실습 : tuple(1)

```
#include <iostream>
#include <string>
#include <tuple>

using namespace std;

tuple<int, string> getAgeandName( )
{
    int age;
    string name;
    cout << "나이를 입력하세요: ";
    cin >> age;

    cout << "이름을 입력하세요: ";
    cin >> name;

    return make_tuple(age, name);
}
```


실습 : tuple(2)

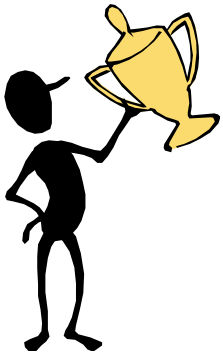
```
int main( )
{
    tuple<int, string> personInfo;

    personInfo = getAgeandName( );

    cout << "나이: " << get<0>(personInfo) << endl;
    cout << "이름: " << get<1>(personInfo) << endl;

    return 0;
}
```

2. C++ 쓰레드(Thread)



C++11의 쓰레드 클래스

- chrono
  - 나노세컨드까지 계산이 가능한 초정밀 타임 객체
- atomic
  - Lock 없이 Lock-Free로 변수값을 변경할 수 있다.
  - 정수형 또는 포인터 타입에 아토믹하게(thread-safe) 하게 이용할 수 있게하는 클래스
  - 객체들의 경우 원자적 연산 시에 메모리에 접근할 때 어떠한 방식으로 접근하는지 지정 가능
- thread
  - 멀티스레드 라이브러리
- mutex
  - 동기화 객체
- async/future
  - 함수를 비동기로 실행

실습 : chrono

```
#include <iostream>
#include <chrono>

using namespace std;
using namespace chrono;

int main( )
{
    system_clock::time_point startTime = system_clock::now( );
    for (auto i = 0; i < 1000000; i++);
    system_clock::time_point endTime = system_clock::now( );

    nanoseconds nano = endTime - startTime;

    cout << "나노초: " << nano.count( ) << endl;
    return 0;
}
```


실습 : atomic

```
#include <iostream>
#include <atomic>

using namespace std;

int main( )
{
    atomic<int> intAtomic = 1;

    intAtomic.fetch_add(1);        // 값 1 증가. 결과는 2
    cout << intAtomic << endl;

    intAtomic.fetch_sub(2);        // 값 2 감소. 결과는 0
    cout << intAtomic << endl;

    return 0;
}
```

쓰레드(Thread) 지원 라이브러리

- 기존의 C/C++의 스레드 생성은 다분히 OS 종속적
  - MS Windows : W32 Thread
  - UNIX/Linux/macOS : PThread(POSIX Thread)
  - 운영체제가 제공하는 함수를 호출해 스레드를 생성하고 조작
    → 좋지 않은 이식성
- C++11 표준에 스레드 라이브러리를 제공
  - OS 독립적이며 C++ 과도 궁합이 맞는 코드를 쉽게 생성 가능
  - std::thread 클래스 : 스레드를 생성/관리, <thread>헤더
  - std::mutex 클래스 : 상호배제, <mutex>헤더
- 자원 공유시 발생할 수 있는 문제는 mutex의 lock과 unlock으로 제어가능

실습 : thread(1)

```
#include <iostream>
#include <thread>

using namespace std;

//스레드에 의해 호출되는 함수
void call_from_thread(int tid) {
    cout << "스레드 실행 : " << tid << endl;
    std::this_thread::sleep_for(std::chrono:: nanoseconds(tid*20));
}

int main( ) {
    thread t[10];

    //10개의 스레드 시작
   thread(call_from_thread, 0).detach( );
    for (int i = 1; i < 10; ++i) {
        t[i] = thread(call_from_thread, i);
    }
```

실습 : thread(2)

```
    cout << "메인 함수 시작" << endl;

    //스레드가 종료될 때 까지 대시
    for (int i = 1; i < 10; ++i) {
        if(t[i].joinable( ))
            t[i].join( );
    }

    return 0;
}
```


실습 : mutex(1)

```
#include <iostream>
#include <thread>
#include <mutex>

using namespace std;

int value = 0;
mutex value_mutex;

void increase_value( )
{
    // 뮤텍스를 이용하여 동기화
    value_mutex.lock( );
    value++;
    cout << "스레드 실행 : " << value << endl;
    value_mutex.unlock( );
}
```

실습 : mutex(2)

```
int main( )
{
    thread t[10];

    for (auto i = 0; i < 10; i++) {
        // 스레드 생성
        t[i] = thread(increase_value);
    }

    for (int i = 0; i < 10; i++) {
        // 스레드 대기 종료
        t[i].join( );
    }

    return 0;
}
```