14주차_12장 c++ 파일 입출력 13장 예외처리

텍스트파일

- 사람들이 사용하는 글자, 혹은 문자로만 구성되는 문서파일
- c, cpp, java 소스파일, txt, html, xml 파일

바이너리파일

- 사진, 오디오, 그래픽이미지, 컴파일된 코드 등은 문자로 표현되지 않는 바이너리 정보
- 바이너리파일은 이런 바이너리 정보들을 저장한다
- jpeg, bmp 이미지, mp3 오디오, hwp, doc, ppt 멀티미디어 문서, exe,obj 컴파일코드

ex 12-1) txt파일 저장하기

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

int main(){
    char name[10],dept[20];
    int sid;

    // 키보드로부터 읽기
    cout << "이름: ";
    cin >> name;
    cout << "학번: ";
    cin >> sid;
    cout << "학과: ";
    cin >> dept;

    // 파일 열기
    ofstream fout("student.txt");
    if(!fout){
        cout << "파일을 열 수 없습니다." << endl;
        return 1;
    }

    // 파일 쓰기
    fout << name << endl;
    fout << sid << endl;
    fout << dept << endl;

    // 파일 닫기
```

```
    fout.close();
}
```

ex 12-3) get을 이용한 텍스트 파일 읽기

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

int main(){
    const char* filename = "student.txt";

    ifstream fin(filename);
    if(!fin){
        cout << "파일을 열 수 없습니다." << endl;
        return 0;
    }
    int cnt =0;
    int c;
    while((c = fin.get()) != EOF){ // EOF를 만날 때까지 읽기 end of file
        cout << (char)c;
        cnt++;
    }
    cout << "읽은 바이트 수는 " << cnt << "입니다." << endl;
    fin.close();
}
```

get() 과 EOF 원리
EOF = end of file , 파일이 끝나는 지점.
get() 은 파일의 끝에서 읽기를 시도하면 get() 은 EOF(-1값) 를 리턴한다

텍스트 파일의 라인단위 읽어들이기

2가지방법

- istream 의 getline(char* line, int n) 함수이용
- getline(ifstream& fin, string& line) 함수이용

ex 12-5) istream의 getline()

```
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
    ifstream fin("system.ini");
    if (!fin)
```

```
    {
        cout << "system.ini 열기 실패" << endl;
        return 0;
    }
    char buf[81]; // 한 라인이 최대 80개의 문자로 구성된다고 가정
    while (fin.getline(buf, 81))
    {                          // 한 라인이 최대 80개의 문자로 구성
        cout << buf << endl; // 라인 출력
    }
    fin.close();
}
```

ex 12-6) getline(ifstream&, string&)

- 전체코드는 따로 확인해볼것. getline 으로 가져오는 방식만 체크.

```
void fileRead(vector<string> &v, ifstream &fin)
{ // fin으로부터 벡터 v에 읽어 들임
    string line;
    while (getline(fin, line)) // <<--- getline
    {                       // fin 파일에서 한 라인 읽기
        v.push_back(line); // 읽은 라인을 벡터에 저장
    }
}
void search(vector<string> &v, string word)
{ // 벡터 v에서 word를 찾아 출력
    for (int i = 0; i < v.size(); i++)
    {
        int index = v[i].find(word);
        if (index != -1) // found
            cout << v[i] << endl;
    }
}
```

바이너리 I/O

- ios::binary 속성으로 설정하는것이 필수, 디폴트가 text I/O 이기 때문.
- get(), put() 으로 동일하게 바이너리 바이트를 읽고 쓸 수 있다.

read(), write()로 블록 단위 파일 입출력

- get과 put은 한 바이트 단위로 입출력을 수행하지만, read와 write는 **블록단위로 입출력**
- read() : istream , write() : ostream

ex 12-8) read를 이용해 블록 단위로 텍스트 파일 읽기

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

int main()
{
    const char *file = "system.ini";
    ifstream fin;

    fin.open(file, ios::in | ios::binary); // 읽기 모드로 파일 열기
    if (!fin)
    { // 열기 실패 검사
        cout << "파일 열기 오류";
        return 0;
    }

    int count = 0;
    char s[32];

    while (!fin.eof())
    {                           // 파일 끝까지 읽는다.
        fin.read(s, 32);        // 최대 32 바이트를 읽어 배열 s에 저장
        int n = fin.gcount(); // 실제 읽은 바이트 수 알아냄
        cout.write(s, n);       // 버퍼에 있는 n 개의 바이트를 화면에 출력
        count += n;
    }
    cout << "읽은 바이트 수는 " << count << endl;
    fin.close(); // 입력 파일 닫기
}
```

- **개발자의 논리가 잘못된경우**
  -> 외적으로 발생하는 입력이나 상황에 대한 대처가 없을 때 발생하는 오류
- **예외처리를 하지 않은 경우**
  ->결과가 틀리거나 엉뚱한 코드 실행, 프로그램이 비정상적으로 종료됨

cpp 예외처리의 기본 형식

try

- 예외가 발생할 소지가 있는 문장들은 try{} **블록**으로 묶은 다음 catch(){ }**블록**을 바로 연결해 선언하며, try블록 안에는 **예외발생을 탐지하는 코드**를 작성해야함

throw

- try 안에서 실행되는 문으로 현재 실행중인 프로그램 내에 **예외의 발생을 알림

```
try { // 예외가 발생할 가능성이 있는 실행문. try {} 블록
.....................................................................
예외를 발견한다면 {
throw XXX; // 예외 발생을 알림. XXX는 예외 값
}
예외를 발견한다면 {
throw YYY; // 예외 발생을 알림. YYY는 예외 값
}
}
catch(처리할 예외 파라미터 선언) { // catch { } 블록
예외 처리문
}
catch(처리할 예외 파라미터 선언) { // catch { } 블록
예외 처리문
}
```

ex13-4) 0이나 음수로 나누는 오류를 탐지, 예외처리

```
#include <iostream>
using namespace std;

int main()
{
    int n, sum, average;
    while (true)
    {
        cout << "합을 입력하세요>>";
        cin >> sum;
        cout << "인원수를 입력하세요>>";
        cin >> n;
        try{
            if (n <= 0) throw n; // 예외 발생. catch(int x) 블록으로 점프
            else average = sum / n;
        } catch (int x)
        {
            cout << "예외 발생!! " << x << "으로 나눌 수 없음" << endl;
            average = 0;
            cout << endl;
            continue;
        }
        cout << "평균 = " << average << endl
             << endl; // 평균 출력
    }
}
```

하나의 try { } 블록에 다수의 catch() { } 블록 연결 가능

```
try{
    throw "음수 불가능"; // 1번째 catch로 간다
    throw 3;             // 2번째 catch 로
        }
catch (const char *s){ // 문자열타입 예외 처리. “음수 불가능”이 s에 전달
        }
catch (int x){ // int 타입 예외 처리. 3이 x에 전달됨
        }
}
```

함수를 포함하는 try{ } 블록

```
int main()
{
    try{
        int n = multiply(2, -3);
        cout << "곱은 " << n << endl;
    }
    catch (const char *negative){
        cout << "exception happened : " << negative;
    }
}

int multiply(int x, int y) {
        if(x < 0 || y < 0) throw "음수 불가능";
        else return x*y;
}
```

책보면서 공부하자 이것들은 ->

- 예외처리를 클래스를 통해서도 가능하다!

extern "C"

-> c컴파일러와 cpp컴파일러의 규칙이 다르기때문에 링크시에는 오류가 발생!
때문에, extern "C"를 사용해서 C파일에서 컴파일가능하도록 선언

팀원과의 논의한점

주제: 파일 입출력에서 텍스트 파일과 바이너리 파일의 선택

- 텍스트 파일은 가독성이 좋아 수정이 편리하지만, 큰 용량을 차지한다. 바이너리 파일은 용량 효율적이지만 가독성이 떨어진다. 어떤 상황에서 텍스트, 바이너리를 선택해야 하는가?

--> 예를 들어 설정 파일처럼 사람이 읽고 수정해야 하는 경우에는 텍스트 파일이 적절하겠지만, 이미지나 동영상과 같은 큰 데이터를 다룰 때는 바이너리 파일이 용량 면에서 효율적라고 생각한다.

주제 : 예외 처리와 안정성:

- 파일 입출력에서 예외 처리는 왜 필요한지에 대해 생각해보자.
  ->예외 처리를 통해 파일이 존재하지 않거나 읽기/쓰기 권한이 없는 경우에 대비할 수 있다. 특히 외부 자원을 다룰 때 예측 불가능한 상황에 대비하는 데 도움이 된다.