Lab 4
Vector

- Vectors are sequence containers representing arrays that can change 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(INITIAL_SIZE, INITIAL_VALUE);`
- `vector<int> v3(v2);`
- `vector<int> v4 {10, 20, 30}; // c++ 11~`

Vector Examples

- 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

```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);
    return 0;
}
```

• 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 array? (later)
Vector Examples

• 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

```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);
    return 0;
}
```

> g++ vector_prac.cpp -o vector_prac & & ./vector_prac
[print_vector]: 3 2 1
[size_and_capacity]: 3 3
3122
Vector Examples

- Delete some value
- `pop_back()` removes the last element
- `clear()` wipes all contents and make size 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 expands to guarantee amortized constant time complexity.
- `clear()` reduces only the size of vector, not the capacity.
Size and Capacity in vector

```cpp
int main() {
    vector<int> v;
    for (size_t i = 0; i < 16; i++) {
        v.push_back(i);
        cout << "size: " << v.size() << " capacity: " << v.capacity() << endl;
    }
    return 0;
}
```
Vector Iterator

- 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

- `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.

```cpp
int main() {
    vector<int> v;
    vector<int>::reverse_iterator it;
    v.push_back(0);
v.push_back(1);
v.push_back(2);
v.push_back(3);
v.push_back(4);
    it = v.rbegin();
    cout << *it << endl;
    it++;
    cout << *it << endl;
    it = v.rend();
    it--;
    cout << *it << endl;
    return 0;
}
```

```shell
g++ vector_prac.cpp -o vector_prac && ./vector_prac
4
3
0
```
Vector Functions with iterator

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

```cpp
int main() {
    vector<int> v;
    vector<int>::iterator it;
    v.push_back(0), v.push_back(1), v.push_back(2), v.push_back(3), v.push_back(4);
    it = v.begin();
    it += 2;
    print_vector(v);
    v.erase(it);
    print_vector(v);
    v.insert(it, 10);
    print_vector(v);
    return 0;
}
```

```bash
$ 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
```
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

(1) 절대 경로

```cpp
int main()
{
    ifstream in("C:/Users/SONGMAN/Documents/VSC/lab4/base.cpp/test.txt");
    string s;
    if (in.is_open())
    {
        in >> s;
        cout << "hihi" "s" std::endl;
    }
    else {
        cout << "no file" std::endl;
    }
    return 0;
}
```

(2) 상대 경로

```cpp
int main()
{
    ifstream in("lab4/test.txt");
    string s;
    return 0;
}
```
function open:
onopen (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)
<table>
<thead>
<tr>
<th>ios::in</th>
<th>Open for input operations.</th>
<th>ifstream default parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>ios::out</td>
<td>Open for output operations.</td>
<td>ofstream default parameter</td>
</tr>
<tr>
<td>ios::binary</td>
<td>Open in binary mode.</td>
<td></td>
</tr>
<tr>
<td>ios::app</td>
<td>All output operations are performed at the end of the file, appending the content to the current content of the file.</td>
<td></td>
</tr>
<tr>
<td>ios::trunc</td>
<td>If the file is opened for output operations and it already existed, its previous content is deleted and replaced by the new one.</td>
<td></td>
</tr>
</tbody>
</table>

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

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

//conduct the same opening operation
```cpp
ofstream myfile ("example.bin", ios::out | ios::app | ios::binary);
```
C++ File I/O – OPEN() & CLOSE()

#include <fstream>
#include <iostream>
#include <string>

int main() {
  // 파일 입출력 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;
}

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

seekg(offset, location) :
lacatio기준으로 offset만큼 이동한 위치를 나타낸다.
tellg() :
커서전까지의 데이터 크기를 알려준다.
#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;
}
std:: getline()
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");

```bash
cat output.txt
hello world!
```
```cpp
#include <fstream>
#include <iostream>
#include <string>

int main() {
    std::ofstream out("test.txt", std::ios::app);
    out << "append hello";
    return 0;
}
```
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;
}
```


output:

```
Splitting string "James,1992,10,23,180,male" into tokens:
James
1992
10
23
180
male
```
Sample code continued

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

using namespace std;

int main()
{
    string filePath = "fileOprac.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.