창의적 소프트웨어 설계

4주차 실습 – 빌드 과정 및 c언어 복습

노인우, inwoo13@hanyang.ac.kr
한중수, soohan@hanyang.ac.kr

Overview

목표

- C언어 문법 복습
  - Syntax and "Hello World!"
  - Function, Memory, Variable (and Multi-byte Variable)
  - Scope
  - Operators
  - 반복문 vs 재귀문

- 소프트웨어 설계
  - Complexity
  - Mutability
  - Flexibility

- Appendix #1: Static Member in C++ Class
- Appendix #2: Single Responsibility Principle
- Appendix #3: Monospaced Fonts

Build Stage

- Compile
- Link

```
imtutor@imtutor-desktop:~/class_materials/0925$ g++ --save-temp example.cpp -o test
imtutor@imtutor-desktop:~/class_materials/0925$ ll
total 464
drwxrwxr-x 3 imtutor imtutor   4096  9월 25 19:01 ./
drwxrwxr-x 5 imtutor imtutor   4096  9월 25 18:41 ../
-rw-rw-r-- 1 imtutor imtutor    597  9월 25 18:43 example.cpp
-rw-rw-r-- 1 imtutor imtutor 428724  9월 25 19:01 example.ii
-rw-rw-r-- 1 imtutor imtutor   4424  9월 25 19:01 example.o
-rw-rw-r-- 1 imtutor imtutor   4354  9월 25 19:01 example.s
-rwxrwxr-x 1 imtutor imtutor   9776  9월 25 19:01 test*
```

Build Stage

- Compile
- Link

```
imtutor@imtutor-desktop:~/class_materials/0925$ g++ --save-temp example.cpp -o test
imtutor@imtutor-desktop:~/class_materials/0925$ ll
total 464
drwxrwxr-x 3 imtutor imtutor   4096  9월 25 19:01 ./
drwxrwxr-x 5 imtutor imtutor   4096  9월 25 18:41 ../
-rw-rw-r-- 1 imtutor imtutor    597  9월 25 18:43 example.cpp
-rw-rw-r-- 1 imtutor imtutor 428724  9월 25 19:01 example.ii
-rw-rw-r-- 1 imtutor imtutor   4424  9월 25 19:01 example.o
-rw-rw-r-- 1 imtutor imtutor   4354  9월 25 19:01 example.s
-rwxrwxr-x 1 imtutor imtutor   9776  9월 25 19:01 test*
```

// Compile

Build Stage – .ii file

```
-rw-rw-r-- 1 imtutor imtutor    597  9월 25 18:43 example.cpp
-rw-rw-r-- 1 imtutor imtutor 428724  9월 25 19:01 example.ii
-rw-rw-r-- 1 imtutor imtutor   4424  9월 25 19:01 example.o
-rw-rw-r-- 1 imtutor imtutor   4354  9월 25 19:01 example.s
-rwxrwxr-x 1 imtutor imtutor   9776  9월 25 19:01 test*
```

example.ii

```
...

# 1 "/usr/include/x86_64-linux-gnu/c++/5/bits/c++config.h" 1 3
# 194 "/usr/include/x86_64-linux-gnu/c++/5/bits/c++config.h" 3

# 194 "/usr/include/x86_64-linux-gnu/c++/5/bits/c++config.h" 3
namespace std
{
  typedef long unsigned int size_t;
  typedef long int ptrdiff_t;
}
```

Build Stage – .s file

```
-rw-rw-r-- 1 imtutor imtutor    597  9월 25 18:43 example.cpp
-rw-rw-r-- 1 imtutor imtutor 428724  9월 25 19:01 example.ii
-rw-rw-r-- 1 imtutor imtutor   4424  9월 25 19:01 example.o
-rw-rw-r-- 1 imtutor imtutor   4354  9월 25 19:01 example.s
-rwxrwxr-x 1 imtutor imtutor   9776  9월 25 19:01 test*
```

example.s

```
…

main:
.LFB1028:
        .cfi_startproc
        .cfi_personality 0x3,__gxx_personality_v0
        .cfi_lsda 0x3,.LLSDA1028
        pushq      %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq       %rsp, %rbp
        .cfi_def_cfa_register 6
```

Build Stage – .o file

```
-rw-rw-r-- 1 imtutor imtutor    597  9월 25 18:43 example.cpp
-rw-rw-r-- 1 imtutor imtutor 428724  9월 25 19:01 example.ii
-rw-rw-r-- 1 imtutor imtutor   4424  9월 25 19:01 example.o
-rw-rw-r-- 1 imtutor imtutor   4354  9월 25 19:01 example.s
-rwxrwxr-x 1 imtutor imtutor   9776  9월 25 19:01 test*
```

example.o

```
...          ELF
7f45 4c46 0201 0100 0000 0000 0000 0000
0100 3e00 0100 0000 0000 0000 0000 0000
0000 0000 0000 0000 c80b 0000 0000 0000
0000 0000 4000 0000 0000 4000 1600 1300
0100 0000 0700 0000 0100 0000 0a00 0000
5548 89e5 4154 5348 83ec 10bf 0800 0000
e800 0000 0048 89c3 4889 dfe8 0000 0000
4889 5de8 e800 0000 0089 c3be 0000 0000
bf00 0000 00e8 0000 0000 89de 4889 c7e8
0000 0000 be00 0000 0048 89c7 e800 0000
```

Build Stage – executable file

```
-rw-rw-r-- 1 imtutor imtutor    597  9월 25 18:43 example.cpp
-rw-rw-r-- 1 imtutor imtutor 428724  9월 25 19:01 example.ii
-rw-rw-r-- 1 imtutor imtutor   4424  9월 25 19:01 example.o
-rw-rw-r-- 1 imtutor imtutor   4354  9월 25 19:01 example.s
-rwxrwxr-x 1 imtutor imtutor   9776  9월 25 19:01 test*
```

test

```
...
7f45 4c46 0201 0100 0000 0000 0000 0000
0200 3e00 0100 0000 3009 4000 0000 0000
4000 0000 0000 0000 301e 0000 0000 0000
0000 0000 4000 3800 0900 4000 2000 1d00
0600 0000 0500 0000 4000 0000 0000 0000
4000 4000 0000 0000 4000 4000 0000 0000
f801 0000 0000 0000 f801 0000 0000 0000
0800 0000 0000 0000 0300 0000 0400 0000
3802 0000 0000 0000 3802 4000 0000 0000
```

반복문 vs 재귀문

- 메모리 관리 차이:
  - 반복문: 하나의 stack 내에서 처리
  - 재귀문: 함수를 반복 호출하기 때문에 stack 메모리 누적
- 반복문의 장점:
  - 단일 쓰레드에서 수행할 때 속도, 메모리 효율이 좋음
- 재귀문의 장점:
  - 분기 재귀의 경우 sub tree를 분산 처리 가능

반복문 vs 재귀문

◆ 최대 결과값 초과 여부 확인 함수

```
#include <iostream>
#include <stdlib.h>     /* atoi */

bool check_max_exceed(unsigned int cur_num, unsigned int result) {
     const unsigned int UINT_MAX = 0xffffffff;

     // we want to check (result * cur_num > UINT_MAX)
     // but there is no space to save result * cur_num …
     // … if it is already bigger than UINT_MAX
     // so, we will going to compare result with UINT_MAX / cur_num

     if (UINT_MAX / cur_num < result) {
          std::cout << "[Error][check_max_exceed] max limit exceeded" << std::endl;
          return false;
     }

     return true;
}
```

반복문 vs 재귀문

◆ 반복문 함수:

```
unsigned int factorial_iter(unsigned int input) {
    unsigned int result = 1;
    unsigned int cur_num = input;

    while (cur_num > 0) {
        if (check_max_exceed(cur_num, result) == false) {
            return 0;
        }

        result *= cur_num--;
    }

    return result;
}
```

반복문 vs 재귀문

◆ 재귀문 함수:

```
unsigned int factorial_recu(unsigned int input) {
    unsigned int result = input * factorial_recu(input - 1);

    if (check_max_exceed(input, result) == false) {
        return 0;
    }

    return result;
}
```

반복문 vs 재귀문

◆ 메인 함수

```
int main(int argc, char **argv){
    unsigned int input = atoi(argv[1]);

    std::cout << "Input: " << input << std::endl;

    unsigned int iter_result = factorial_iter(input);
    unsigned int recu_result = factorial_recu(input);

    std::cout << "Iterative: " << iter_result << std::endl;
    std::cout << "Recursive: " << recu_result << std::endl;

    return 0;
}
```

위 메인 함수에 버그가 있다면 어떤 부분일까?

반복문 vs 재귀문

◆ 메인 함수 (error handled)

```
int main(int argc, char **argv){
	if(argc != 2){
		std::cout << "[Error][main] argument count unmatch, (" << argc << ") exit." << std::endl;
		return 0;
	}

	int input = atoi(argv[1]);

	std::cout << "Input: " << input << std::endl;

	if (input < 1) {
		std::cout << "[Error][main] input scope unmatch, exit." << std::endl;
		return 0;
	}

	unsigned int iter_result = factorial_iter((unsigned int)input);
	unsigned int recu_result = factorial_recu((unsigned int)input);

	std::cout << "Iterative: " << iter_result << std::endl;
	std::cout << "Recursive: " << recu_result << std::endl;

	return 0;
}
```

반복문 vs 재귀문

◆ 반복문 함수 (error handled):

```
unsigned int factorial_iter(unsigned int input) {
    unsigned int result = 1;
    unsigned int cur_num = input;

    if (input == 0) {
        return 0;
    }

    while (cur_num > 0) {
        if (check_max_exceed(cur_num, result) == false) {
            return 0;
        }

        result *= cur_num--;
    }

    return result;
}
```

반복문 vs 재귀문

◆ 재귀문 함수 (error handled):

```
unsigned int factorial_recu(unsigned int input) {
    if (input == 0) {
        return 0;
    }

    unsigned int result = input * factorial_recu(input - 1);

    if (check_max_exceed(input, result) == false) {
        return 0;
    }

    return result;
}
```

예외 체크 시점 버그가 있다!

반복문 vs 재귀문

◆ 재귀문 함수 (error handled 2):

```
unsigned int factorial_recu(unsigned int input) {
    if (input == 1) {
        return 1;
    }

    unsigned int fact_result = factorial_recu(input - 1);

    if (check_limit_exceed(input, fact_result) == false) {
        return 0;
    }

    unsigned int result = input * fact_result;

    return result;
}
```

동일 성능이라면 짧은 코드가 반드시 좋다?

- 상기 예제를 단일 예외처리 함수로 처리:
  - 가능은 하다
  - 그러나 코드 가독성을 해칠 수 있다
  - 불필요한 구문을 제거하는 것과 무조건 코드 라인을 줄이는 것은 다르다

Addressing Complexity

- if문을 사용 할 때, 정상 보다는 비정상을 체크!

```
// 정상을 체크
if(is_ok == true){
        // code
        if(is_sub_ok == true){
                //code blabla 1
                if(is_subsub_ok == true){
                        // code blabla 2
                }
                else{
                        // handling blabla 2
                }
        }
        else {
                // handling blabla
        }
}
```

```
// 비정상을 체크
if(is_ok == false){
        // handling blabla
}

if(is_sub_ok == false){
        // handling blabla2
}

// code blabla 1

if(is_subsub_ok == false){
        // handling blabla3
}

// code blabla 2
```

참고자료

1. 컴파일 과정, http://shinluckyarchive.tistory.com/285
2. Single Responsibility Principle, https://en.wikipedia.org/wiki/Single_responsibility_principle
3. Chain of Responsibility Pattern, https://en.wikipedia.org/wiki/Chain-of-responsibility_pattern


Q&A

Appendix #1. Static Member Function

```
#include <iostream>

class StrTest{
private:
    int iTest;
    static int m_version;

public:
    int pubTest;

    static int getVersion(){
        m_version = 10;
        return m_version;
    }

    StrTest(){
        std::cout << "constructor" << std::endl;
    }

    ~StrTest(){
        std::cout << "destructor" << std::endl;
    }
};
```

```
int StrTest::m_version;

int main(){
StrTest* st = new StrTest();

int result = StrTest::getVersion();

std::cout << "Result Version : " << result << std::endl;

delete st;

    return 0;
}
```

Appendix #2. Single Responsibility Principle

- Module or Class should have responsibility over a **single part of the functionality** provided by the software
- Responsibility should be **encapsulated** by the class

Appendix #3. Monospaced(Fixed Width) Font

- 알파벳에 상관 없이 width가 일정한 폰트
  - 코드 가독성을 향상
  - 오타 확인에 유용

Appendix #4. test_function

◆ TDD (Test Driven Development) 까지는 아니더라도!

```
bool test_factorial(unsigned int (*fact1)(unsigned int), unsigned int(*fact2)(unsigned
int), unsigned int test_limit) {
     std::cout << "[test_factorial] start! with limit: " << test_limit << std::endl;
     unsigned int result_1 = 0;
     unsigned int result_2 = 0;

     for (unsigned int n = 1; n < test_limit; n++) {
          result_1 = fact1(n);
          result_2 = fact2(n);

          if (result_1 != result_2) {
               std::cout << "[test_factorial][Error] Input: " << n << std::endl;
               std::cout << "[test_factorial][Error] fact1: " << result_1 << std::endl;
               std::cout << "[test_factorial][Error] fact2: " << result_2 << std::endl;
               return false;
          }
     }

     return true;
}
```

Appendix #4. test_function

◆ 함수 호출 부분:

```
bool test_result = test_factorial(factorial_iter, factorial_recu, 20);

if (test_result == false) {
    return 0;
}
```