하계 방학 현장실습

하계 현장실습 보고서

기관명: 고등기술 연구원(IAE)

부서: 지능기계시스템센터

현장 실습생: 이 진 희

2020년 8월 24일

아주대학교

정보통신대학

소프트웨어학과

목 차

1. 서론. 5

1. 아주대 하계인턴. 5

1) 목표. 5

2. 실시간 운영체제 다루기. 6

1. Octavo 보드에 RTOS(RTLinux) 포팅을 위한 SD Card 제작. 6

1) ARM Cross Compiler: GCC 설치. 6

2) 부트로더(U-Boot) 설치 및 컴파일. 6

3) 리눅스 커널 컴파일. 7

4) 루트 파일 시스템 설치. 8

5) MicroSD 카드에 설치. 9

2. RTOS일때와 NON-RTOS일떄의 속도차이 실험. 12

1) RTOS란 무엇인가?. 12

2) CYCLIC TEST 실험. 13

3) CYCLIC TEST 결과. 14

3. 미들웨어 아키텍처 개발. 16

1. RTOS기반 미들웨어 아키텍처. 16

1) 미들웨어 아키텍처 구조 학습. 16

4. Concurrent Hierarchical Finite State Machine 실습. 17

1. 유한상태기계(Finite State Machine). 17

1) 유한상태기계(Finite State Machine). 17

2. Concurrent Hierarchical Finite State Machine (CHSM). 18

1) Concurrent Hierarchical Finite State Machine (CHSM). 18

2) CHSM Language System. 18

2) CHSM 예제 학습. 21

5. FSM기반 공정제어 시스템 구현. 23

1. FSM기반 공정제어 시스템 구현 23

1) 사용자 요구 분석 및 State Chart 제작. 23

2) 최종 사용자 요구 분석 및 State Chart 제작. 30

2. FSM기반 공정제어 시스템 코드 분석 및 시연. 32

1) chsmc 컴파일 변환 cpp 코드 분석. 32

2) 최종 결과물. 36

6. C++ 학습내용 37

1. 기타 C++ 학습내용 37

1) 얕은 복사 및 깊은 복사. 37

2) 클래스 멤버 변수 초기화 리스트. 37

3) 접근자. 38

4) 종족 멤버의 기본. 38

5) 인라인 함수, Const 함수. 38

6) 멤버 함수 포인터. 38

7) Friend class. 38

8) 객체를 멤버로 가지는 클래스. 39

9) 스마트 포인터. 39

10) 템플릿. 40

11) 클래스와 관련된 형변환. 41

12) Delete와 Delete[]의 차이점. 41

12) 메모리 얼라이먼트. 42

참고사이트. 43

부록. 44

1. 서 론

1. 아주대 하계 인턴

1) 목표

하계 방학동안 아주대학교 링크산업단을 통한 하계 방학 현장실습 인턴 신청하여 고등 기술 연구원에서 하계 방학 (7월 6일 ~ 8월 28일) 동안 인턴을 실시하게 되었다. 고등 기술 연구원 지능기계시스템 부서에 지원하였다.

실시간 운영체제의 기본 학습 및 preemptive RTOS에 대한 학습을 진행하며, 미들웨어 아키텍처 이해 및 화장 컴포넌트를 개발하고, 유한 상태 기계(FSM)을 이해하고 학습하며, 유한 상태 기계 프로그래밍 언어인 Concurrent Hierarchical Finite State Machine(CHSM)에 관련된 논물을 바탕으로 CHSM기반 사업공정 제어 시스템 소프트웨어를 구현하는 프로그램을 진행한다.

이후에 CHSM기반 산업공정 제어 시스템 소프트웨어 요구사항을 전달받아 소프트웨어 요구사항 분석 및 설계하며 사용자 편의를 위한 분석을 진행하였다. 분석 및 설계 결과를 바탕으로 CHSM기반의 공정제어 소프트웨어 모듈을 구현하고, 구현된 CHSM기반 모듈을 RTOS 미들웨어 확장 컴포넌트화 구현한다.

2. 실시간 운영체제 다루기

1. Octavo 보드에 RTOS(RTLinux) 포팅을 위한 SD Card 제작

1) ARM Cross Compiler: GCC 설치

먼저 Octavo 보드는 ARM기반 보드이므로 컴파일 할 때 ARM보드에서 읽을 수 있게 컴파일 해야 한다. 하지만 ARM기반 보드에서 컴파일 하기에 무거운 코드가 있을 수 있으므로 이기종 플랫폼에서 타겟 플랫폼에 맞는 실행파일을 만들어 주는 **크로스 컴파일러**를 설치하여 컴파일 해주어야 합니다. 해당 사용 이기종 플랫폼은 INTEL기반 PC이며, 타겟 플랫폼은 ARM기반 보드(Octavo)입니다. 다운로드 및 컴파일러 경로 설정 방법은 아래와 같습니다.

wget -c <https://releases.linaro.org/components/toolchain/binaries/5.3-2016.05/arm-linux-gnueabihf/gcc-linaro-5.3.1-2016.05-x86_64_arm-linux-gnueabihf.tar.xz>

tar xf gcc-linaro-5.3.1-2016.05-x86\_64\_arm-linux-gnueabihf.tar.xz

export CC=`pwd`/gcc-linaro-5.3.1-2016.05-x86\_64\_arm-linux-gnueabihf/bin/arm-linux-gnueabihf-

설치 후 확인하여 정상설치가 되었는지 확인합니다.

2) 부트로더(U-Boot) 다운로드 및 컴파일(빌드)

부드로더란 운영체제가 시동되기 전 실행되는 최초의 시스템 프로그램으로, 하드웨어 초기화 및 검사들의 기초적인 작업으로 커널이 올바르게 실행되기 위한 작업을 하는 프로그램이다. U-Boot는 임베디드 시스템에서 가장 많이 쓰이는 부트로더이다.

Git clone <https://github.com/u-boot/u-boot>

명령어로 다운 받을 수 있다. 다음 과정은 u-boot폴더로 이동하여 Patch를 설치하는 과정이다.

wget -c <https://rcn-ee.com/repos/git/u-boot-patches/v2016.09-rc2/0001-am335x_evm-uEnv.txt-bootz-n-fixes.patch>

patch -p1 < 0001-am335x\_evm-uEnv.txt-bootz-n-fixes.patch

다음 과정으로는 빌드 설정 및 빌드하는 과정이다.

make ARCH=arm CROSS\_COMPILE=${CC} distclean

make ARCH=arm CROSS\_COMPILE=${CC} am335x\_evm\_defconfig

make ARCH=arm CROSS\_COMPILE=${CC}

위 과정을 통하여 커널이 완료되면 u-boot폴더 안에 MLO와 u-boot.img가 생성된다. 각각을 first 부트로더, second 부트로더라고 하는데 초기화 안된 초기 하드웨어의 가용할 수 있는 rom용량이 작아서 first 부트로더가 second 부트로더를 올릴 수 있게 최소한의 하드웨어 초기화를 하고 초기화가 된 ram에 second 부트로더를 올려 나머지 초기화를 진행합니다.

3) 리눅스 커널 컴파일

해당 과정은 Kernel, Module, Device Tree Binaries를 빌드하고 Deploy 폴더에 카피하는 일을 하는 과정이다. 먼저 커널을 다운로드 한다.

git clone <https://github.com/RobertCNelson//bb-kernel>

cd bb-kernel/

다음으로 어떤 커널을 빌드 할 것인지 정하는 checkout 과정을 거친다. 이때 실시간 운영체제인 RTOS 커널(am33x-rt-4.19)(Longterm 4.19.x + Real-Time Linux)을 가지고 와야 한다. 커널 checkout은 아래와 같다.

git checkout origin/am33x-v4.19 -b tmp

다음으로는 커널을 빌드하는 것이다. 커널을 빌드하기 위해서는 일반적으로 타겟 보드에 맞는 커널 패치와 커널 설정(make menuconfig)을 해야 한다. 하지만 build\_kernel.sh란 스크립트에서 자동적으로 진행하므로 커널 설정을 넘겨도 된다. 해당 작업이 완료되면 kernel 폴더가 생성이 된다.

./build\_kernel.sh

커널을 컴파일 하기 전에 다음 패키지를 install 해야 한다.

sudo apt install lzop

sudo apt-get update

sudo apt-get install lzma pkg-config libmpc-dev u-boot-tools

4) 루트 파일 시스템 설치

커널과 인터페이스를 할 수 있는 쉘과 같은 프로그램을 포함하는 파일 시스템이 있어야 운영체제 로서의 활용이 가능하다. 파일 시스템은 데비안 및 우분투, Angstrom, Arago 계열이 있으며, 여기에서는 데비안을 사용한다.

wget -c <https://rcn-ee.com/rootfs/eewiki/minfs/debian-10.4-minimal-armhf-2020-05-10.tar.xz>

sha256sum debian-10.4-minimal-armhf-2020-05-10.tar.xz

cd598e42850cbef87602bf15ee343abfbf0d8c6ba81028c741672b5f24263534 debian-10.4 minimal-armhf-2020-05-10.tar.xz

tar xf debian-10.4-minimal-armhf-2020-05-10.tar.xz

여기까지 SD Card에 부트로더, 리눅스커널, 루트파일 시스템을 넣을 준비를 완료하였다.

5) MicroSD 카드에 설치

위 과정이 모두 끝나면 부트로더와 커널 그리고 루트 파일 시스템을 설치해야 한다. 보통 microSD 카드를 카드 리더기에 연결하면 장치면은 /dev/sd(x)로 나온다. 이는 lsblk로 확인 가능하다. 설치하기 위해 microSD 카드의 이름을 확인 후 다음과 같이 환경 변수를 설정한다.

export DISK=/dev/sdb

다음으로 microSD 카드를 다음 명령어로 파티션을 지우고 포맷한다.

Sudo dd if=/dev/zero of=${DISK} bc=1M count=10

부트로더를 설치한다.

sudo dd if=./u-boot/MLO of=${DISK} count=1 seek=1 bs=128k

sudo dd if=./u-boot/u-boot.img of=${DISK} count=2 seek=1 bs=384k

U-Boot 폴더안에서 MLO와 u-boot.img 파일의 크기를 보면 각각 92.6kb 와 476.5kb이다. 따라서 첫번째 행을 실행하면 그 결과로, u-boot 폴더에 있는 MLO 파일이 이전에 환경 변수를 설정한 DISK위치 즉 MicroSD에 128k 크기만큼 복사를 한다(MLO 파일이 92.6kB이므로 충분히 커버가 된다. 두번째 행의 명령어를 하기와 같이 입력한다면, 결과로, u-boot 폴더에 있는 u-boot.img 파일이 위에 설명했던 내용과 동일하게 MicroSD에 384k\*2 크기만큼 복사를 한다.

다음으로는 파티션을 생성해야 한다. 부팅로더가 들어간 부분과 그렇지 않은 부분에 대해서 파티션을 생성해야 하는데, util-linux 버전 별로 상이하다고 한다.따라서 다음과 같이 먼저 버전을 확인할 필요가 있다.

sudo sfdisk --unit M ${DISK} <<-\_\_EOF\_\_

4,,L,\*

\_\_EOF\_\_

다음은 파티션 포맷이다.

sudo mkfs.ext4 -L rootfs ${DISK}1

MicroSD 카드를 사용하기 위해서는 임의의 폴더를 만들고, 그곳에 마운트를 시켜야 지만 MicroSD카드를 이용 가능하다. 따라서, 여기서는 /media/rootfs/라는 폴더를 만든 다음, SDcard에 mount한다.

sudo mkdir -p /media/rootfs/

sudo mount ${DISK}1 /media/rootfs/

지금까지 SD Card에 부트로더를 올린 후 파티션을 나누고 포맷을 진행 그리고 임의의 디렉토리에 마운트를 진행하였다. 이제 커널과 루트 파일 시스템을 설치하는 과정이다. 우선 커널 버전에 대한 환경변수를 설정을 해야 하는데, 기존 리눅스 소스를 다운받은 bb-kernel 폴더에서 kenel\_version 파일을 통해 커널 버전을 다음과 같이 확인이 가능하다. 커널 버전을 확인 후 환경변수를 설정한다.

export kernel\_version={커널 버젼}

루트 파일 시스템을 복사하는 과정은 다음과 같다.

sudo tar xfvp ./\*-\*-\*-armhf-\*/armhf-rootfs-\*.tar -C /media/rootfs/

sync

sudo chown root:root /media/rootfs/

sudo chmod 755 /media/rootfs/

첫번째 명령어는 기존에 우리가 "ubuntu-180.04.2-minimal-armhf-2019-02-16" 폴더에 다운로드한 우분투 파일시스템 압축 파일을, MicroSD 카드를 마운트한 /media/rootfs/폴더에 압축해제 하는 것이다.

- sync: 메모리에 저장되어 있는 작업내용을 디스크에 저장을 한다.

- sudo chown root:root /media/rootfs/ : /media/rootfs/ 내부의 owner/group을 root로 바꾼다.

다음으로는 부트 이미지의 이름을 설정한다. uEnv.txt 파일에서 uname\_r을 설정을 한다

sudo sh -c "echo 'uname\_r=${kernel\_version}' >> /media/rootfs/boot/uEnv.txt"

기조에 linux kernel을 컴파일 한 zImage 파일을 microSD 카드에 복사를 하는 것이다.

sudo cp -v ./bb-kernel/deploy/${kernel\_version}.zImage /media/rootfs/boot/vmlinuz-${kernel\_version}

하기와 같이 MicroSD 카드에 폴더를 만들고 기존에 다운로드했던 bb-kernel의 폴더 안의 deploy 폴더에서 디바이스 트리 바이너리 압축파일을 MicroSD 카드에 새로 만든 폴더에 압축을 푼다.

sudo mkdir -p /media/rootfs/boot/dtbs/${kernel\_version}/

sudo tar xfv ./bb-kernel/deploy/${kernel\_version}-dtbs.tar.gz -C /media/rootfs/boot/dtbs/${kernel\_version}/

ARM에서 디바이스 트리 바이너리는 x86 계열에서의 BIOS라고 보면 된다. 부트로더가 커널을 시작시키기 전에 디바이스 트리 바이너리(DTB)를 메모리 영역에 Copy를 한 후 커널을 시작한다고 한다.

bb-kernel 폴더 내의 deploy 폴더에 기존에 다운로드 받았던 modules을 MicroSD카드에 압축을 해제한다.

sudo tar xfv ./bb-kernel/deploy/${kernel\_version}-modules.tar.gz -C /media/rootfs/

위의 명령어를 사용하여 파일시스템 테이블 정보를 생성한다.

sudo sh -c "echo '/dev/mmcblk0p1 / auto errors=remount-ro 0 1' >> /media/rootfs/etc/fstab"

네트워크 정보를 생성한다.

sudo nano /media/rootfs/etc/network/interfaces

auto lo

iface lo inet loopback

auto eth0

iface eth0 inet dhcp

microSD Card를 제거한다

sync

sudo umount /media/rootfs

2. RTOS일때와 NON-RTOS일떄의 속도차이 실험

1) RTOS이란 무엇인가?

실시간 운영체제 또는 RTOS(Real-Time Operating System는 실시간 응용 프로그램을 위해 개발된 운영체제이다. 운영체제의 기능 중 CPU 시간 관리 부분에 초점을 맞추어 설계되었다. 프로그래머가 프로세스 우선 순위에 더 많은 제어를 할 수 있으므로 응용 프로그램의 우선 순위가 시스템 프로그램의 우선 순위를 보다 더 높은 권한을 가질 수 있다. 따라서 응용 프로그램의 처리 요청을 정해진 시간 안에 처리해 줄 수 있다.

실시간 운영체제의 핵심은 응용 프로그램의 **태스크(Task)** 처리에 걸리는 시간을 일관되게 유지할 수 있는 정도에 있다. 실시간 운영체제는 2가지로 분류하는데, Hard Real-Time과 Soft Real-Time으로 나뉜다. 먼저 **Hard Real-Time**은 어떤 작업을 일정 시간 안에 반드시 처리해야 하며, 그 시간이 지난 후의 결과 값은 정확해도 소용이 없는 경우 사용하며, 군사장비 및 비행기와 같은 정밀한 장비에 사용이 된다. 이와 반대로 **Soft Real-Time**은 어떤 시간 안에 처리하면 좋지만 그렇지 못한 경우에 그 시간에서 약간 경과한 후의 값도 인정이 된다. RTOS의 주된 설계 목표는 높은 [처리율](https://ko.wikipedia.org/wiki/%EC%B2%98%EB%A6%AC%EC%9C%A8)(throughput)이 아니라, [실시간 성능 보장](https://ko.wikipedia.org/wiki/%EC%8B%A4%EC%8B%9C%EA%B0%84_%EC%BB%B4%ED%93%A8%ED%8C%85)에 있다. 실시간 시스템의 데드라인을 대체로 맞추는 RTOS를 **Hard Real-Time**라 하고, 데드라인을 [결정론적 알고리즘](https://ko.wikipedia.org/wiki/%EA%B2%B0%EC%A0%95%EB%A1%A0%EC%A0%81_%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98)(deterministic algorithm)에 의해 만족하는 경우를 **Hard Real-Time**라 한다

RTOS의 설계 방식은 2가지로 나뉜다. 먼저 이벤트 구동 방식(Event-Driven)은 우선 순위 기반 스케줄링 또는 선점형(Preemptive) 스케줄링이라고 부른다. 태스크 전환이 현재 수행중인 태스크 보다 높은 우선 순위를 갖는 이벤트가 발생시 Context Switching이 발생한다. 다음으로는 시분할(Time-Sharing) 스케줄링 방식이다. 이는 클럭 인터럽트나 라운드 로빈과 같은 주기적인 이벤트가 발생할 때 태스크의 전환이 일어난다. 따라서 시분할(Time-Sharing)이 더 자주 태스크가 바뀌는 Context Switching이 발생한다. 하지만 이러한 점으로 인하여 자연스럽고, 예측하기 쉬운 멀티테스킹을 제공하며, 하나의 프로세스나 한 명의 사용자가 장치를 독점적으로 사용하는 것과 같은 효과를 제공한다. 이 때 사용하는 스케줄링 알고리즘은 다음과 같다.

* 협력형 스케줄링 (cooperative scheduling)
* 선점형 스케줄링 (pre-emptive scheduling)
  + 라운드 로빈 스케줄링 (Round-robin scheduling)
  + 고정 우선순위 선점형 스케줄링 (fixed priority pre-emptive scheduling)

2) CYCLIC TEST 실험

Cyclic Test는 실시간 운영체제의 성능을 측정하는 벤치마크 프로그램이다. 우리가 사용하는 RTLinux는 프로세스간 우선순위를 두어 우선순위가 높은 프로세스에게 스케줄링 상에서 실행 우선권을 점유할 수 있도록 하는 것뿐만 아니라 시스템 콜을 처리할 때에도 처리 선점권을 기짐으로써 더욱 실시간을 보장할 수 있도록 패치 된 리눅스이다. 우리가 사용할 벤치마크는 **"RT 리눅스 또는 운영체제가 얼마나 RT를 제대로 보장해주고 있느냐"**를 측정할 수 있도록 도와준다.

Cyclic Test을 다운받아 설치하여 실행한다.

git clone git://git.kernel.org/pub/scm/utils/rt-tests/rt-tests.git

cd rt-tests

git checkout stable/v1.0

make all

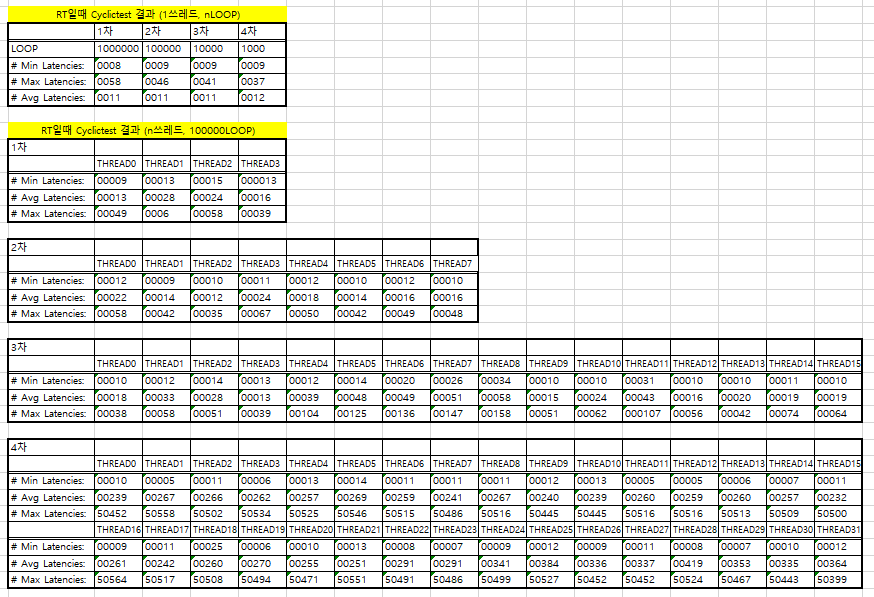
make install

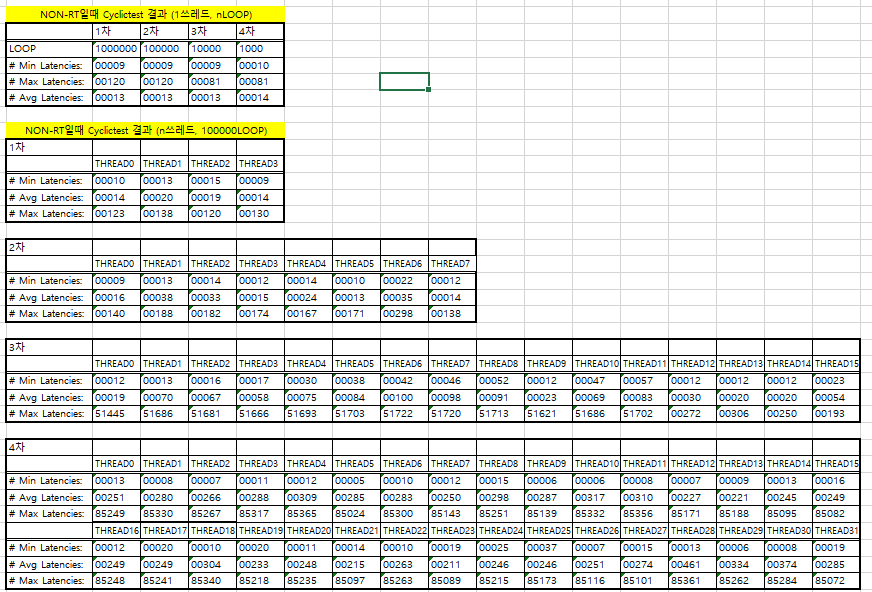
-l 옵션은 루프의 회수, -t 옵션은 실행될 쓰레드의 개수, -p는 쓰레드의 우선순위(RT 기준)을 나타낸다. Cyclic Test 프로세스는 -t 옵션에 명시된 개수만큼의 LWP(Light Weight Process)를 생성한다.

./cyclictest -l100000 -m -n -a0 -t1 -p99 -i400 -h400

3) CYCLIC TEST 결과

실험을 통해서 Non-RT일때와 RT일때의 Latency(지연율)을 비교해 보겠다. Latency는 수 십만번의 벤치마크 루프 중 가장 Latency가 컸을 때를 의미하고, Min Latency는 가장 짧았던 Latency, Avg는 평균을 의미한다. 사실 Real-time이라는 건 시스템 부하가 없는 상황에서는 크게 의미가 없다고 볼 수 있다. 왜냐하면 CPU가 Idle 한 상태에서는 무엇을 처리해도 빨리빨리 수행할 것이고, 이러한 상황에서는 RT 리눅스가 의도하는 바인 Pre-emptive의 효과는 크게 두드러지지 않을 것이기 때문이다. 따라서 각 테스트 케이스에서 RT 벤치마크 결과의 차이점을 뚜렷하게 확인하기 위해서는 시스템에 의도적인 부하를 가해야 할 필요가 있다. 부하를 인위적으로 주는 방법은 여러가지가 있겠지만, Cyclic Test의 쓰레드 개수를 늘리는 방법을 사용할 것이다.



아래는 Non-RT일때의 결과이다.

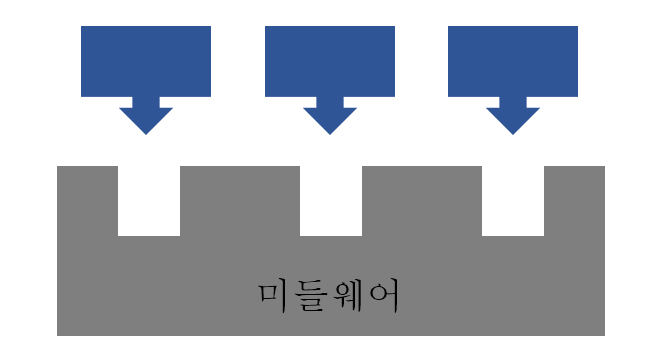
이를 바탕으로 결과를 도출해 낸다면, RT 4차 실험(100000루프 32쓰레드)일 때의 결과와 Non-RT 3차 실험 (100000루프, 16쓰레드)일 때 Max Latency에서 비슷한 결과가 나오게 되었다. 따라서 다수의 쓰레드를 사용하기에는 RT일 때 더욱 빠르다는 것을 알게 되었다.

3. 미들웨어 아키텍처 개발

1. RTOS기반 미들웨어 아키텍처

1) 미들웨어 아키텍처 구조 학습.

미들웨어 아키텍처 구조는 JSON을 사용하여 작동한다. 이미 Framework가 개발되어 있었고 자세히는 모른다. 다만 아래와 같은 구조를 가지고 있다.



다음과 같이 미들웨어에 컴포넌트들이 장착되어 각각의 컴포넌트들은 독립성을 유지한다. 모든 데이터는 미들웨어를 통하여 통신이 되는 구조이다. 이 컴포넌트들은 json파일을 사용하여 구현하였다. json파일에 응용프로그램이 들어가게 되면 미들웨어에서 이 응용프로그램을 작동시키는 구조이다.

4. Concurrent Hierarchical Finite State Machine 실습

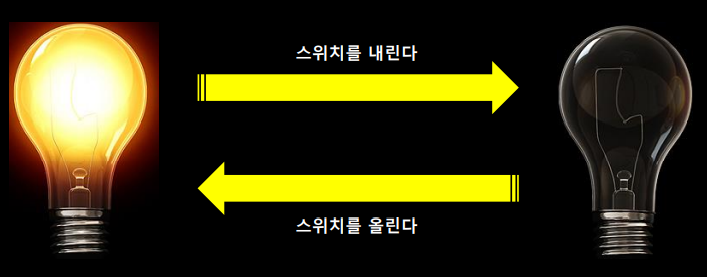
1. 유한상태기계(Finite State Machine)

1) 유한상태기계(Finite State Machine)

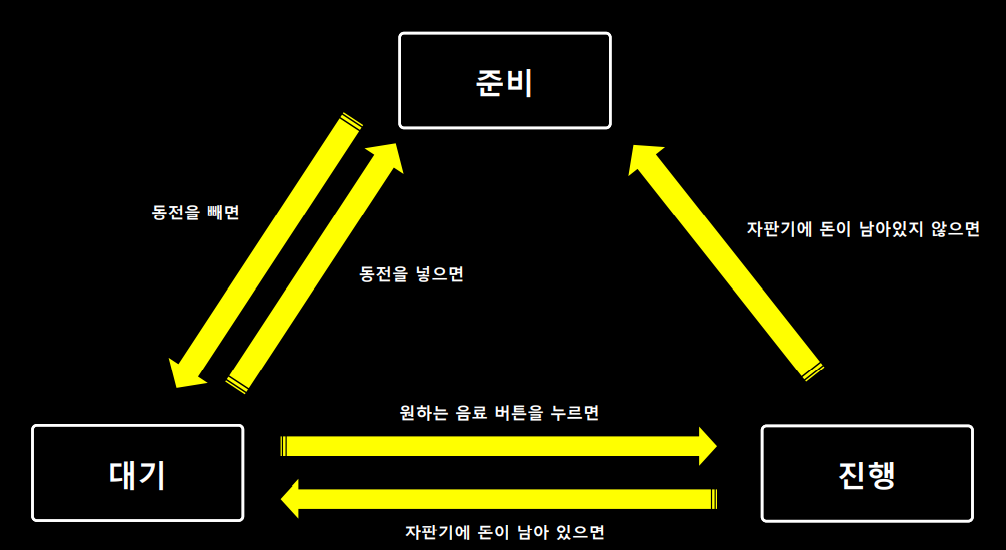
유한 상태 기계는 주어진 모든 시간에 처해있을 수 있는 유한개의 상태를 가지고 주어지는 입력에 따라 어떤 상태에서 다른 상태로 이동시키거나 출력 또는 액션이 일어나게 하는 장치 또는 그런 장치를 나타낸 모델이다. 유한상태기계 바탕에 깔린 아이디어는 객체의 행동을 쉽게 처리할 수 있는 **덩어리** 또는 **상태**들로 분해하는 것이다.

유한상태기계의 장점으로는 첫째, 빠르고 코딩하기가 쉽다는 점이다. 또한 행동을 쉽게 관리할 수 있는 덩어리로 분할하기 때문에 이상 행동이 발견될 시 각 상태의 추적 코드를 보면 됨으로 오류수정이 용이하다. 다음으로는 새로운 상태들과 규칙을 추가함으로써 행동의 범위를 간단하게 확장 시킬 수 있다. 마지막 장점으로 if, this, then, that과 같은 사고 절차를 사용하지 않고 본질적으로 코드 화된 규칙들을 따르므로 프로세서의 귀중한 시간을 빼앗지 않는다.

유한상태기계의 예시로서 전등의 스위치를 들 수 있다. 전등은 ON, OFF 2가지의 상태를 가지고 있다. 스위치를 올리면 꺼짐에서 커짐으로 상태가 이동하며, 내리면 커짐에서 꺼짐으로 상태가 전환된다. 아래 그림을 참고해 보자.



전등의 예시를 이렇게 표현할 수 있다. 그리고 유한상태기계를 구현하기 위해서는 구현 전에 State Chart를 먼저 구현하는 것이 좋다. State Chart는 어떤 상태에서 무슨 이벤트로 인하여 다른 상태로 넘어가는지를 그림으로 그린 것이다. 아래는 자판기의 State Chart이다.



그림에서 나타내 듯이 네모 칸은 상태(State)를 나타내고 있고 화살표는 상태에서 다른 상태로 갈수 있는 길을 나타낸다. 마지막으로 화살표 옆에 있는 내용은 이벤트가 발생되는 원인이며, 이 원인이 발생하면 상태가 변하게 된다.

하지만 유한상태기계에도 문제점이 있는데 먼저 계층화가 어렵다. 또한 다시 돌아올 때 갔던 길을 다시 되돌아와야 한다. 마지막으로 1차원 적으로 증가하여 functional commonalty와 logical group을 파괴한다. 따라서 이러한 단점을 보완하기 위해서 Concurrent Hierarchical Finite State Machine이 등장하게 되었다.

2 Concurrent Hierarchical Finite State Machine (CHSM)

1) Concurrent Hierarchical Finite State Machine (CHSM)

Concurrent Hierarchical Finite State Machine (이하 CHSM)은 논리적 계층 구조나 고립의 개념이 없기 때문에 STD는 "Flat"과 "Global"이고, 전환이 많은 경우에 전환은 많은 주(州)에 걸쳐 재현되어야 한다. 따라서 기계가 초기 상태로 돌아가는 일종의 "고급 인터럽트"를 가지고 있어야 하며, 시스템이 선형적으로 성장함에 따라 상태 수가 기하급수적으로 증가하는 경우가 많다는 FSM의 많은 문제점을 보완하기 위해 개발된 언어이다.

CHSM의 State Chart를 정의하면 5개(S, A, T, C, E)의 구성요소로 구성되어 있다.

S: 모든 상태의 집합(Set)을 나타낸다.

A: 모든 Active 상태의 집합이다.

T: 상태 사이의 모든 Transition의 집합이다.

C: 관련 조건이 참은 모든 Transition의 집합이다.

E: 이벤트의 집합이다.

상태가 2개이상의 transition이 있을 때, 하나의 transition이 임의로 선택된다. 이때 부모 상태와 자식 상태 모두 같은 경우 부모 상태가 선택이 된다. 또한 자식 상태에 들어갈 때는 부모 상태로 먼저 들어가야 하며 부모 상태를 나올 때는 자식 상태를 먼저 나와야 한다. 외부적 또는 내부적으로 생성된 이벤트는 transition할 때 발생하지 않아야 한다.

이벤트 발생 시 일련의 transition을 수행하기 위한 알고리즘이 제공된다. 객체 지향적인 패러다임에 따라서 3가지 버전의 enter()와 exit()함수가 있다. ->표기와 this 명령어는 c++에서 빌려왔다. -> 표기는 외쪽의 객체에서 오른쪽 객체를 함수에 적용하는 역할을 한다. This는 현재의 객체를 가리키는 명령어이다. Enter()와 Exit()함수는 Boolean값을 반환한다. True는 함수에 대한 호출의 결과로 상태가 입력 또는 종료되었음을 나타낸다. False는 이미 활성 또는 비활성 상태였음을 나타낸다. 이것은 상태가 두 번 이상 활성화되거나 비활성화되는 것과 관련된 단계를 수행하지 못하도록 하기 위해 사용된다. 매개변수는 생략된다. Exit() 함수의 매개변수는 대상 상태, 즉 현재 상태가 종료되는 상태를 말한다.

2) CHSM Language System

CHSM은 선언문, CHSM 설명, 사용자 코드 섹션의 3개의 섹션으로 구성된 일반 텍스트 파일이다. 섹션은 %% 문자로 구분된다. 사용자 코드 섹션을 생략할 경우 후행 %%도 생략할 수 있다.

Declarations

%%

Description

%%

User code

Description은 보통 c++ 소스코드에 통합되기 때문에 declaration 섹션에는 글로벌 변수, 상수 선언 등 후속 섹션의 c++ 코드 조각 내에서 사용하고 하는 필요하거나 유용한 선언과 전처리기 지침이 포함되어야 한다. 또한 description 섹션에는 State Chart를 명령어로 바꾼 내용도 들어가게 된다.

상태는 state-description에 따라 지정된다. state-description은 set, cluster, state의 키워드 중에 하나로 구성이 된다. Description은 상태가 어떤 상태인지 cluster인지 set인지 state인지에 따라 달라지게 된다. Cluster와 set 설명은 더 많은 정보가 제공되어야 한다는 점을 제외하고 일반 상태 설명과 같다. 먼저 이름을 선언하고 자신이 가지고 있는 자식 상태를 괄호 안에 넣어야 한다. Cluster와 set도 자신으로 다시 이동하는 transition을 가질 수 있다. 이 경우에는 transition 앞에 history 키워드를 사용하여 작성하여야 한다.

cluster x(a,b,c) history {

delta->d %{ f(); %};

} is {

state a {

alpha,epsilon->c;

beta->b;

}

state b { gamma(vc; }

state c;

}

다음으로는 transition을 정의하는 방법을 보겠다. Transition에는 여러가지 옵션이 있다. 가장 간단한 transition은 어떤 사건의 발생의 결과로 한 상태가 다른 상태로 전환되는 것을 명시한다. 이는 아래와 같이 표기한다.

beta->b;

이는 베타 사건으로 인하여 b상태로 바뀌는 것을 의미한다. 만약 다른 조건이지만 같은 상태로 이동할 때는 아래와 같이 표기한다. 이는 알파, 엡실론 transition이 발생할 때 c 상태로 이동한다는 의미이다.

alpha,epsilon->c;

이벤트의 transition이 조건에 따라 발생할 때는 transition 이름 뒤에 조건을 붙인다. 사용법은 아래와 같다. V라는 변수가 4보다 작으면 감마 transition이 발생하고 c 상태로 이동한다는 것이다. 조건에 사용되는 표현식은 값을 반환하는 임의의 유효한 c++ 함수일 수도 있다. 또한 CHSM의 내장형 함수 $in()으로 조건문을 나타낼 수도 있다.

gamma(v<4)->c;.

{ alpha($in(c)->b; }.

만약 한 transition으로 인하여 다른 transition이 broadcast된다면 이 상태를 다음과 같이 나타낸다.

beta->b/epsilon;

마지막으로 %{ (c++ 문법의 코드) %}를 사용해 조건문이나 이벤트를 c++로 작성할 수 있다.

이벤트 이름을 미리 선언할 필요가 없기 때문에 이벤트 이름을 지정하기 위해 특별히 할 필요가 없다. 이벤트는 CHSM 문법 기준에 따라 동일한 이름의 이벤트를 금하고 있다. 앞서 암묵적으로 broadcast 되는 Enter(), Exit() 이벤트를 보았다. 그러한 이벤트는 transition을 위한 이벤트로 사용될 수 있다. 아래는 예시이다.

state a {

alpha,enter(b)->c %{

if ( event == $enter(b) )

// do something only when enter(b) caused the transition

%};

}

또한 메크로 이벤트도 가능하다. 이는 이벤트를 하나의 이름으로 정의하여 필요할 때 찾아 쓰는 역할을 한다.

event coin is { nickel, dime, quarter }

state Idle { coin->Collect; }

3) CHSM 예제 학습

CHSM의 주어진 예제 중 Microwave 예제를 바탕으로 어떻게 진행이 되는지 공부하였다. 먼저 microwave.chsmc 코드를 보면 앞서 말한 것과 같이 %% 키워드로 분기가 나눠져 있다. 이 텍스트를 chsmc명령어로 컴파일 하게 되면 cpp파일이 결과로서 나오게 된다.

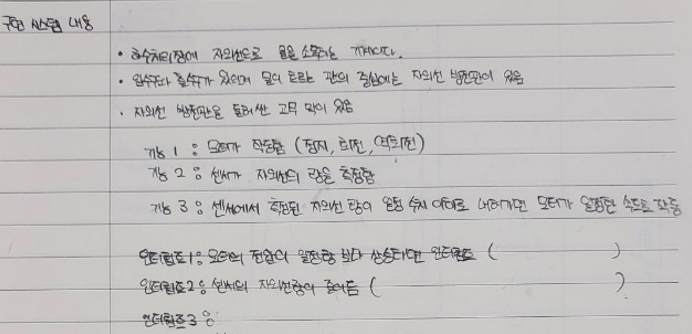
텍스트이(가) 표시된 사진

자동 생성된 설명다음은 microwave의 State Chart이다. 이를 바탕으로 State Chart 작성법 및 코드 작성법을 배우게 되었다. 또한 cpp 파일의 작성법도 알게 되었는데. 이를 통하여 다음에 설명할 실제로 사용자 요구 및 구현하고자 하는 목표가 무엇이며, 사용자 요구사항과 기본적인 입력 값이 무엇이며 출력은 무엇인지의 대해서 분석하여 State Chart와 cpp파일을 구현하는 실습과정을 진행하였다.

5. FSM기반 공정제어 시스템 구현

1 FSM기반 공정제어 시스템 구현

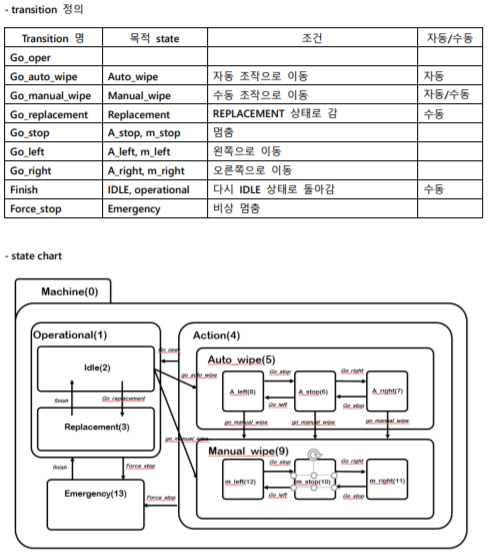
1) 사용자 요구 분석 및 State Chart 제작

공정제어 시스템의 요구사항 및 분석이 예정 시간보다 늦게 전달되어 먼저 그때까지 정해진 내용으로 시스템을 구현하였다. 먼저 모든 내용이 확정되기전 확정된 내용을 바탕으로 내용 분석 및 사용자 요구를 분석하고 State Chart와 Data Sheet를 만들게 되었다. 다음은 구현 시스템을 분석한 뒤 정리한 것이다.

다음은 초기 State Chart이며 수정을 반복하여 최종 State Chart를 도출해 내는 과정이다.

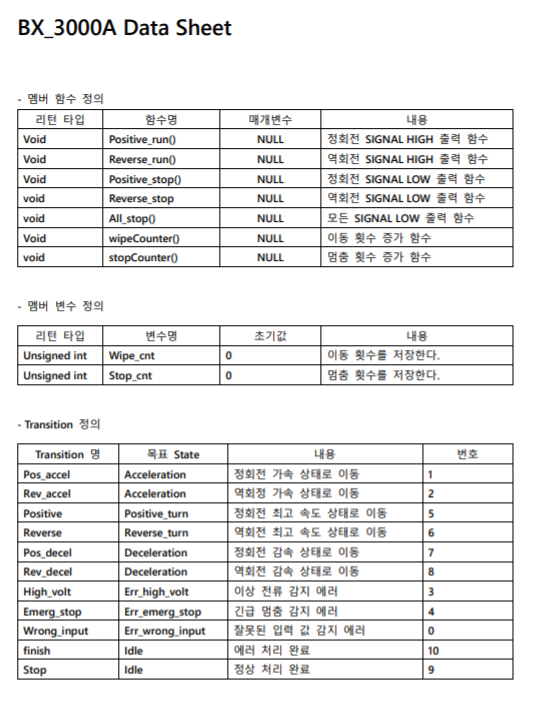
|  |  |
| --- | --- |
| 제1차 아이디어 | |
|  | |
| 문제점 | CHSM의 State Chart에 대하여 이해 부족으로 인한 수정이 불가피함 |
| 제2차 아이디어 | |
|  | |
| 문제점 | 사용자 입력이 필요하며 사용자 입력에는 전원, 강제 종료, 정방향, 역방향 이동이 부재 및 추가 필수. |
| 제3차 아이디어 | |
|  | |
| 문제점 | 모터의 세분화, Emergency State 제작, 명료한 네이밍 |
| 제6차 아이디어 | |
|  | |
| 문제점 | 회전수에 따라 이동위치 파악, 속도에 관한 내용이 들어가야함 |
| 제6-1차 아이디어 | |
|  | |
| 문제점 |  |

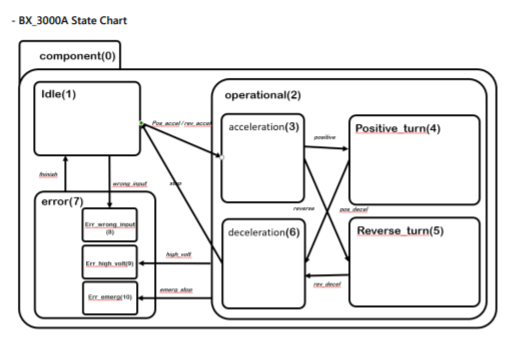
다음은 6-1 아이디어의 Data Sheet이다.



2) 최종 사용자 요구 분석 및 State Chart 제작

최종적으로 UV Cleaning 시스템의 취급 설명서가 전달되었다. 이를 바탕을 입력 신호와 출력신호, 모더가 어떻게 작동되며, 전압은 어떻게 판단하는지에 대한 내용을 알게 되었다. 따라서 이를 바탕으로 앞서 구현한 내용을 수정하여 다시 State Chart와 Data Sheet를 제작하였다.

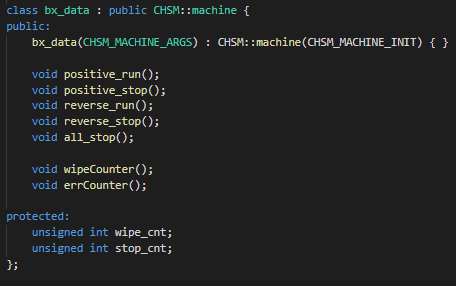




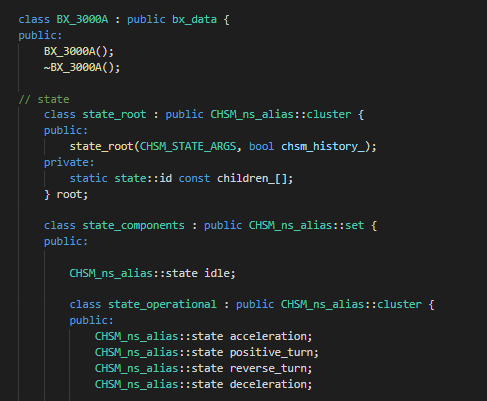
2 FSM기반 공정제어 시스템 코드 분석 및 시연

1) chsmc 컴파일 변환 cpp 코드 분석

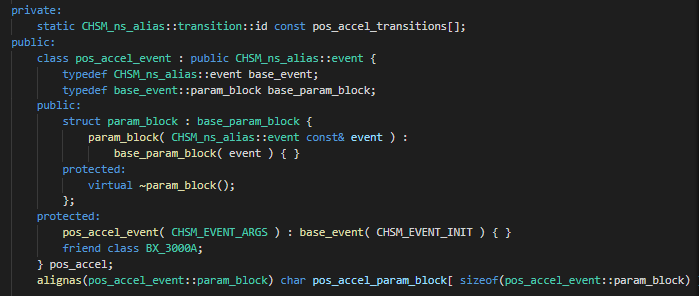
Chsmc파일과 cpp파일은 부록에 첨부하였다. 이제 이 코드를 하나씩 분석해 보겠다. 가장먼저 cpp파일에는 전처리기 파트가 나와야 하며, 이 프로그램에서 사용할 변수와, 함수에 대한 정의가 들어가는 클래스가 있어야 한다.



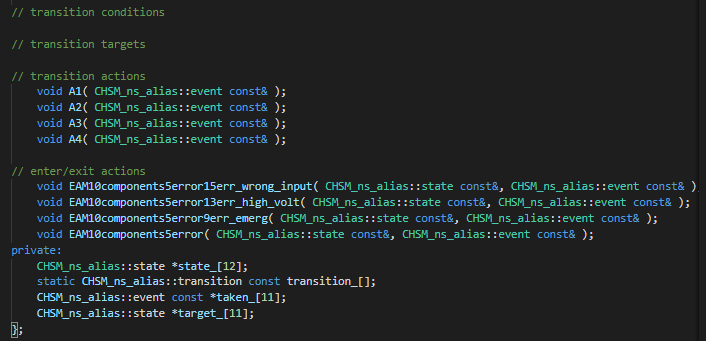
그 다음에 오는 부분은 chsmc 파일에서 %% 이하에 있는 description 부분을 클래스와 한 부분이 오게 된다. Set, ,cluster, state는 각각의 class를 가지며, set의 클래스 안에는 cluster클래스가 있고 그 안에는 state클래스가 있다.



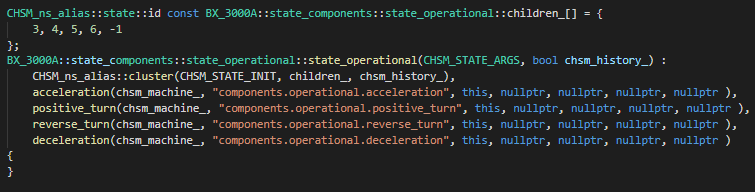
다음으로는 Transition과 Event에 대한 내용이 구조체화 되어 프로그램이 transition을 발생시켯을때 온전한 모든 transition의 내용이 전달되도록 하였다.



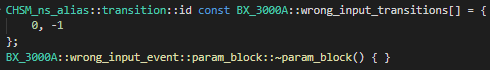
그 후 이 transition이 어떤 형태를 가지는 지 함수로 나누게 된다. 먼저 조건이 있는 transition, 액션이 있는 transition, Enter()와 Exit(), target이 있는 transition인지로 나뉘게 된다. 위에 3가지가 정의된 클래스를 하나 생성한다. 그 다음으로는 이제 클래스에 있는 내용을 정의하는 과정이다.

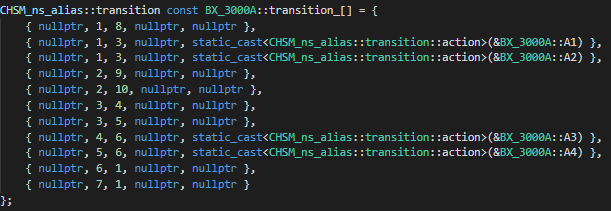


Set, cluster, state는 chsmc파일에서 선언한 순서대로 id가 붙는다. Transition도 같은 순서로 번호가 붙는다. 이를 바탕으로 다음으로는 state를 정의한다. 자식 배열에는 id 번호가 들어가게 도니다. 또한 생성자에는 그 함수의 위치와 enter(), exit()에 관한 정보가 들어가게 된다.



다음으로는 transition에 대한 내용이다. 앞서 각각의 transition의 번호가 들어가게 된다.



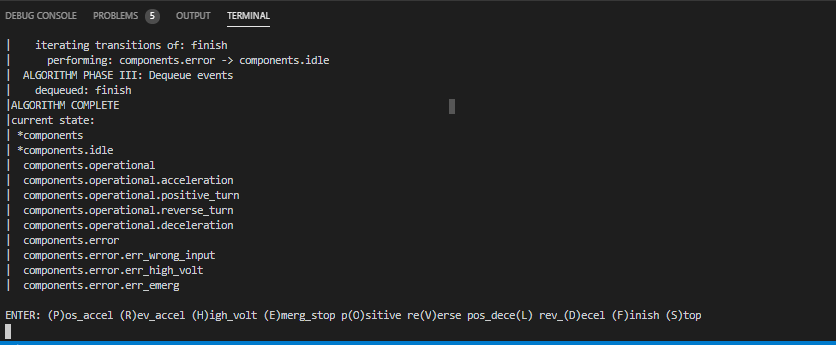
그런 후 전체 transition 배열에 transition이 몇 번 state에서 몇 번 state로 이동하는지에 대한 내용이 들어가게 된다. 이때 배열안에 들어가는 배열의 순서는 condition transition, departure, destination, target transition, action transition 순으로 들어가게 된다. 

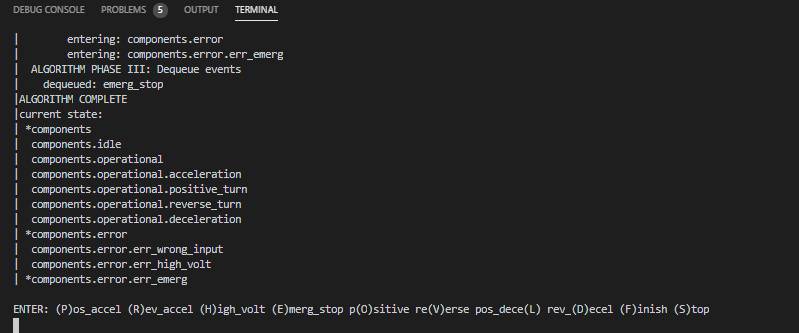
이후 이렇게 정의된 것을 가장 상위 클래스이 생성자에서 초기화 하게 된다. 이 코드는 아래와 같다.



이 다음으로는 chsmc파일과 마찬가지로 메인 함수가 오게 된다.

2) 최종 결과물

최종 결과물은 아래와 같다.

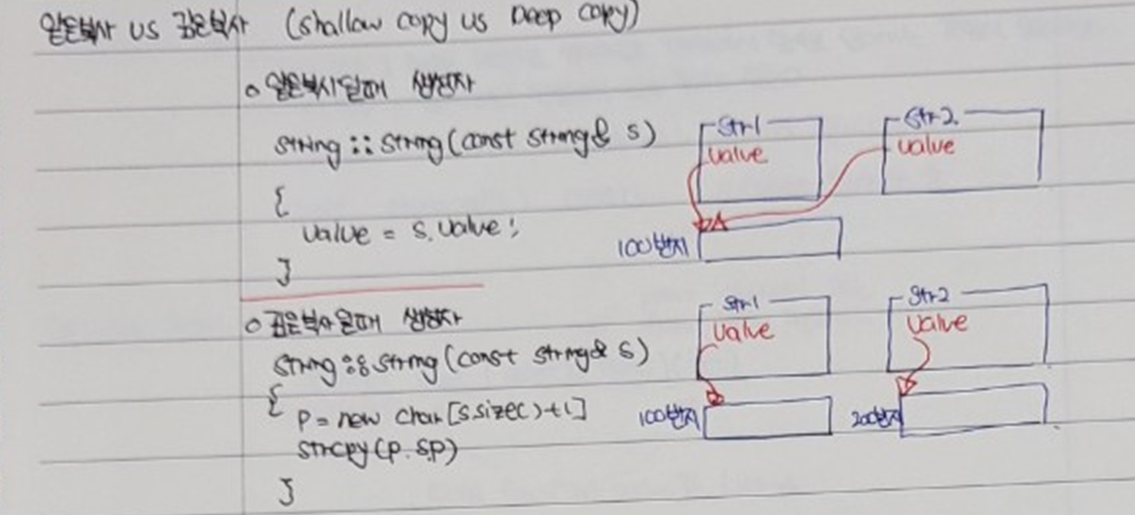


6. C++ 학습내용

1 기타 C++ 학습내용

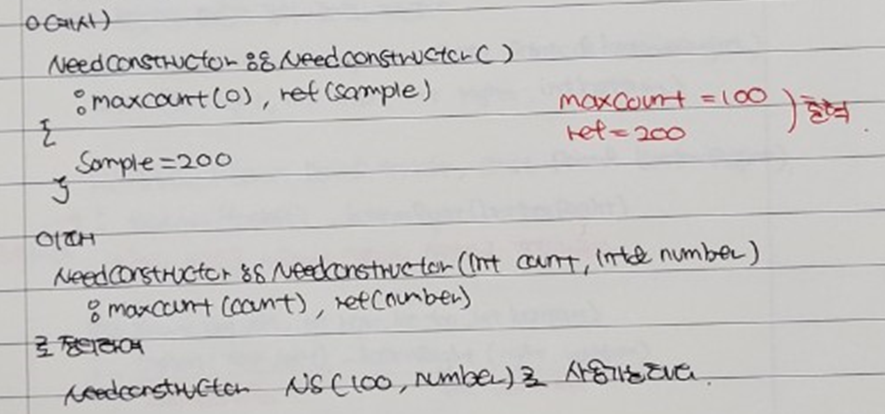
1. 얕은 복사 및 깊은 복사

깊은 복사와 얕은 복사의 차이점은 아래와 같다.



1. 클래스 멤버 변수 초기화 리스트

반드시 초기화해야 하는 멤버 변수들은 생성자의 초기화 리스트를 사용하여 초기화 한다.



1. 접근자

단순히 public 접근권한을 가진 멤버 함수로 외부에서 멤버변수에 접근하는 것을 도와주고 감시하는 역할을 한다.

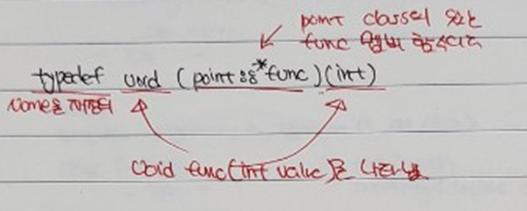
1. 정적 멤버의 기본

정적 멤버는 모든 객체가 공유하고 있는 멤버이다. 정적 멤버는 하나이기 때문에 모든 객체는 클래스내에서 모두 같이 사용한다.

1. 인라인 함수, const 함수

인라인 함수란 함수를 메인으로 인식하므로 메인에서 함수로 넘어가는 과정이 없어지는 함수이다. Const 함수는 멤버 변수를 변경하지 않는 함수를 뜻한다.

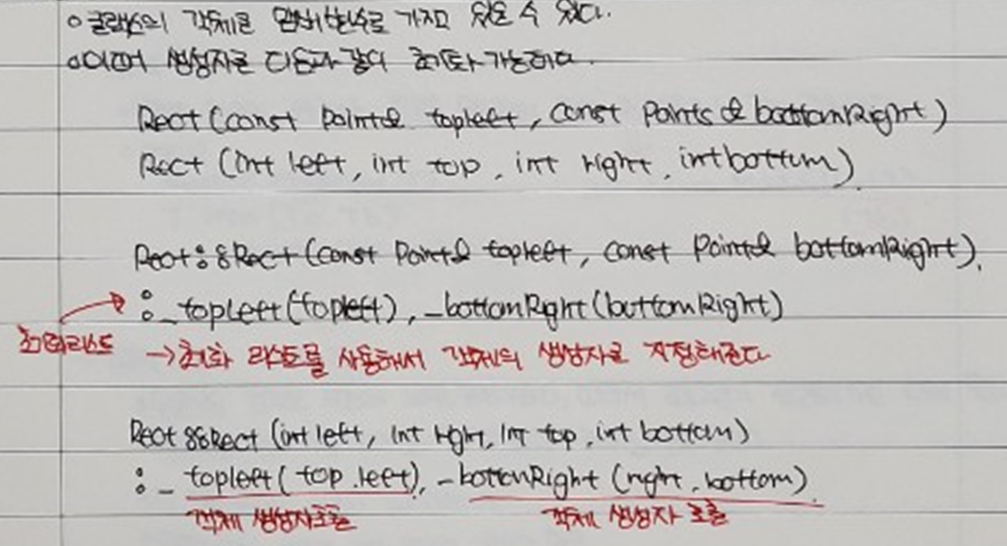
1. 멤버 함수 포인터



1. Friend class

Friend class로 선언이 되면 friend class로 선언된 다른 클래스의 private이나 protected 멤버에 접근이 가능해진다.

1. 객체를 멤버로 가지는 클래스

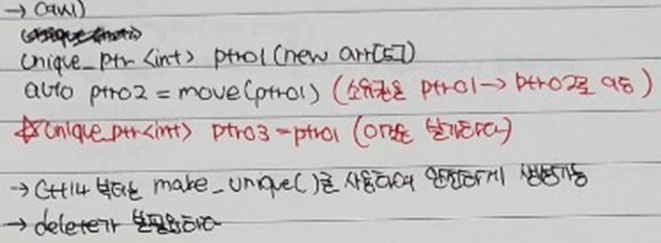


1. 스마트 포인터

C++에서는 메모리 누수로부터 프로그램의 안정성을 보장하기 위해 스마트 포인터를 제공한다. 스마트 포인터의 수명이 다하면 메모리를 해제한다. AUTO\_PTR이 있지만 C++11부터 사용하지 않는다.

- UNIQUE\_PTR

하나의 스마트 포인터 만이 특정 객체를 소유할 수 있도록 한다. (객체의 소유권 개념 도입)



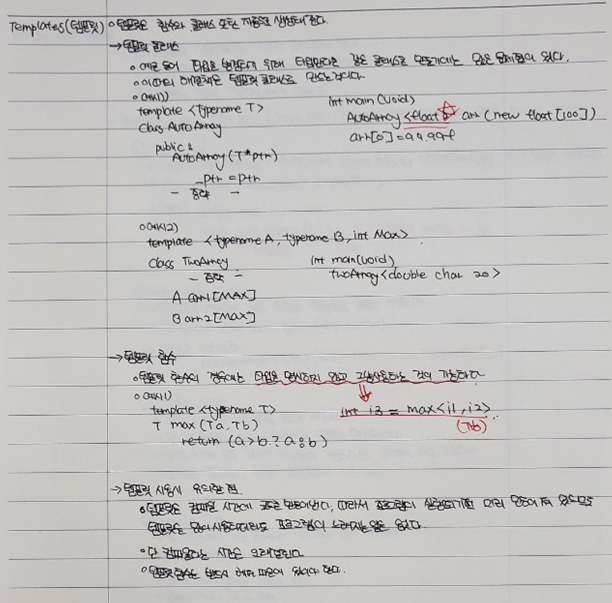
- SHARED\_PTR

하나의 특정 객체를 탐지/참조하는 스마트 포인터가 총 몇 개인지 참조하는 스마트 포인터이다.

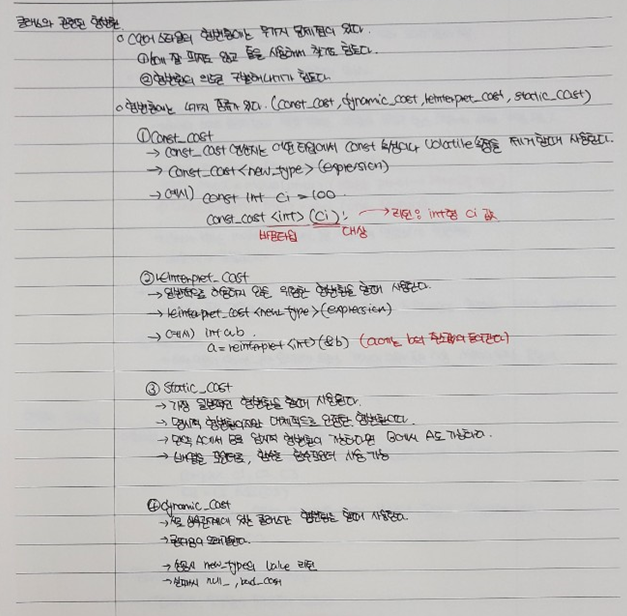
- WEAK\_PTR

하나 이상의 SHARED\_PTR 인스턴스가 소유하는 객체에 대한 접근 허용, 소유자의 수에는 포함하지 않는다.

1. 템플릿



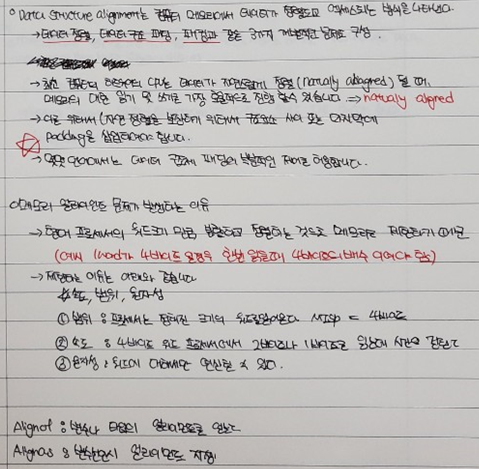
1. 클래스와 관련된 형변환



1. DELETE 와 DELETE[]의 차이점

DELETE는 단일객체에 대한 소멸자 호출과 메모리 반환을 한다. 반면에 DELETE[]은 배열객체에 대한 소멸자 호출과 메모리 반환을 한다.

1. 메모리 얼라이먼트



참고 사이트

// Octavo 보드에 RTOS(RTLinux) 포팅을 위한 SD Card 제작

<https://decdream08.tistory.com/35?category=358199>

<https://joyho.tistory.com/2>

<https://www.digikey.com/eewiki/display/linuxonarm/OSD3358-SM-RED>

// 디바이스 트리

<https://limit0.tistory.com/entry/%EB%94%94%EB%B0%94%EC%9D%B4%EC%8A%A4-%ED%8A%B8%EB%A6%AC%EC%97%90-%EB%8C%80%ED%95%9C-%EA%B0%84%EB%9E%B5%ED%95%9C-%EC%84%A4%EB%AA%85>

// RTOS / RTLINUX

<https://ko.wikipedia.org/wiki/%EC%8B%A4%EC%8B%9C%EA%B0%84_%EC%9A%B4%EC%98%81_%EC%B2%B4%EC%A0%9C>

<https://ppua.tistory.com/entry/%EC%8B%A4%EC%8B%9C%EA%B0%84-%EC%9A%B4%EC%98%81%EC%B2%B4%EC%A0%9C-RTOS%EC%9D%98-%EC%9D%B4%ED%95%B4#drawer>

<https://blog.naver.com/alice_k106/221149061940>

// Cyclic Test

<https://m.blog.naver.com/alice_k106/221170259817>

부 록

<부록 1> BX\_3000A.chsmc

/\*

\*\* CHSM Language System

\*\* examples/c++/BX\_3000A.chsmc

\*\*

\*\* Copyright (C) 1996-2018 Paul J. Lucas & Fabio Riccardi

\*\*

\*\* This program is free software; you can redistribute it and/or modify

\*\* it under the terms of the GNU General Public License as published by

\*\* the Free Software Foundation; either version 3 of the License, or

\*\* (at your option) any later version.

\*\*

\*\* This program is distributed in the hope that it will be useful,

\*\* but WITHOUT ANY WARRANTY; without even the implied warranty of

\*\* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

\*\* GNU General Public License for more details.

\*\*

\*\* You should have received a copy of the GNU General Public License

\*\* along with this program. If not, see <http://www.gnu.org/licenses/>.

\*/

/\*\*

\* @file

\* Complete program to implement a microwave oven.

\*

\* @note This is a "toy" program that exists for pedagogical reasons.

\*/

// CHSM

#include <chsm.h>

// STANDARD

#include <cctype>

#include <iostream>

#include <iomanip>

using namespace std;

///////////////////////////////////////////////////////////////////////////////

class bx\_data : public CHSM::machine {

public:

bx\_data(CHSM\_MACHINE\_ARGS) : CHSM::machine(CHSM\_MACHINE\_INIT) { }

void positive\_run();

void positive\_stop();

void reverse\_run();

void reverse\_stop();

void all\_stop();

void wipeCounter();

void errCounter();

protected:

unsigned int wipe\_cnt;

unsigned int stop\_cnt;

};

void bx\_data::positive\_run() {

}

void bx\_data::reverse\_run() {

}

void bx\_data::positive\_stop(){

}

void bx\_data::reverse\_stop(){

}

void bx\_data::all\_stop() {

}

void bx\_data::wipeCounter() {

wipe\_cnt += 1;

}

void bx\_data::errCounter() {

stop\_cnt += 1;

}

%%

///////////////////////////////////////////////////////////////////////////////

chsm<bx\_data> BX\_3000A is {

set components( idle, operational, error ) is {

state idle {

wrong\_input -> err\_wrong\_input;

pos\_accel -> operational.acceleration %{

positive\_run();

%};

rev\_accel -> operational.acceleration %{

reverse\_turn();

%};

}

cluster operational( acceleration, positive\_turn, reverse\_turn, deceleration ) history {

high\_volt->error.err\_high\_volt;

emerg\_stop->error.err\_emerg;

} is {

state acceleration {

positive -> positive\_turn;

reverse -> reverse\_turn;

}

state positive\_turn {

pos\_decel -> deceleration %{

positive\_stop();

%};

}

state reverse\_turn {

rev\_decel -> deceleration %{

reverse\_stop();

%};

}

state deceleration {

stop->idle;

}

}

cluster error( err\_high\_volt, err\_emerg, err\_wrong\_input ) history {

upon enter %{

errCounter();

%}

finish->idle;

} is {

state err\_high\_volt {

upon enter %{

all\_stop();

%}

}

state err\_emerg {

upon enter %{

all\_stop();

%}

}

state err\_wrong\_input {

upon enter %{

all\_stop();

%}

}

}

}

}

///////////////////////////////////////////////////////////////////////////////

%%

int main() {

cout.fiil( '0' );

BX\_3000A bx;

bx.debug( CHSM::machine::DEBUG\_ALL );

bx.enter();

bx.emerg\_stop();

bx.finish();

while( !cin.eof() ) {

bx.dump\_state();

cout << "\nENTER: (P)os\_accel (R)ev\_accel (H)igh\_volt (E)merg\_stop "

"p(O)sitive re(V)erse pos\_dece(L) rev\_(D)ecel (F)inish (S)top\n";

char ch;

cin >> ch;

switch( toupper( ch ) ) {

case 'P': bx.pos\_accel(); break;

case 'R': bx.rev\_accel(); break;

case 'H': bx.high\_volt(); break;

case 'E': bx.emerg\_stop(); break;

case 'O': bx.positive(); break;

case 'V': bx.reverse(); break;

case 'L': bx.pos\_decel(); break;

case 'D': bx.rev\_decel(); break;

case 'F': bx.finish(); break;

case 'S': bx.stop(); break;

default : bx.wrong\_input(); break;

}

}

return 0;

}

}<부록 2> BX\_3000A.cpp

/\*\*

 \* @file

 \* Complete program to implement a BX\_3000A.

 \*

 \* @note

 \*/

 #include <chsm.h>

 #include <cctype>

 #include <iostream>

 #include <iomanip>

 using namespace std;

 /\*\*

 \* Contains additional data and member functions for a microwave CHSM.

 \*/

class bx\_data : public CHSM::machine {

public:

    bx\_data(CHSM\_MACHINE\_ARGS) : CHSM::machine(CHSM\_MACHINE\_INIT) { }

    void positive\_run();

    void positive\_stop();

    void reverse\_run();

    void reverse\_stop();

    void all\_stop();

    void wipeCounter();

    void errCounter();

protected:

    unsigned int wipe\_cnt;

    unsigned int stop\_cnt;

};

void bx\_data::positive\_run() {

}

void bx\_data::reverse\_run() {

}

void bx\_data::positive\_stop(){

}

void bx\_data::reverse\_stop(){

}

void bx\_data::all\_stop() {

}

void bx\_data::wipeCounter() {

    wipe\_cnt += 1;

}

void bx\_data::errCounter() {

    stop\_cnt += 1;

}

#ifndef infinite\_chsmc\_H

#define infinite\_chsmc\_H

///// CHSM: <<CHSM 5.0>>

#include <chsm.h>

namespace CHSM\_ns\_alias = CHSM\_NS;

///// CHSM: user declarations

#line 138 "BX\_3000A"

///// CHSM: machine class declarations

class BX\_3000A : public bx\_data {

public:

    BX\_3000A();

    ~BX\_3000A();

// state

    class state\_root : public CHSM\_ns\_alias::cluster {                      // id = -1

    public:

        state\_root(CHSM\_STATE\_ARGS, bool chsm\_history\_);

    private:

        static state::id const children\_[];

    } root;

    class state\_components : public CHSM\_ns\_alias::set {                    // id = 0

    public:

        CHSM\_ns\_alias::state idle;                                          // id = 1

        class state\_operational : public CHSM\_ns\_alias::cluster {           // id = 2

        public:

            CHSM\_ns\_alias::state acceleration;                               // id = 3

            CHSM\_ns\_alias::state positive\_turn;                             // id = 4

            CHSM\_ns\_alias::state reverse\_turn;                              // id = 5

            CHSM\_ns\_alias::state deceleration;                              // id = 6

            state\_operational(CHSM\_STATE\_ARGS, bool chsm\_history\_);

        private:

            static state::id const children\_[];

        } operational;

        class state\_error : public CHSM\_ns\_alias::cluster {                 // id = 7

        public:

            CHSM\_ns\_alias::state err\_wrong\_input;                           // id = 8

            CHSM\_ns\_alias::state err\_high\_volt;                             // id = 9

            CHSM\_ns\_alias::state err\_emerg;                                 // id = 10

            state\_error(CHSM\_STATE\_ARGS, bool chsm\_history\_);

        private:

            static state::id const children\_[];

        } error;

        state\_components(CHSM\_STATE\_ARGS);

    private:

        static state::id const children\_[];

    } components;

// event

private:

    static CHSM\_ns\_alias::transition::id const wrong\_input\_transitions[];

public:

    class wrong\_input\_event : public CHSM\_ns\_alias::event {

        typedef CHSM\_ns\_alias::event base\_event;

        typedef base\_event::param\_block base\_param\_block;

    public:

        struct param\_block : base\_param\_block {

            param\_block( CHSM\_ns\_alias::event const& event ) :

                base\_param\_block( event ) { }

        protected:

            virtual ~param\_block();

        };

    protected:

        wrong\_input\_event( CHSM\_EVENT\_ARGS ) : base\_event( CHSM\_EVENT\_INIT ) { }

        friend class BX\_3000A;

    } wrong\_input;

    alignas(wrong\_input\_event::param\_block) char wrong\_input\_param\_block[ sizeof(wrong\_input\_event::param\_block) ];

private:

    static CHSM\_ns\_alias::transition::id const pos\_accel\_transitions[];

public:

    class pos\_accel\_event : public CHSM\_ns\_alias::event {

        typedef CHSM\_ns\_alias::event base\_event;

        typedef base\_event::param\_block base\_param\_block;

    public:

        struct param\_block : base\_param\_block {

            param\_block( CHSM\_ns\_alias::event const& event ) :

                base\_param\_block( event ) { }

        protected:

            virtual ~param\_block();

        };

    protected:

        pos\_accel\_event( CHSM\_EVENT\_ARGS ) : base\_event( CHSM\_EVENT\_INIT ) { }

        friend class BX\_3000A;

    } pos\_accel;

    alignas(pos\_accel\_event::param\_block) char pos\_accel\_param\_block[ sizeof(pos\_accel\_event::param\_block) ];

private:

    static CHSM\_ns\_alias::transition::id const rev\_accel\_transitions[];

public:

    class rev\_accel\_event : public CHSM\_ns\_alias::event {

        typedef CHSM\_ns\_alias::event base\_event;

        typedef base\_event::param\_block base\_param\_block;

    public:

        struct param\_block : base\_param\_block {

            param\_block( CHSM\_ns\_alias::event const& event ) :

                base\_param\_block( event ) { }

        protected:

            virtual ~param\_block();

        };

    protected:

        rev\_accel\_event( CHSM\_EVENT\_ARGS ) : base\_event( CHSM\_EVENT\_INIT ) { }

        friend class BX\_3000A;

    } rev\_accel;

    alignas(rev\_accel\_event::param\_block) char rev\_accel\_param\_block[ sizeof(rev\_accel\_event::param\_block) ];

private:

    static CHSM\_ns\_alias::transition::id const positive\_transitions[];

public:

    class positive\_event : public CHSM\_ns\_alias::event {

        typedef CHSM\_ns\_alias::event base\_event;

        typedef base\_event::param\_block base\_param\_block;

    public:

        struct param\_block : base\_param\_block {

            param\_block( CHSM\_ns\_alias::event const& event ) :

                base\_param\_block( event ) { }

        protected:

            virtual ~param\_block();

        };

    protected:

        positive\_event( CHSM\_EVENT\_ARGS ) : base\_event( CHSM\_EVENT\_INIT ) { }

        friend class BX\_3000A;

    } positive;

    alignas(positive\_event::param\_block) char positive\_param\_block[ sizeof(positive\_event::param\_block) ];

private:

    static CHSM\_ns\_alias::transition::id const reverse\_transitions[];

public:

    class reverse\_event : public CHSM\_ns\_alias::event {

        typedef CHSM\_ns\_alias::event base\_event;

        typedef base\_event::param\_block base\_param\_block;

    public:

        struct param\_block : base\_param\_block {

            param\_block( CHSM\_ns\_alias::event const& event ) :

                base\_param\_block( event ) { }

        protected:

            virtual ~param\_block();

        };

    protected:

        reverse\_event( CHSM\_EVENT\_ARGS ) : base\_event( CHSM\_EVENT\_INIT ) { }

        friend class BX\_3000A;

    } reverse;

    alignas(reverse\_event::param\_block) char reverse\_param\_block[ sizeof(reverse\_event::param\_block) ];

private:

    static CHSM\_ns\_alias::transition::id const pos\_decel\_transitions[];

public:

    class pos\_decel\_event : public CHSM\_ns\_alias::event {

        typedef CHSM\_ns\_alias::event base\_event;

        typedef base\_event::param\_block base\_param\_block;

    public:

        struct param\_block : base\_param\_block {

            param\_block( CHSM\_ns\_alias::event const& event ) :

                base\_param\_block( event ) { }

        protected:

            virtual ~param\_block();

        };

    protected:

        pos\_decel\_event( CHSM\_EVENT\_ARGS ) : base\_event( CHSM\_EVENT\_INIT ) { }

        friend class BX\_3000A;

    } pos\_decel;

    alignas(pos\_decel\_event::param\_block) char pos\_decel\_param\_block[ sizeof(pos\_decel\_event::param\_block) ];

private:

    static CHSM\_ns\_alias::transition::id const rev\_decel\_transitions[];

public:

    class rev\_decel\_event : public CHSM\_ns\_alias::event {

        typedef CHSM\_ns\_alias::event base\_event;

        typedef base\_event::param\_block base\_param\_block;

    public:

        struct param\_block : base\_param\_block {

            param\_block( CHSM\_ns\_alias::event const& event ) :

                base\_param\_block( event ) { }

        protected:

            virtual ~param\_block();

        };

    protected:

        rev\_decel\_event( CHSM\_EVENT\_ARGS ) : base\_event( CHSM\_EVENT\_INIT ) { }

        friend class BX\_3000A;

    } rev\_decel;

    alignas(rev\_decel\_event::param\_block) char rev\_decel\_param\_block[ sizeof(rev\_decel\_event::param\_block) ];

private:

    static CHSM\_ns\_alias::transition::id const stop\_transitions[];

public:

    class stop\_event : public CHSM\_ns\_alias::event {

        typedef CHSM\_ns\_alias::event base\_event;

        typedef base\_event::param\_block base\_param\_block;

    public:

        struct param\_block : base\_param\_block {

            param\_block( CHSM\_ns\_alias::event const& event ) :

                base\_param\_block( event ) { }

        protected:

            virtual ~param\_block();

        };

    protected:

        stop\_event( CHSM\_EVENT\_ARGS ) : base\_event( CHSM\_EVENT\_INIT ) { }

        friend class BX\_3000A;

    } stop;

    alignas(stop\_event::param\_block) char stop\_param\_block[ sizeof(stop\_event::param\_block) ];

private:

    static CHSM\_ns\_alias::transition::id const high\_volt\_transitions[];

public:

    class high\_volt\_event : public CHSM\_ns\_alias::event {

        typedef CHSM\_ns\_alias::event base\_event;

        typedef base\_event::param\_block base\_param\_block;

    public:

        struct param\_block : base\_param\_block {

            param\_block( CHSM\_ns\_alias::event const& event ) :

                base\_param\_block( event ) { }

        protected:

            virtual ~param\_block();

        };

    protected:

        high\_volt\_event( CHSM\_EVENT\_ARGS ) : base\_event( CHSM\_EVENT\_INIT ) { }

        friend class BX\_3000A;

    } high\_volt;

    alignas(high\_volt\_event::param\_block) char high\_volt\_param\_block[ sizeof(high\_volt\_event::param\_block) ];

private:

    static CHSM\_ns\_alias::transition::id const emerg\_stop\_transitions[];

public:

    class emerg\_stop\_event : public CHSM\_ns\_alias::event {

        typedef CHSM\_ns\_alias::event base\_event;

        typedef base\_event::param\_block base\_param\_block;

    public:

        struct param\_block : base\_param\_block {

            param\_block( CHSM\_ns\_alias::event const& event ) :

                base\_param\_block( event ) { }

        protected:

            virtual ~param\_block();

        };

    protected:

        emerg\_stop\_event( CHSM\_EVENT\_ARGS ) : base\_event( CHSM\_EVENT\_INIT ) { }

        friend class BX\_3000A;

    } emerg\_stop;

    alignas(emerg\_stop\_event::param\_block) char emerg\_stop\_param\_block[ sizeof(emerg\_stop\_event::param\_block) ];

private:

    static CHSM\_ns\_alias::transition::id const finish\_transitions[];

public:

    class finish\_event : public CHSM\_ns\_alias::event {

        typedef CHSM\_ns\_alias::event base\_event;

        typedef base\_event::param\_block base\_param\_block;

    public:

        struct param\_block : base\_param\_block {

            param\_block( CHSM\_ns\_alias::event const& event ) :

                base\_param\_block( event ) { }

        protected:

            virtual ~param\_block();

        };

    protected:

        finish\_event( CHSM\_EVENT\_ARGS ) : base\_event( CHSM\_EVENT\_INIT ) { }

        friend class BX\_3000A;

    } finish;

    alignas(finish\_event::param\_block) char finish\_param\_block[ sizeof(finish\_event::param\_block) ];

// transition conditions

// transition targets

// transition actions

    void A1( CHSM\_ns\_alias::event const& );

    void A2( CHSM\_ns\_alias::event const& );

    void A3( CHSM\_ns\_alias::event const& );

    void A4( CHSM\_ns\_alias::event const& );

// enter/exit actions

    void EAM10components5error15err\_wrong\_input( CHSM\_ns\_alias::state const&, CHSM\_ns\_alias::event const& );

    void EAM10components5error13err\_high\_volt( CHSM\_ns\_alias::state const&, CHSM\_ns\_alias::event const& );

    void EAM10components5error9err\_emerg( CHSM\_ns\_alias::state const&, CHSM\_ns\_alias::event const& );

    void EAM10components5error( CHSM\_ns\_alias::state const&, CHSM\_ns\_alias::event const& );

private:

    CHSM\_ns\_alias::state \*state\_[12];

    static CHSM\_ns\_alias::transition const transition\_[];

    CHSM\_ns\_alias::event const \*taken\_[11];

    CHSM\_ns\_alias::state \*target\_[11];

};

///// CHSM: THE END

#endif

#include <new>

///// CHSM: state definitions

CHSM\_ns\_alias::state::id const BX\_3000A::state\_root::children\_[] = {

    0, -1

};

BX\_3000A::state\_root::state\_root(CHSM\_STATE\_ARGS, bool chsm\_history\_) :

    CHSM\_ns\_alias::cluster(CHSM\_STATE\_INIT, children\_, chsm\_history\_)

{

}

CHSM\_ns\_alias::state::id const BX\_3000A::state\_components::children\_[] = {

    1, 2, 7, -1

};

BX\_3000A::state\_components::state\_components(CHSM\_STATE\_ARGS) :

    CHSM\_ns\_alias::set(CHSM\_STATE\_INIT, children\_),

    idle(chsm\_machine\_, "components.idle", this, nullptr, nullptr, nullptr, nullptr ),

    operational(chsm\_machine\_, "components.operational", this, nullptr, nullptr, nullptr, nullptr ,true ),

    error(chsm\_machine\_, "components.error", this, static\_cast<CHSM\_ns\_alias::state::action>(&BX\_3000A::EAM10components5error), nullptr, nullptr, nullptr, true )

{

}

CHSM\_ns\_alias::state::id const BX\_3000A::state\_components::state\_operational::children\_[] = {

    3, 4, 5, 6, -1

};

BX\_3000A::state\_components::state\_operational::state\_operational(CHSM\_STATE\_ARGS, bool chsm\_history\_) :

    CHSM\_ns\_alias::cluster(CHSM\_STATE\_INIT, children\_, chsm\_history\_),

    acceleration(chsm\_machine\_, "components.operational.acceleration", this, nullptr, nullptr, nullptr, nullptr ),

    positive\_turn(chsm\_machine\_, "components.operational.positive\_turn", this, nullptr, nullptr, nullptr, nullptr ),

    reverse\_turn(chsm\_machine\_, "components.operational.reverse\_turn", this, nullptr, nullptr, nullptr, nullptr ),

    deceleration(chsm\_machine\_, "components.operational.deceleration", this, nullptr, nullptr, nullptr, nullptr )

{

}

CHSM\_ns\_alias::state::id const BX\_3000A::state\_components::state\_error::children\_[] = {

    8, 9, 10, -1

};

BX\_3000A::state\_components::state\_error::state\_error(CHSM\_STATE\_ARGS, bool chsm\_history\_) :

    CHSM\_ns\_alias::cluster(CHSM\_STATE\_INIT, children\_, chsm\_history\_),

    err\_wrong\_input(chsm\_machine\_, "components.error.err\_wrong\_input", this, static\_cast<CHSM\_ns\_alias::state::action>(&BX\_3000A::EAM10components5error15err\_wrong\_input), nullptr, nullptr, nullptr ),

    err\_high\_volt(chsm\_machine\_, "components.error.err\_high\_volt", this, static\_cast<CHSM\_ns\_alias::state::action>(&BX\_3000A::EAM10components5error13err\_high\_volt), nullptr, nullptr, nullptr ),

    err\_emerg(chsm\_machine\_, "components.error.err\_emerg", this, static\_cast<CHSM\_ns\_alias::state::action>(&BX\_3000A::EAM10components5error9err\_emerg), nullptr, nullptr, nullptr )

{

}

///// CHSM: event definitions

CHSM\_ns\_alias::transition::id const BX\_3000A::wrong\_input\_transitions[] = {

    0, -1

};

BX\_3000A::wrong\_input\_event::param\_block::~param\_block() { }

CHSM\_ns\_alias::transition::id const BX\_3000A::pos\_accel\_transitions[] = {

    1, -1

};

BX\_3000A::pos\_accel\_event::param\_block::~param\_block() { }

CHSM\_ns\_alias::transition::id const BX\_3000A::rev\_accel\_transitions[] = {

    2, -1

};

BX\_3000A::rev\_accel\_event::param\_block::~param\_block() { }

CHSM\_ns\_alias::transition::id const BX\_3000A::high\_volt\_transitions[] = {

    3, -1

};

BX\_3000A::high\_volt\_event::param\_block::~param\_block() { }

CHSM\_ns\_alias::transition::id const BX\_3000A::emerg\_stop\_transitions[] = {

    4, -1

};

BX\_3000A::emerg\_stop\_event::param\_block::~param\_block() { }

CHSM\_ns\_alias::transition::id const BX\_3000A::positive\_transitions[] = {

    5, -1

};

BX\_3000A::positive\_event::param\_block::~param\_block() { }

CHSM\_ns\_alias::transition::id const BX\_3000A::reverse\_transitions[] = {

    6, -1

};

BX\_3000A::reverse\_event::param\_block::~param\_block() { }

CHSM\_ns\_alias::transition::id const BX\_3000A::pos\_decel\_transitions[] = {

    7, -1

};

BX\_3000A::pos\_decel\_event::param\_block::~param\_block() { }

CHSM\_ns\_alias::transition::id const BX\_3000A::rev\_decel\_transitions[] = {

    8, -1

};

BX\_3000A::rev\_decel\_event::param\_block::~param\_block() { }

CHSM\_ns\_alias::transition::id const BX\_3000A::stop\_transitions[] = {

    9, -1

};

BX\_3000A::stop\_event::param\_block::~param\_block() { }

CHSM\_ns\_alias::transition::id const BX\_3000A::finish\_transitions[] = {

    10, -1

};

BX\_3000A::finish\_event::param\_block::~param\_block() { }

///// CHSM: Transitions

CHSM\_ns\_alias::transition const BX\_3000A::transition\_[] = {

    { nullptr, 1, 8, nullptr, nullptr },

    { nullptr, 1, 3, nullptr, static\_cast<CHSM\_ns\_alias::transition::action>(&BX\_3000A::A1) },

    { nullptr, 1, 3, nullptr, static\_cast<CHSM\_ns\_alias::transition::action>(&BX\_3000A::A2) },

    { nullptr, 2, 9, nullptr, nullptr },

    { nullptr, 2, 10, nullptr, nullptr },

    { nullptr, 3, 4, nullptr, nullptr },

    { nullptr, 3, 5, nullptr, nullptr },

    { nullptr, 4, 6, nullptr, static\_cast<CHSM\_ns\_alias::transition::action>(&BX\_3000A::A3) },

    { nullptr, 5, 6, nullptr, static\_cast<CHSM\_ns\_alias::transition::action>(&BX\_3000A::A4) },

    { nullptr, 6, 1, nullptr, nullptr },

    { nullptr, 7, 1, nullptr, nullptr }

};

///// CHSM: CHSM Constructor definition

BX\_3000A::BX\_3000A() :

    bx\_data ( state\_, root, transition\_, taken\_, target\_, 11 ),

    root( \*this, "root", nullptr, nullptr, nullptr, nullptr, nullptr, false ),

    components( \*this, "components", &root, nullptr, nullptr, nullptr, nullptr ),

    wrong\_input( this, wrong\_input\_transitions, "wrong\_input", nullptr ),

    pos\_accel( this, pos\_accel\_transitions, "pos\_accel", nullptr ),

    rev\_accel( this, rev\_accel\_transitions, "rev\_accel", nullptr ),

    high\_volt( this, high\_volt\_transitions, "high\_volt", nullptr ),

    emerg\_stop( this, emerg\_stop\_transitions, "emerg\_stop", nullptr ),

    positive( this, positive\_transitions, "positive", nullptr ),

    reverse( this, reverse\_transitions, "revers", nullptr ),

    pos\_decel( this, pos\_decel\_transitions, "pos\_decel", nullptr ),

    rev\_decel( this, rev\_decel\_transitions, "rev\_decel", nullptr ),

    stop( this, stop\_transitions, "stop", nullptr ),

    finish( this, finish\_transitions, "finish", nullptr )

{

    state\_[0] = &components;

    state\_[1] = &components.idle;

    state\_[2] = &components.operational;

    state\_[3] = &components.operational.acceleration;

    state\_[4] = &components.operational.positive\_turn;

    state\_[5] = &components.operational.reverse\_turn;

    state\_[6] = &components.operational.deceleration;

    state\_[7] = &components.error;

    state\_[8] = &components.error.err\_wrong\_input;

    state\_[9] = &components.error.err\_high\_volt;

    state\_[10] = &components.error.err\_emerg;

    state\_[11] = nullptr;

}

BX\_3000A::~BX\_3000A() { }

// user-code

void BX\_3000A::A1( CHSM\_ns\_alias::event const &event ) {

    (void)event;

#line 88 "infinite.chsmc"

        positive\_run();

    }

void BX\_3000A::A2( CHSM\_ns\_alias::event const &event ) {

    (void)event;

#line 92 "infinite.chsmc"

        reverse\_run();

}

void BX\_3000A::A3( CHSM\_ns\_alias::event const &event ) {

    (void)event;

#line 110 "infinite.chsmc"

        positive\_stop();

}

void BX\_3000A::A4( CHSM\_ns\_alias::event const &event ) {

    (void)event;

#line 117 "infinite.chsmc"

        reverse\_stop();

}

void BX\_3000A::EAM10components5error( CHSM\_ns\_alias::state const &chsm\_state\_, CHSM\_ns\_alias::event const &event ) {

    (void)event;

#line 127 "infinite.chsmc"

    CHSM\_ns\_alias::cluster const &state = dynamic\_cast<CHSM\_ns\_alias::cluster const&>(chsm\_state\_);

    (void)state;

        errCounter();

}

void BX\_3000A::EAM10components5error13err\_high\_volt( CHSM\_ns\_alias::state const &chsm\_state\_, CHSM\_ns\_alias::event const &event ) {

    (void)event;

#line 133 "infinite.chsmc"

    CHSM\_ns\_alias::state const &state = dynamic\_cast<CHSM\_ns\_alias::state const&>(chsm\_state\_);

    (void)state;

        all\_stop();

}

void BX\_3000A::EAM10components5error9err\_emerg( CHSM\_ns\_alias::state const &chsm\_state\_, CHSM\_ns\_alias::event const &event ) {

    (void)event;

#line 139 "infinite.chsmc"

    CHSM\_ns\_alias::state const &state = dynamic\_cast<CHSM\_ns\_alias::state const&>(chsm\_state\_);

    (void)state;

        all\_stop();

}

void BX\_3000A::EAM10components5error15err\_wrong\_input( CHSM\_ns\_alias::state const &chsm\_state\_, CHSM\_ns\_alias::event const &event ) {

    (void)event;

#line 145 "infinite.chsmc"

    CHSM\_ns\_alias::state const &state = dynamic\_cast<CHSM\_ns\_alias::state const&>(chsm\_state\_);

    (void)state;

        all\_stop();

}

#line 152 "infinite.chsmc"

///// CHSM: THE END

int main() {

    BX\_3000A bx;

    bx.debug( CHSM::machine::DEBUG\_ALL );

    bx.enter();

    bx.emerg\_stop();

    bx.finish();

    while( !cin.eof() ) {

        bx.dump\_state();

        cout << "\nENTER: (P)os\_accel (R)ev\_accel (H)igh\_volt (E)merg\_stop "

                "p(O)sitive re(V)erse pos\_dece(L) rev\_(D)ecel (F)inish (S)top\n";

        char ch;

        cin >> ch;

        switch( toupper( ch ) ) {

            case 'P': bx.pos\_accel(); break;

            case 'R': bx.rev\_accel(); break;

            case 'H': bx.high\_volt(); break;

            case 'E': bx.emerg\_stop(); break;

            case 'O': bx.positive(); break;

            case 'V': bx.reverse(); break;

            case 'L': bx.pos\_decel(); break;

            case 'D': bx.rev\_decel(); break;

            case 'F': bx.finish(); break;

            case 'S': bx.stop(); break;

            default : bx.wrong\_input(); break;

        }

    }

    return 0;

}