close();
        }

        stringstream ss(str);
        vector<string> result;

        while( ss.good() )
        {
                string substr;
                getline( ss, substr, ',' );
                result.push_back( substr);
                cout << substr << endl;
        }
        return 0;
}
```

- You get the same output as the first one.

```
PS C:\Users\DK\Desktop\cppExample\lab4example> .\a.exe
James
1992
10
23
180
male
```