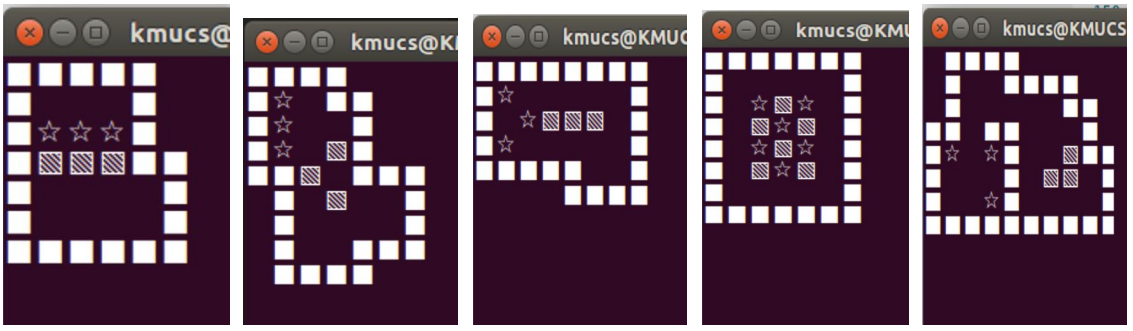
C++ 프로그래밍 프로젝트 보고서

ncurses 라이브러리를 이용한 PushBox Game

20181617 박정현
20181619 박종흠
20181686 장병준

1단계 실행 화면



Stage1 Stage2 Stage3 Stage4 Stage5

1단계 소스코드

1. 변수 및 상수 구성
   - A. GAMEY, GAMEX - 맵 크기
   - B. BLOCK (enum) - 블럭 타입 열거체
   - C. gamemap (2차원 char vector) - 맵 데이터(벽, 목표지점, 바닥, 맵밖)
   - D. objectmap (2차원 char vector) - 오브젝트 데이터(상자, 플레이어)
   - E. res (1차원 string 배열) - 블럭 리소스

2. 함수 구성
   - A. string getresource(int type) - 블럭 타입을 리소스로 변환
   - B. void gameinit() - 게임(ncurses) 초기설정
   - C. void loadstage(int stage_num) - 스테이지 로드(파일 입출력)
   - D. void refreshmap() - 화면 업데이트
   - E. int main() - 메인함수

3. 함수 구현
   A. string getresource(int type)

```
// 블럭 타입을 리소스로 변환
string getresource(int type)
{
    // type을 인덱스로 변환해서 반환
    return res[type - FLOOR];
}
```

   B. void gameinit()

```
// 게임(ncurses) 초기설정
void gameinit()
{
    // 한글 출력을 위한 locale설정
    setlocale(LC_ALL,"");

    // 맵 크기에 맞게 터미널 크기 변경
    resize_term(GAMEY, 2*GAMEX);
    char cmd[100];
    sprintf(cmd, "resize -s %d %d", GAMEY, 2*GAMEX);
    system(cmd);

    // curses 모드 시작
    initscr();

    // 키보드 입력
    keypad(stdscr, TRUE);
    // 커서설정
    curs_set(0);
    noecho();
}
```

C. void loadstage(int stage_num)

```
// 스테이지 로드(파일 입출력)
void loadstage(int stage_num) {
    // 스테이지 데이터 파일 열기
    ifstream f("stage/" + to_string(stage_num));
    if (f.is_open()) {
        int r, c;
        f >> r >> c;
        gamemap.resize(r, vector<char>(c, FLOOR));
        objectmap.resize(r, vector<char>(c, NONE));
        for(int i = 0; i < r; i++){
            for(int j = 0; j < c; j++){
                char ch;
                f >> ch;
                if(ch == BOX) objectmap[i][j] = ch;
                else gamemap[i][j] = ch;
            }
        }
        f >> playery >> playerx;
        objectmap[playery][playerx] = PLAYER;
        f.close();
    } else {
        printw("파일 없음");
    }
}
```

D. void refreshmap()

```
// 화면 업데이트
void refreshmap()
{
    // 맵 데이터 출력
    for(int i=0; i<gamemap.size(); i++){
        for(int j=0; j<gamemap[i].size(); j++){
            mvprintw(i,2*j,"%s",getresource(gamemap[i][j]).c_str());
        }
    }
    // 그 위에 오브젝트 데이터 출력
    for(int i=0; i<objectmap.size(); i++){
        for(int j=0; j<objectmap[i].size(); j++){
            mvprintw(i,2*j,"%s", getresource(objectmap[i][j]).c_str());
        }
    }
    // 터미널 화면 업데이트
    refresh();
}
```

E. int main()

```
// 메인함수
int main()
{
    // 게임 초기설정
    gameinit();
    // 스테이지 로드
    loadstage(2);   // 1 ~ 5
    // 키 입력시 종료
    getch();
    endwin();
    return 0;
}
```

2단계 실행 화면

Stage1 Stage2 Stage3 Stage4 Stage5

2단계 소스코드

1. 변수 및 상수 구성
   A. playerx, playery - 플레이어의 좌표
   B. diry[4], dirx[4] (1차원 int 배열) - 플레이어가 움직일 방향(상하좌우)

2. 함수 구성
   A. void keyevent() - 사용자가 입력한 키코드를 처리
   B. bool moveobject(int y, int x, int dir, int count)
      - 사용자 입력을 받으면 오브젝트의 움직임을 처리
   C. int getdiry(int dir) - 키 입력에 따른 y 방향 값 반환
   D. int getdirx(int dir) - 키 입력에 따른 x 방향 값 반환

3. 함수 구현
   A. void keyevent()

```
void keyevent(){
    int key;
    do{
        // movement
        switch (key)
        {
        case KEY_DOWN:
        case KEY_UP:
        case KEY_LEFT:
        case KEY_RIGHT:
            if(moveobject(playery, playerx, key, 1)){
                playery += getdiry(key);
                playerx += getdirx(key);
            }
            break;
```

```
            default:
                break;
            }

            refreshmap();
            // 스테이지가 클리어 되었는지 확인한다.
            if(clearcheck()){
                break;
            }
        }while((key = getch()) != KEY_F(2));
        // F2키를 누르면 게임을 즉시 중단한다.
    }
```

B. bool moveobject(int y, int x, int dir, int count)

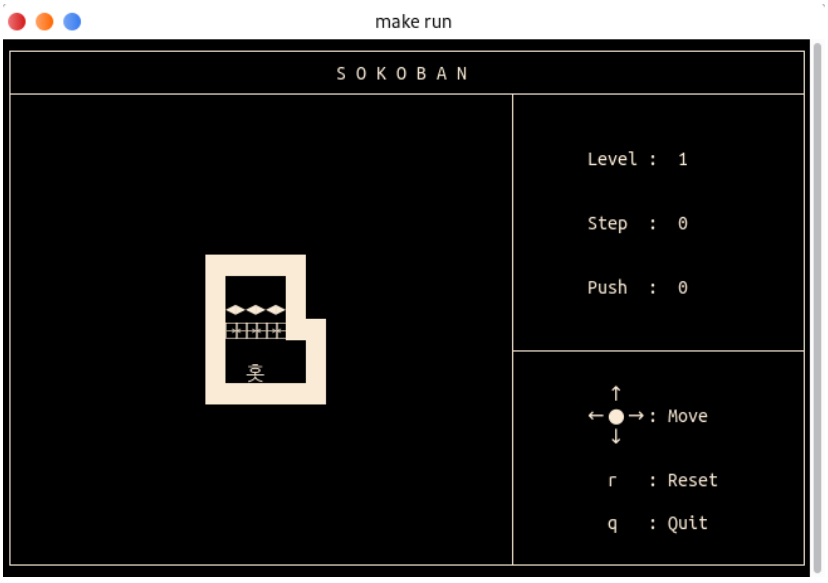
```
    // nCurses의 키를 입력받으면, 오브젝트의 움직임을 처리
    bool moveobject(int y, int x, int dir, int count)
    {
        if(count > 2) return false;
        int movey = y + getdiry(dir), movex = x + getdirx(dir);
        if(gamemap[movey][movex] == WALL)
            return false;
        if(objectmap[movey][movex] == BOX){
            if(!moveobject(movey, movex, dir, count + 1))
                return false;
            push++;
            step--;
        }
        objectmap[movey][movex] = objectmap[y][x];
        objectmap[y][x] = NONE;
        step++;
        return true;
    }
```

C. getdiry(int dir), getdirx(int dir)

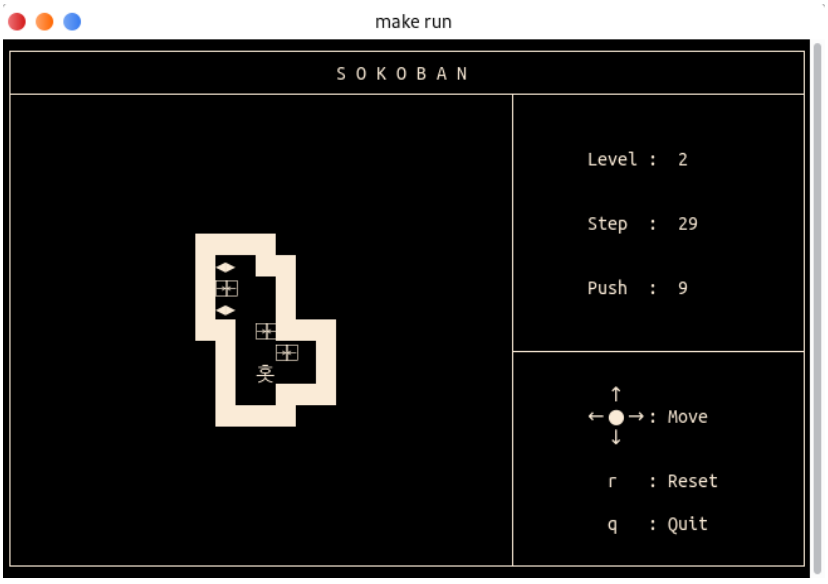
```
    // nCurses의 키를 입력받으면 변환해야할 y 좌표 값을 반환
    int getdiry(int dir)
    {
        return diry[dir - KEY_DOWN];
    }
```

```
// nCurses의 키를 입력받으면 변환해야할 x 좌표 값을 반환
int getdirx(int dir)
{
    return dirx[dir - KEY_DOWN];
}
```

3단계 실행 화면



Game with UI



Game with Player move

3단계 소스코드

1. 변수 및 상수 구성
   - A. level - 스테이지 레벨
   - B. step - 플레이어가 움직인 횟수
   - C. push - 상자가 움직인 횟수
   - D. right_top - 오른쪽 점수판
   - E. right_bottom - 오른쪽 플레이 안내
   - F. title - 상단 타이틀
   - G. game - 게임 화면

2. 함수 업데이트

A. moveobject 함수에 step, push 카운팅 추가

```
// nCurses의 키를 입력받으면, 오브젝트의 움직임을 처리
bool moveobject(int y, int x, int dir, int count) {
    if (count > 2)
        return false;
    int movey = y + getdiry(dir), movex = x + getdirx(dir);
    // 이동하려는 방향에 벽인지 확인
    if (gamemap[movey][movex] == WALL)
        return false;
    // 이동하려는 방향이 상자인지 확인
    if (objectmap[movey][movex] == BOX) {
        // 상자를 움직일 수 있는지 확인
        if (!moveobject(movey, movex, dir, count + 1))
            return false;
        // 상자를 움직인 횟수 증가
        push++;
        // 플레이어가 움직인 횟수 감소
        step--;
    }
    // 현재 물체를 이동하려는 방향으로 옮김
    objectmap[movey][movex] = objectmap[y][x];
    objectmap[y][x] = NONE;
    // 플레이어가 움직인 횟수 증가
    step++;
    return true;
}
```

B. UI(윈도우) 구성

```
// 터미널 전체 보더
    border(ACS_VLINE, ACS_VLINE, ACS_HLINE, ACS_HLINE, ACS_ULCORNER,
           ACS_URCORNER, ACS_LLCORNER, ACS_LRCORNER);
    refresh();

    // 윈도우 선언
    right_top = newwin(13, 30, 2, 50);
    right_bottom = newwin(11, 30, 14, 50);
    title = newwin(3, 0, 0, 0);
```

```
    game = newwin(21, 47, 3, 2);

    // 윈도우 내부 보더 표시
    wborder(right_top, ACS_VLINE, ACS_VLINE, ACS_HLINE, ACS_HLINE,
           ACS_TTEE, ACS_RTEE, ACS_LTEE, ACS_RTEE);
    wborder(right_bottom, ACS_VLINE, ACS_VLINE, ACS_HLINE, ACS_HLINE,
           ACS_LTEE, ACS_RTEE, ACS_BTEE, ACS_LRCORNER);
    wborder(title, ACS_VLINE, ACS_VLINE, ACS_HLINE, ACS_HLINE,
           ACS_ULCORNER, ACS_URCORNER, ACS_LTEE, ACS_RTEE);
    wborder(game, ACS_VLINE, ACS_VLINE, ACS_HLINE, ACS_HLINE,
           ACS_ULCORNER, ACS_URCORNER, ACS_LLCORNER, ACS_LRCORNER);

    // 타이틀에 글자 표시
    mvwprintw(title, 1, 33, "S O K O B A N");

    // 설명 입력
    mvwprintw(right_bottom, 1, 10, "↑");
    mvwprintw(right_bottom, 2, 8, "← ● → : Move");
    mvwprintw(right_bottom, 3, 10, "↓");
    mvwprintw(right_bottom, 5, 10, "r    : Reset");
    mvwprintw(right_bottom, 7, 10, "q    : Quit");
    mvwprintw(right_bottom, 9, 10, "z    : Undo");

    // 윈도우 갱신
    wrefresh(title);
    wrefresh(right_top);
    wrefresh(right_bottom);
    wrefresh(game);
```

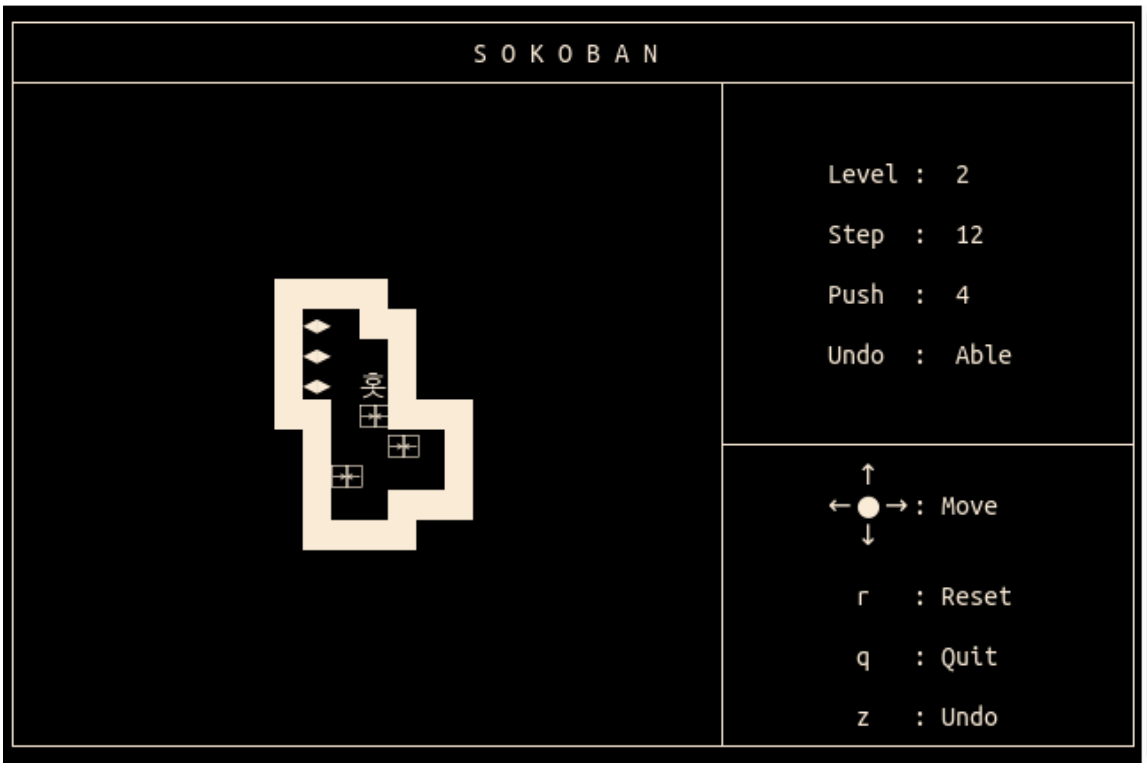
C. 점수판, 게임 화면 업데이트 구현

```
// 화면 업데이트
void refreshmap() {
    // 맵을 윈도우 중앙에 놓기
    int offsety, offsetx;
    offsety = 10 - gamemap.size() / 2;
    offsetx = 24 - gamemap[0].size();
    // 맵 데이터 출력
    for (int i = 0; i < gamemap.size(); i++) {
        for (int j = 0; j < gamemap[i].size(); j++) {
```

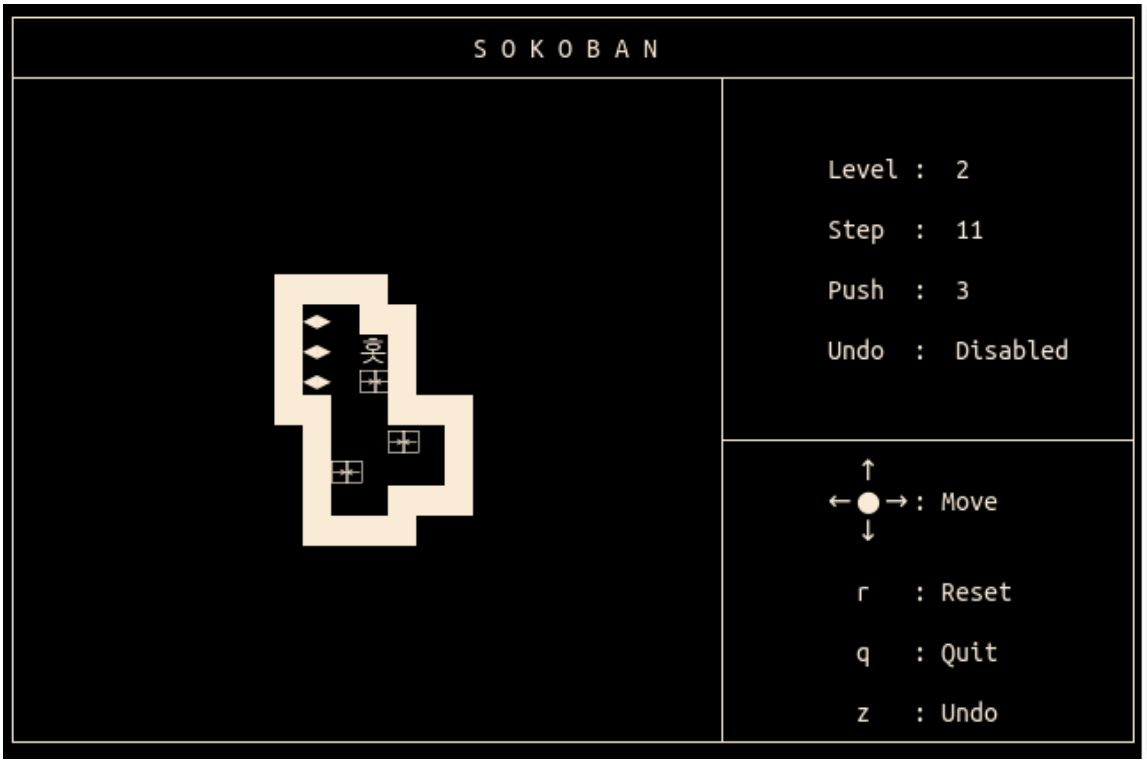
```
            mvwprintw(game, i + offsety, 2 * j + offsetx, "%s",
            getresource(gamemap[i][j]).c_str());
        }
    }
    // 그 위에 오브젝트 데이터 출력
    for (int i = 0; i < objectmap.size(); i++) {
        for (int j = 0; j < objectmap[i].size(); j++) {
            mvwprintw(game, i + offsety, 2 * j + offsetx, "%s",
            getresource(objectmap[i][j]).c_str());
        }
    }
    // 터미널 화면 업데이트
    wrefresh(game);
}

// 우측 정보 업데이트
void refreshstatus() {
    mvwprintw(right_top, 3, 8, "Level :  %d", level);
    mvwprintw(right_top, 6, 8, "Step  :  %d", step);
    mvwprintw(right_top, 9, 8, "Push  :  %d", push);
    wrefresh(right_top);
}
```

추가 기능



Undo 실행 전



Undo 실행 후

1. Undo(실행취소) 기능 (최근 1번 이동까지 실행취소 가능) ‘z’키로 작동

```
// 실행취소
void undo() {
    // 움직인 적이 있을때만 사용가능
    if(!step) return;
    // 이전에 실행취소했다면 return
    if(prev_push == -1) return;
    // 이전 데이터로 롤백
    step = prev_step;
    push = prev_push;
    playerx = prev_playerx;
    playery = prev_playery;
    gamemap = prev_gamemap;
    objectmap = prev_objectmap;
    prev_push = -1;
}
```

Makefile

Makefile 명령

1. make
   - 소스코드 빌드
2. make clean
   - 목적파일(.o), 실행파일(a.out), 의존성파일(.d) 삭제
3. make run
   - 빌드된 실행파일 (build/a.out) 실행
4. make vars
   - 빌드에 사용되는 매크로 내용을 화면에 출력

Makefile 코드

```
CXX := g++
```
- makefile에서 사용할 c++ 컴파일러

```
CXXFLAGS := -std=c++14 -g
```
- c++ 컴파일러 플래그

```
LDFLAGS := $(CXXFLAGS) -MMD -MP -c
```
- 목적파일(.o) 생성을 위한 플래그 MMD MP옵션으로 의존성파일(.d) 생성

```
LIBS := -lncursesw
```
- 사용하는 라이브러리 포함 (ncursesw)

```
Q := @
```
- makefile 커맨드를 화면에 표시하지 않도록 하는 옵션

```
ECHO := $(Q) echo
```
- 화면에 echo 뒤 내용을 출력, 출력을 보기 좋게 만들기 위해 추가

```
OUT := build
```
- 목적파일, 의존성파일, 실행파일이 생성되는 폴더

```
SRC := src
```
- 소스코드가 들어있는 폴더

```
TARGET := $(OUT)/a.out
```
- 실행파일 선언 매크로

```
SOURCES := $(wildcard $(SRC)/*.cpp)
```
- 소스코드 폴더에서 .cpp파일을 모두 찾아 저장

```
OBJECTS := $(subst $(SRC),$(OUT),$(patsubst %.cpp,%.o,$(SOURCES)))
```
- 목적파일은 SOURCES매크로에서 .cpp를 .o로 변경하고, SRC폴더대신 OUT폴더를 사용하도록 변경

```
DEPENDS := $(patsubst %.o,%.d,$(OBJECTS))
```
- 의존성 파일은 목적파일과 같은 이름에 .o를 .d로 변경한 것.

```
.PHONY: all clean vars
```

- all clean vars명령은 실제 파일이 생성되는것이 아니므로 .PHONY로 선언하여야 폴더에 all, clean, vas 파일이 있어도 문제가 발생하지 않음.

```
all: $(OUT) $(TARGET)
```

- 소스코드 빌드(make 기본값)

```
vars:
    $(ECHO) 'TARGET - $(TARGET), OBJECTS - $(OBJECTS), DEPENDS -
$(DEPENDS), SOURCES - $(SOURCES)'
```

- 각종 매크로 정보를 출력

```
$(OUT):
    $(ECHO) "'$(OUT)' folder is not exists. creating folder."
    $(Q) mkdir -p $(OUT)
```

- OUT매크로에 해당하는 폴더가 없을 경우 생성

```
$(TARGET): $(OBJECTS)
    $(ECHO) '[LD] $^ => $@'
    $(Q) $(CXX) $(CXXFLAGS) -o $(TARGET) $(OBJECTS) $(LIBS)
```

- 목적파일을 모두 모아 타겟 파일(실행파일) 생성

```
$(OUT)/%.o: $(SRC)/%.cpp Makefile
    $(ECHO) '[CXX] $< => $@'
    $(Q) $(CXX) $(LDFLAGS) -o $@ $< $(LIBS)
```

- .cpp파일을 컴파일하여 .o파일을 생성

```
clean:
    $(ECHO) '[DEL] $(TARGET)'
    $(Q)rm -f $(TARGET)
    $(ECHO) '[DEL] $(OBJECTS)'
    $(Q)rm -f $(OBJECTS)
    $(ECHO) '[DEL] $(DEPENDS)'
    $(Q)rm -f $(DEPENDS)
```

- 실행파일, 목적파일, 의존성파일 삭제

```
run:
    $(Q) $(TARGET)
```

- 빌드된 프로그램 실행

```
-include $(DEPENDS)
```

- .d에 해당하는 파일을 Makefile로 포함하는 옵션
- .cpp파일이 참조하는 헤더나 다른 소스코드에 대한 정보가 저장되어 있는 .d파일을 Makefile에 포함하기 때문에, 소스코드가 참조하는 헤더가 수정되는 경우 자동으로 증분빌드를 하게 된다.