6장. 디버깅과 오류처리

소프트웨어 개발

- 소프트웨어 개발
  - 분석 → 설계 → 구현(소스코드)
  - 구현된 소스 코드에 대한 검증
    - 소스 코드 내에 매크로 등을 이용하여 테스트 코드 삽입
    - 오류 처리 기능**(errno, perror, assert)** 등을 이용한 오류 분석
    - **GDB** 와 같은 디버거 사용

GDB

GDB (GNU DeBugger)

- GNU 프로젝트의 디버거
- C, C++, Objective-C, Modular 2, Ada 등의 언어 지원
- 기능
  - 프로그램을 시작한다.
  - 프로그램이 특정 조건을 만족할 때 정지시킨다.
  - 프로그램이 정지했을 때 어떤 일이 벌어졌는지 조사한다.
  - 프로그램 일부를 수정하여 발견한 버그를 제거했을 때의 실행 결과를 미리 알아볼 수 있게 한다.
- 요구사항
  - Gcc 컴파일 시에 "-g" 옵션을 사용하여 디버깅 정보를 오브젝트 코드에 삽입
  - 디버깅 수준에 따라 -g1, -g2, -g3 등과 같이 명시
    - 명시하지 않으면 기본적으로 –g2 옵션 사용
- 공식 홈페이지 : http://www.gnu.org/software/gdb/

GDB

GDB 사용 예

◈ 프로그램 소스 코드

```
 1 /*
 2  * 4장 디버깅과 오류 처리
 3  * 파일 이름: dbg_first.c
 4  */
 5
  6 #include <stdio.h>
 7
 8 main()
 9 {
10    printf("Hello, World.\n");
11 }
```

◈ 실행 파일 생성 명령

```
[cprog2@seps5 ch4]$ gcc -g dbg_first.c -o dbg_first
[cprog2@seps5 ch4]$ dbg_first
Hello, World.
```

GDB

GDB 사용 예

```
[cprog2@seps5 ch4]$ gdb dbg_first
GNU gdb Red Hat Linux (5.2.1-4)
Copyright 2002 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-redhat-linux"...
(gdb) list          // 프로그램 소스를 리스트
1       /*
2        * 4장 디버깅과 오류 처리
3        * 파일 이름: dbg_first.c
4        */
5
6       #include <stdio.h>
7
8       main()
9       {
10          printf("Hello, World.\n");
```

GDB

```
(gdb) break 10      // 10번째 줄에 정지점 설정
Breakpoint 1 at 0x8048338: file dbg_first.c, line 10.
(gdb) run           // 디버깅 시작을 위하여 프로그램 실행
Starting program: /home/cprog2/book2/ch4/dbg_first
Breakpoint 1, main () at dbg_first.c:10    // 정지점에서 프로그램 실행 정지
10        printf("Hello, World.\n");        // 다음번에 실행할 줄을 보여준다.
(gdb) next                                  // 한 단계 실행
Hello, World.
11      }
(gdb) next
0x4003b56d in __libc_start_main () from /lib/libc.so.6
(gdb) next
Single stepping until exit from function __libc_start_main,
which has no line number information.
Program exited with code 016.
(gdb) quit                      // gbd 종료
[cprog2@seps5 ch4]$
```

GDB 도움말

help 명령어 – GDB 명령어에 대한 정보 조회

```
(gdb) help              // 명령어 클래스에 대한 도움말 조회
List of classes of commands:
aliases -- Aliases of other commands
breakpoints -- Making program stop at certain points
data -- Examining data
files -- Specifying and examining files
internals -- Maintenance commands
obscure -- Obscure features
running -- Running the program
stack -- Examining the stack
status -- Status inquiries
support -- Support facilities
tracepoints -- Tracing of program execution without stopping the program
user-defined -- User-defined commands

Type "help" followed by a class name for a list of commands in that class.
Type "help" followed by command name for full documentation.
Command name abbreviations are allowed if unambiguous.
(gdb)
```

GDB 도움말

▣ Status 도움말

◈ 프로그램 관련된 디버깅 정보 검색과 관련된 명령어 모음

```
(gdb) help status          // running 명령어 클래스에 대한 도움말 조회
Status inquiries.
List of commands:
info -- Generic command for showing things about the program being debugged
show -- Generic command for showing things about the debugger

Type "help" followed by command name for full documentation.
Command name abbreviations are allowed if unambiguous.
(gdb)
```

GDB 도움말

Files 도움말

- 작업 디렉토리 설정/검색, 실행 파일 지정 등의 파일과 관련된 명령어 모음
- 잘 사용하는 명령어
  - **cd:** 디버거와 디버깅중인 프로그램에 대한 작업 디렉토리를 변경함.
  - **directory:** 소스 파일에 대한 탐색경로를 추가함.
  - **file:** 디버깅할 파일을 지정함.
  - **list:** 지정된 함수 또는 줄을 보여줌.
  - **path:** 목적 파일에 대한 탐색경로를 추가함.
  - **pwd:** 현재의 작업 디렉토리를 보여줌.
  - **search:** 정규 수식을 가장 최근에 보여진 줄 다음부터 탐색함.

GDB 도움말

Files 도움말

(gdb) ***help files***
Specifying and examining files.
List of commands:
add-shared-symbol-files -- Load the symbols from shared objects in the dynamic linker's link map
add-symbol-file -- Usage: add-symbol-file FILE ADDR [-s <SECT> <SECT_ADDR> -s <SECT> <SECT_ADDR>
cd -- Set working directory to DIR for debugger and program being debugged
core-file -- Use FILE as core dump for examining memory and registers
directory -- Add directory DIR to beginning of search path for source files
exec-file -- Use FILE as program for getting contents of pure memory
file -- Use FILE as program to be debugged
forward-search -- Search for regular expression (see regex(3)) from last line listed
generate-core-file -- Save a core file with the current state of the debugged process
list -- List specified function or line
load -- Dynamically load FILE into the running program
nosharedlibrary -- Unload all shared object library symbols
path -- Add directory DIR(s) to beginning of search path for object files
pwd -- Print working directory
reverse-search -- Search backward for regular expression (see regex(3)) from last line listed
search -- Search for regular expression (see regex(3)) from last line listed
section -- Change the base address of section SECTION of the exec file to ADDR
set extension-language -- Set mapping between filename extension and source language
set gnutarget -- Set the current BFD target
sharedlibrary -- Load shared object library symbols for files matching REGEXP
show gnutarget -- Show the current BFD target
symbol-file -- Load symbol table from executable file FILE
(gdb)

GDB 도움말

Running 도움말

- 단계별 실행과 관련된 명령어 모음
- 잘 사용하는 명령어
  - **run:** 디버깅할 프로그램을 시작함. **run** 명령어의 프로그램 인자는 디버깅할 프로그램의 프로그램 인자와 같은 역할을 함.
  - **continue:** 디버깅중인 프로그램의 진행을 계속함.
  - **kill:** 디버깅중인 프로그램의 실행을 정지함.
  - **step:** 다음 줄에 도달할 때까지 프로그램을 진행시킴. 인자 **N**은 이를 **N**번 반복함을 의미함. 함수 호출을 하게 되면 해당 함수로 제어가 넘어감.
  - **next: step** 명령어와 비슷하나, 함수 호출을 하나의 단위로 보아 함수 안을 디버깅할 수 없음. 올바르게 동작하는 함수의 경우 더 이상 디버깅이 필요없을 때 사용하면 편리함.

GDB 도움말

Running 도움말

(gdb) ***help running***
Running the program.
List of commands:
attach -- Attach to a process or file outside of GDB
continue -- Continue program being debugged
detach -- Detach a process or file previously attached
finish -- Execute until selected stack frame returns
handle -- Specify how to handle a signal
info handle -- What debugger does when program gets various signals
interrupt -- Interrupt the execution of the debugged program
jump -- Continue program being debugged at specified line or address
kill -- Kill execution of program being debugged
next -- Step program
nexti -- Step one instruction
run -- Start debugged program
set args -- Set argument list to give program being debugged when it is started
set environment -- Set environment variable value to give the program
set follow-fork-mode -- Set debugger response to a program call of fork or vfork
set scheduler-locking -- Set mode for locking scheduler during execution
set step-mode -- Set mode of the step operation
show args -- Show argument list to give program being debugged when it is started
show follow-fork-mode -- Show debugger response to a program call of fork or vfork
show scheduler-locking -- Show mode for locking scheduler during execution
show step-mode -- Show mode of the step operation
signal -- Continue program giving it signal specified by the argument
step -- Step program until it reaches a different source line
stepi -- Step one instruction exactly
target -- Connect to a target machine or process
thread -- Use this command to switch between threads
thread apply -- Apply a command to a list of threads
apply all -- Apply a command to all threads
tty -- Set terminal for future runs of program being debugged
unset environment -- Cancel environment variable VAR for the program
until -- Execute until the program reaches a source line greater than the current
(gdb)

GDB 도움말

Breakpoints 도움말

- 프로그램을 특정 지점에서 정지하게 하기 위한 명령어 모음
- 잘 사용하는 명령어
  - **break:** 소스 내의 특정 줄이나 특정 함수에 대한 정지점**(breakpoint)**을 설정할 때 사용함**. break** 명령어의 인자로 줄 번호가 지정되면 해당 줄 번호에 정지점이 설정되며**,** 인자로 함수 이름이 지정되면 함수의 시작 부분에 정지점이 설정됨**.**
  - **watch:** 변수 등에 대한 경계점**(watchpoint)**을 설정할 때 사용됨**.** 지정된 경계점의 값이 변경될 경우 프로그램 실행을 정지함**.**
  - **clear:** 특정 줄이나 특정 함수에 설정된 정지점을 해제할 때 사용함**.**
  - **delete:** 정지점 등을 해제할 때 사용함**. clear**와 달리 정지점의 번호를 인자로 사용함**.**

GDB 도움말

Breakpoints 도움말

(gdb) ***help breakpoints***
Making program stop at certain points.
List of commands:
awatch -- Set a watchpoint for an expression
break -- Set breakpoint at specified line or function
catch -- Set catchpoints to catch events
clear -- Clear breakpoint at specified line or function
commands -- Set commands to be executed when a breakpoint is hit
condition -- Specify breakpoint number N to break only if COND is true
delete -- Delete some breakpoints or auto-display expressions
disable -- Disable some breakpoints
enable -- Enable some breakpoints
hbreak -- Set a hardware assisted breakpoint
ignore -- Set ignore-count of breakpoint number N to COUNT
rbreak -- Set a breakpoint for all functions matching REGEXP
rwatch -- Set a read watchpoint for an expression
tbreak -- Set a temporary breakpoint
tcatch -- Set temporary catchpoints to catch events
thbreak -- Set a temporary hardware assisted breakpoint
watch -- Set a watchpoint for an expression
(gdb)

GDB 도움말

Data 도움말

- 변수, 메모리의 정보를 검색하고 설정하는 명령어 모음
- 잘 사용하는 명령어
  - **display:** 프로그램 실행이 정지될 때마다 특정 수식의 값을 자동적으로 보여줌. 디버깅할 때 특정 변수의 값이 변동하는 것을 관찰할 때 편리함.
  - **delete display: display** 명령어에 의하여 지정된 수식을 제거함.
  - **print:** 해당 수식의 값을 보여줌. **display** 명령어와 달리 일회성의 성격을 띰.
  - **set variable:** 특정 변수의 값을 설정하는데 사용함. 디버깅하면서 특정 변수의 값을 변화시켜 가면서 프로그램의 변화를 살펴보는데 유용함.
  - **whatis:** 해당 수식의 데이터 형을 보여줌.

GDB 도움말

Data 도움말

(gdb) ***help data***
Examining data.
List of commands:
call -- Call a function in the program
delete display -- Cancel some expressions to be displayed when program stops
delete mem -- Delete memory region
disable display -- Disable some expressions to be displayed when program stops
disable mem -- Disable memory region
disassemble -- Disassemble a specified section of memory
display -- Print value of expression EXP each time the program stops
enable display -- Enable some expressions to be displayed when program stops
enable mem -- Enable memory region
inspect -- Same as "print" command
mem -- Define attributes for memory region
output -- Like "print" but don't put in value history and don't print newline
print -- Print value of expression EXP
printf -- Printf "printf format string"
ptype -- Print definition of type TYPE
set -- Evaluate expression EXP and assign result to variable VAR
set variable -- Evaluate expression EXP and assign result to variable VAR
undisplay -- Cancel some expressions to be displayed when program stops
whatis -- Print data type of expression EXP
x -- Examine memory: x/FMT ADDRESS
(gdb)

GDB 도움말

Info 도움말

- 디버깅 중인 프로그램에 대한 여러 정보를 보여주는 명령어 모음
- 잘 사용하는 명령어
  - **info args:** 프로그램 인자에 대한 정보를 보여줌.
  - **info breakpoints:** 지정된 정지점에 대한 정보를 보여줌.
  - **info display: display** 명령어에 의하여 지정된 수식 정보를 보여줌.
  - **info files:** 디버깅중인 프로그램에 대한 정보를 보여줌.
  - **info locals:** 현재 스택 프레임내 지역 변수에 대한 정보를 보여줌.
  - **info types:** 현재 정의된 데이터형에 대한 정보를 보여줌.

GDB 도움말

Info 도움말

(gdb) help ***info***
Generic command for showing things about the program being debugged.
List of info subcommands:
info address -- Describe where symbol SYM is stored
info all-registers -- List of all registers and their contents
info args -- Argument variables of current stack frame
info breakpoints -- Status of user-settable breakpoints
info catch -- Exceptions that can be caught in the current stack frame
info common -- Print out the values contained in a Fortran COMMON block
info copying -- Conditions for redistributing copies of GDB
info dcache -- Print information on the dcache performance
info display -- Expressions to display when program stops
info extensions -- All filename extensions associated with a source language
info files -- Names of targets and files being debugged
info float -- Print the status of the floating point unit
info frame -- All about selected stack frame
info functions -- All function names
info handle -- What debugger does when program gets various signals
info line -- Core addresses of the code for a source line
info locals -- Local variables of current stack frame
info mem -- Memory region attributes
info proc -- Show /proc process information about any running process
info program -- Execution status of the program
info registers -- List of integer registers and their contents
info remote-process -- Query the remote system for process info
info scope -- List the variables local to a scope
info set -- Show all GDB settings
info sharedlibrary -- Status of loaded shared object libraries
info signals -- What debugger does when program gets various signals
info source -- Information about the current source file
info sources -- Source files in the program
info stack -- Backtrace of the stack
info symbol -- Describe what symbol is at location ADDR
info target -- Names of targets and files being debugged
info terminal -- Print inferior's saved terminal status
info threads -- IDs of currently known threads
info tracepoints -- Status of tracepoints
info types -- All type names
info udot -- Print contents of kernel ``struct user'' for current child
info variables -- All global and static variable names
info warranty -- Various kinds of warranty you do not have
info watchpoints -- Synonym for ``info breakpoints''
(gdb)

프로그램 디버깅

▣ 디버깅할 프로그램 지정 예

```
[cprog2@seps5 ch4]$ gdb
GNU gdb Red Hat Linux (5.2.1-4)
Copyright 2002 Free Software Foundation, Inc.
…
(gdb) list
No symbol table is loaded.  Use the "file" command.
(gdb) file dbg_first          // file 명령어를 사용한 프로그램 지정
Reading symbols from dbg_first...done.
(gdb) list
1       /*
2        * 4장 디버깅과 오류 처리
3        * 파일 이름: dbg_first.c
4        */
5
6       #include <stdio.h>
7
8       main()
9       {
10          printf("Hello, World.\n");
(gdb) run
Starting program: /home/cprog2/book2/ch4/dbg_first
Hello, World.

Program exited with code 016.
(gdb)
```

프로그램 디버깅

▣ 디버깅할 인자 지정 예

```
1 /*
2  * 4장 디버깅과 오류 처리
3  * 파일 이름: dbg_func.c
4  */
5
6 #include <stdio.h>
7
8 int my_sum(int k)
9 {
10    int i;
11    int sum = 0;
12
13    for (i=1; i<=k; i++)
14       sum += i;
15
16    return sum;
17 }
18
```

```
19 main(int argc, char *argv[])
20 {
21    int i;
22    int k;
23    int sum = 0;
24
25    if (argc < 2) {
26       fprintf(stderr, "Usage: dbg_func 3\n");
27       exit(1);
28    }
29
30    k = atoi(argv[1]);
31    sum = my_sum(k);
32
33    printf("sum = %d\n", sum);
34 }
```

```
[cprog2@seps5 ch4]$ gcc -g dbg_func.c -o dbg_func
[cprog2@seps5 ch4]$
```

프로그램 디버깅

Run 명령어 인자 사용

```
[cprog2@seps5 ch4]$ gdb dbg_func
GNU gdb Red Hat Linux (5.2.1-4) …
(gdb) list
14          sum += i;
15
16        return sum;
17      }
18
19      main(int argc, char *argv[])
20      {
21        int i;
22        int k;
23        int sum = 0;
(gdb) break 21
Breakpoint 1 at 0x8048422: file dbg_func.c, line 21.
(gdb) run 3          //  프로그램 인자를 3으로 지정
// dbg_func 프로그램의 프로그램 인자가 3으로 되었음을 알 수 있다.
Starting program: /home/cprog2/book2/ch4/dbg_func 3

Breakpoint 1, main (argc=2, argv=0xbffff044) at dbg_func.c:23
23        int sum = 0;
(gdb) continue          // 프로그램의 실행을 계속 한다.
Continuing.
sum = 6
Program exited with code 010.
(gdb)
```

프로그램 디버깅

set args 명령어로 디버깅할 인자 지정

```
[cprog2@seps5 ch4]$ gdb dbg_func
GNU gdb Red Hat Linux (5.2.1-4) …
(gdb) list
14          sum += i;
15
16        return sum;
17      }
18
19      main(int argc, char *argv[])
20      {
21        int i;
22        int k;
23        int sum = 0;
(gdb) break 21
Breakpoint 1 at 0x8048422: file dbg_func.c, line 21.
(gdb) set args 3
(gdb) run
Starting program: /home/cprog2/book2/ch4/dbg_func 3

Breakpoint 1, main (argc=2, argv=0xbffff344) at
dbg_func.c:23
23        int sum = 0;
```

```
(gdb) info args
argc = 2
argv = (char **) 0xbffff344
(gdb) print argv[0]
$1 = 0xbffffb5d "/home/cprog2/book2/ch4/dbg_func"
(gdb) print argv[1]
$2 = 0xbffffb7d "3"
(gdb) continue
Continuing.
sum = 6

Program exited with code 010.
(gdb)
```

프로그램 디버깅

- 단계별 실행
  - 각 줄 또는 함수 단위로 동작 상태를 추적
  - 명령어
    - **step** // 한 단계 실행. 함수 호출시 함수 내부 디버깅 가능.
    - **step n**// **n**번 **step** 명령어 실행
    - **Next** // 한 단계 실행. **step** 명령어와 달리 함수 호출도 한 단계로 처리.
    - **next n**// **n**번 **next** 명령어 실행

프로그램 디버깅

Step, next args 명령어로 디버깅

```
[cprog2@seps5 ch4]$ gdb dbg_func
GNU gdb Red Hat Linux (5.2.1-4) …
(gdb) list 1,100
….
30        k = atoi(argv[1]);
31        sum = my_sum(k);
…
(gdb) br 31      // break 명령어. 명령어의 철자 일부를 입력해도
                 // 다른 명령어와 충돌이 발생하지 않으면 인식함.
Breakpoint 1 at 0x8048465: file dbg_func.c, line 31.
(gdb) run 3
Starting program: /home/cprog2/book2/ch4/dbg_func 3

Breakpoint 1, main (argc=2, argv=0xbfffe344) at dbg_func.c:31
31        sum = my_sum(k);    // 다음 실행은 my_sum() 함수와 관련
(gdb) step       // step 명령어에 의해 my_sum() 함수 내부로 이동
my_sum (k=3) at dbg_func.c:11
11        int sum = 0;
(gdb) step
13        for (i=1; i<=k; i++)
(gdb) step
14          sum += i;
(gdb) ↵      // 특정 명령어의 반복 실행시 엔터키만 누르면 됨.
13        for (i=1; i<=k; i++)
(gdb) ↵      // step 명령어의 반복 실행
14          sum += i;
(gdb) ↵      // step 명령어의 반복 실행
13        for (i=1; i<=k; i++)
(gdb) ↵      // step 명령어의 반복 실행
14          sum += i;
(gdb) ↵      // step 명령어의 반복 실행
13        for (i=1; i<=k; i++)
(gdb) ↵      // step 명령어의 반복 실행
16        return sum;
(gdb) ↵      // step 명령어의 반복 실행
17      }
```

```
(gdb) ↵      // step 명령어의 반복 실행
// my_sum() 함수 실행을 마치고 main() 함수로 제어가 복귀.
main (argc=2, argv=0xbfffdbc4) at dbg_func.c:33
33        printf("sum = %d\n", sum);
(gdb) ↵      // step 명령어의 반복 실행
sum = 6
34      }
(gdb) ↵      // step 명령어의 반복 실행
0x4003b56d in __libc_start_main () from /lib/libc.so.6
(gdb) ↵      // step 명령어의 반복 실행
Single stepping until exit from function __libc_start_main,
which has no line number information.

Program exited with code 010.        // 프로그램 디버깅 종료
(gdb) run 10       // 디버깅을 다시 하기 위해서 run 명령어 실행
Starting program: /home/cprog2/book2/ch4/dbg_func 10

Breakpoint 1, main (argc=2, argv=0xbfffe644) at dbg_func.c:31
31        sum = my_sum(k);
(gdb) next       // step명령과 달리 my_sum() 함수를 1단계로 처리
33        printf("sum = %d\n", sum);    // main() 함수내 다음 줄로
제어 이동됨
(gdb) ↵      // next 명령어의 반복 실행
sum = 55

34      }
(gdb) ↵      // next 명령어의 반복 실행
0x4003b56d in __libc_start_main () from /lib/libc.so.6
(gdb) ↵      // next 명령어의 반복 실행
Single stepping until exit from function __libc_start_main,
which has no line number information.

Program exited with code 011.   // 프로그램 디버깅 종료
(gdb)
```

프로그램 디버깅

▣ 경계점(watchpoint)를 이용한 디버깅

◈ 특정 변수 값의 변화를 추적

◈ 사용 예제 프로그램 소스

```
 1 /*
 2  * 4장 디버깅과 오류 처리
 3  * 파일 이름: dbg_watch.c
 4  */
 5
 6 #include <stdio.h>
 7
 8 main()
 9 {
10    int i;
11    int sum = 0;
12
13    for (i=0; i<3; i++)
14       sum += i;
15
16    printf("sum = %d\n", sum);
17 }
```

```
[cprog2@seps5 ch4]$ gcc -g dbg_watch.c -o dbg_watch
[cprog2@seps5 ch4]$
```

프로그램 디버깅

▣ 경계점(watchpoint)를 이용한 디버깅

```
[cprog2@seps5 ch4]$ gdb dbg_watch
GNU gdb Red Hat Linux (5.2.1-4)
...
(gdb) list 1,100      // 범위를 지정해 프로그램 소스 리스트
1     /*
....
17    }
18
(gdb) break 10
Breakpoint 1 at 0x8048338: file dbg_watch.c, line 10.
(gdb) watch sum           // run 명령어 실행 전에 동작 안함.
No symbol "sum" in current context.
(gdb) run
Starting program: /home/cprog2/book2/ch4/dbg_watch

Breakpoint 1, main () at dbg_watch.c:11
11        int sum = 0;
(gdb) watch sum           // 변수 sum에 대한 경계점 설정
Hardware watchpoint 2: sum
(gdb) info break
Num Type           Disp Enb Address    What
1   breakpoint     keep y   0x08048338 in main at
dbg_watch.c:10
        breakpoint already hit 1 time
2   hw watchpoint  keep y              sum
(gdb) continue
Continuing.
Hardware watchpoint 2: sum

Old value = 1075022328
New value = 0
main () at dbg_watch.c:13
13        for (i=0; i<3; i++)
```

```
(gdb) continue
Continuing.
Hardware watchpoint 2: sum

Old value = 0
New value = 1
main () at dbg_watch.c:13
13        for (i=0; i<3; i++)
(gdb) continue
Continuing.
Hardware watchpoint 2: sum

Old value = 1
New value = 3
main () at dbg_watch.c:13
13        for (i=0; i<3; i++)
(gdb) continue
Continuing.
sum = 3

Watchpoint 2 deleted because the program has left the
block in which its expression is valid.
0x4003b56d in __libc_start_main () from /lib/libc.so.6
(gdb) continue
Continuing.

Program exited with code 010.
(gdb)
```

프로그램 디버깅

프로그램 상태 정보 조회

```
 1 /*
 2  * 4장 디버깅과 오류 처리
 3  * 파일 이름: dbg_data.c
 4  */
 5
 6 #include <stdio.h>
 7
 8 int add(int i, int j)
 9 {
10    int k;
11
12    k = i + j;
13    return k;
14 }
15
16 main()
17 {
18    int m = 2, n = 3;
19    int sum;
20
21    sum = m + n;
22    printf("sum = %d\n", sum);
23
24    sum = add(m, n);
25    printf("sum = %d\n", sum);
26 }
```

프로그램 디버깅

▣ 프로그램 상태 정보 조회

```
[cprog2@seps5 ch4]$ gdb dbg_data
GNU gdb Red Hat Linux (5.2.1-4)
...
(gdb) list 1,100      // 프로그램 소스 리스트
1     /*
....
26    }
(gdb) break 18
Breakpoint 1 at 0x804834c: file dbg_data.c, line 18.
(gdb) run
Starting program:
/home/cprog2/book2/ch4/dbg_data

Breakpoint 1, main () at dbg_data.c:18
18        int m = 2, n = 3;
(gdb) step
21        sum = m + n;
(gdb) print m       // 변수 m의 값 출력
$1 = 2
(gdb) print n       // 변수 n의 값 출력
$2 = 3
(gdb) print sum     // 변수 sum의 값 출력
$3 = 134513318
(gdb) step
22        printf("sum = %d\n", sum);
(gdb) print sum     // 변수 sum의 값 출력
$4 = 5
(gdb) info locals   // 현재 스택 프레임에 있는 지역 변수 정보 출력
m = 2
n = 3
sum = 5
(gdb) step
sum = 5
24        sum = add(m, n);
(gdb) set variable m=10   // 변수 m의 값을 10으로 설정
(gdb) display sum     // 변수 sum의 값을 한 단계씩 디버깅할 때마다 자동으로 출력
1: sum = 5
(gdb) step        // 함수 add()로 제어가 넘어감.
      // 변수 sum은 add()내에서 의미없으므로 출력 안됨.
add (i=10, j=3) at dbg_data.c:12
12        k = i + j;
(gdb) info locals   // 함수 add()내의 지역 변수 정보가 출력됨.
k = 1075020672
(gdb) step
13        return k;
(gdb) ↵
14    }
(gdb) ↵
main () at dbg_data.c:25        // main()으로 제어 돌아옴.
25        printf("sum = %d\n", sum);
1: sum = 13                 // display 명령 결과
          // 변수 sum이 의미있어지므로 자동 출력됨.
(gdb) ↵
sum = 13                // 25번째 줄의 printf() 결과
26    }
1: sum = 13             // display 명령 결과
(gdb) ↵
0x4003b56d in __libc_start_main () from /lib/libc.so.6
(gdb) ↵
Single stepping until exit from function
__libc_start_main, which has no line number
information.
Program exited with code 011.
(gdb)
```

프로그램 디버깅

▣ 여러 개의 분산 프로그램

```
 1 /*
 2  * 4장 디버깅과 오류 처리
 3  * 파일 이름: dbg_main.c
 4  */
 5
 6 #include <stdio.h>
 7
 8 extern int my_sum(int k);  // cf. dbg_file1.c
 9
10 main(int argc, char *argv[])
11 {
12    int i;
13    int k;
14    int sum = 0;
15
16    if (argc < 2) {
17       fprintf(stderr, "Usage: dbg_main 3\n");
18       exit(1);
19    }
20
21    k = atoi(argv[1]);
22    sum = my_sum(k);
23
24    printf("sum = %d\n", sum);
25 }
```

```
 1 /*
 2  * 4장 디버깅과 오류 처리
 3  * 파일 이름: dbg_file1.c
 4  */
 5
 6 #include <stdio.h>
 7
 8 int my_sum(int k)
 9 {
10    int i;
11    int sum = 0;
12
13    for (i=1; i<=k; i++)
14       sum += i;
15
16    return sum;
17 }
```

```
[cprog2@seps5 ch4]$ gcc –g -c dbg_file1.c -o dbg_file1.o
[cprog2@seps5 ch4]$ gcc –g -c dbg_main.c -o dbg_main.o
[cprog2@seps5 ch4]$ gcc -o dbg_main dbg_main.o dbg_file1.o
[cprog2@seps5 ch4]$
```

프로그램 디버깅

▣ 여러 개의 분산 프로그램

```
[cprog2@seps5 ch4]$ gdb dbg_main
GNU gdb Red Hat Linux (5.2.1-4)
...
(gdb) list 1,100   // main()함수가 있는 dbg_main.c 파일 리스트.

1     /*
....
25    }
(gdb) break 22
Breakpoint 1 at 0x8048433: file dbg_main.c, line 22.
(gdb) run 3
Starting program: /home/cprog2/book2/ch4/dbg_main 3

Breakpoint 1, main (argc=2, argv=0xbfffe2c4) at dbg_main.c:22
22        sum = my_sum(k);
(gdb) step
my_sum (k=3) at dbg_file1.c:11  // dbg_file1.c로 제어 이동
11        int sum = 0;
(gdb) ↵
13        for (i=1; i<=k; i++)
(gdb) ↵
14          sum += i;
(gdb) ↵
13        for (i=1; i<=k; i++)
(gdb) ↵
14          sum += i;
(gdb) ↵
13        for (i=1; i<=k; i++)
(gdb) ↵
14          sum += i;
(gdb) ↵
13        for (i=1; i<=k; i++)
(gdb) ↵
16        return sum;
(gdb) ↵
17    }
(gdb) ↵
main (argc=2, argv=0xbfffe2c4) at dbg_main.c:24   // dbg_main.c로 제어 돌아옴.
24        printf("sum = %d\n", sum);
(gdb) ↵
sum = 6
25    }
(gdb) ↵
0x4003b56d in __libc_start_main () from /lib/libc.so.6
(gdb) ↵
Single stepping until exit from function __libc_start_main, which has no line number information.

Program exited with code 010.
(gdb)
```

GDB 용 GUI

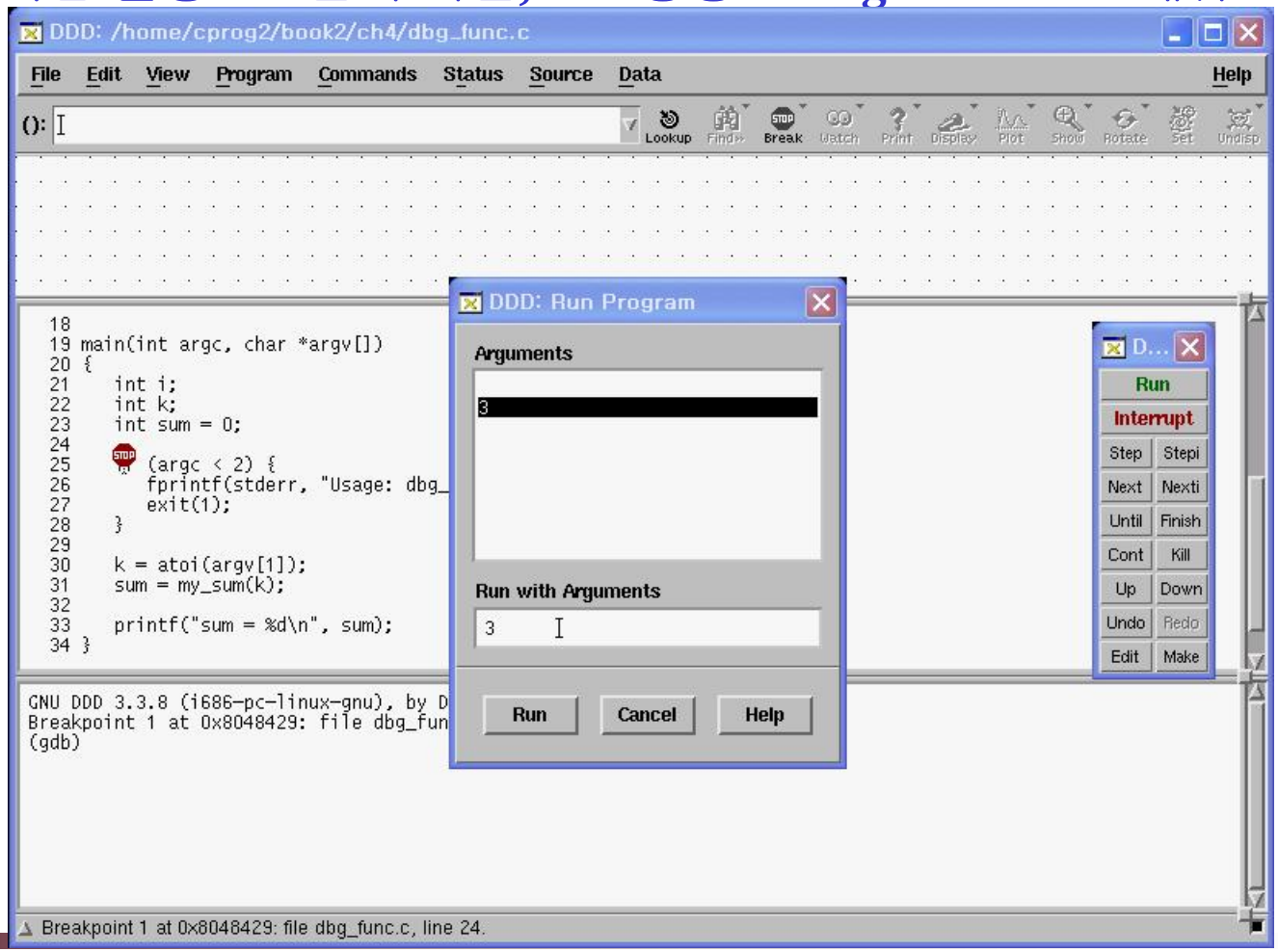
- 디버거의 종류
  - 텍스트 기반 디버거 – **GDB** 등
  - **GUI** 기반 디버거
    - 보다 쉽게 프로그램 디버기 가능
    - **Visual C++(MS Windows)**
    - **DDD(Data Display Debugger), Insight, KDBG**

- **DDD (Data Display Debugger)**
  - 다양한 커맨드라인 디버거를 지원하는 그래픽 사용자 인터페이스
  - **GDB, DBX, WDB, JDB, XDB, Perl Debugger** 등 지원
  - http://www.gnu.org/software/ddd/

GDB 용 GUI

▣ DDD 사용 예 - 예제 4의 dbg_func.c 프로그램 디버깅

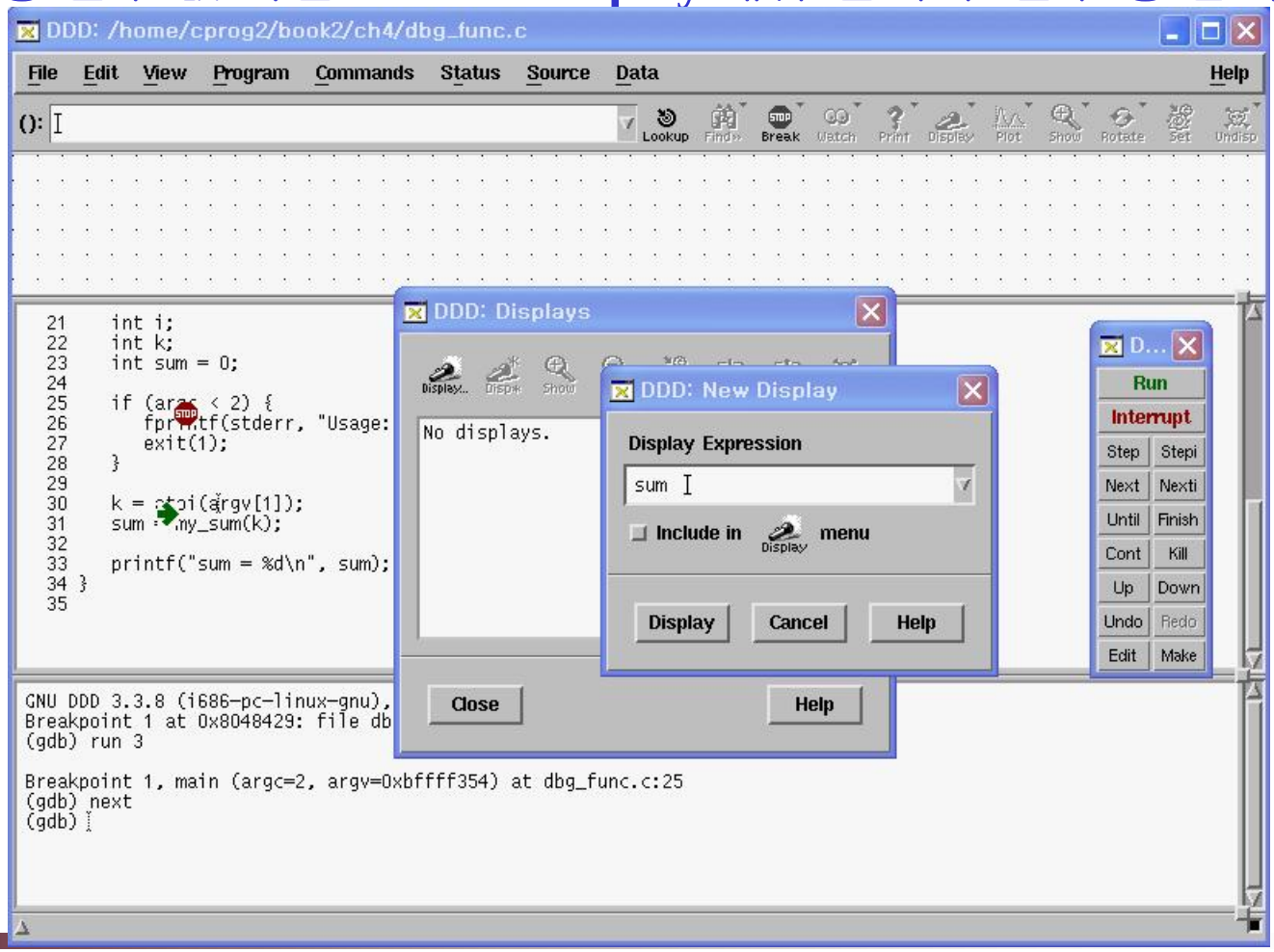
◈ 정지점 설정 : 오른쪽 버튼, run 명령 : Program→Run 메뉴



GDB 용 GUI

▣ DDD 사용 예

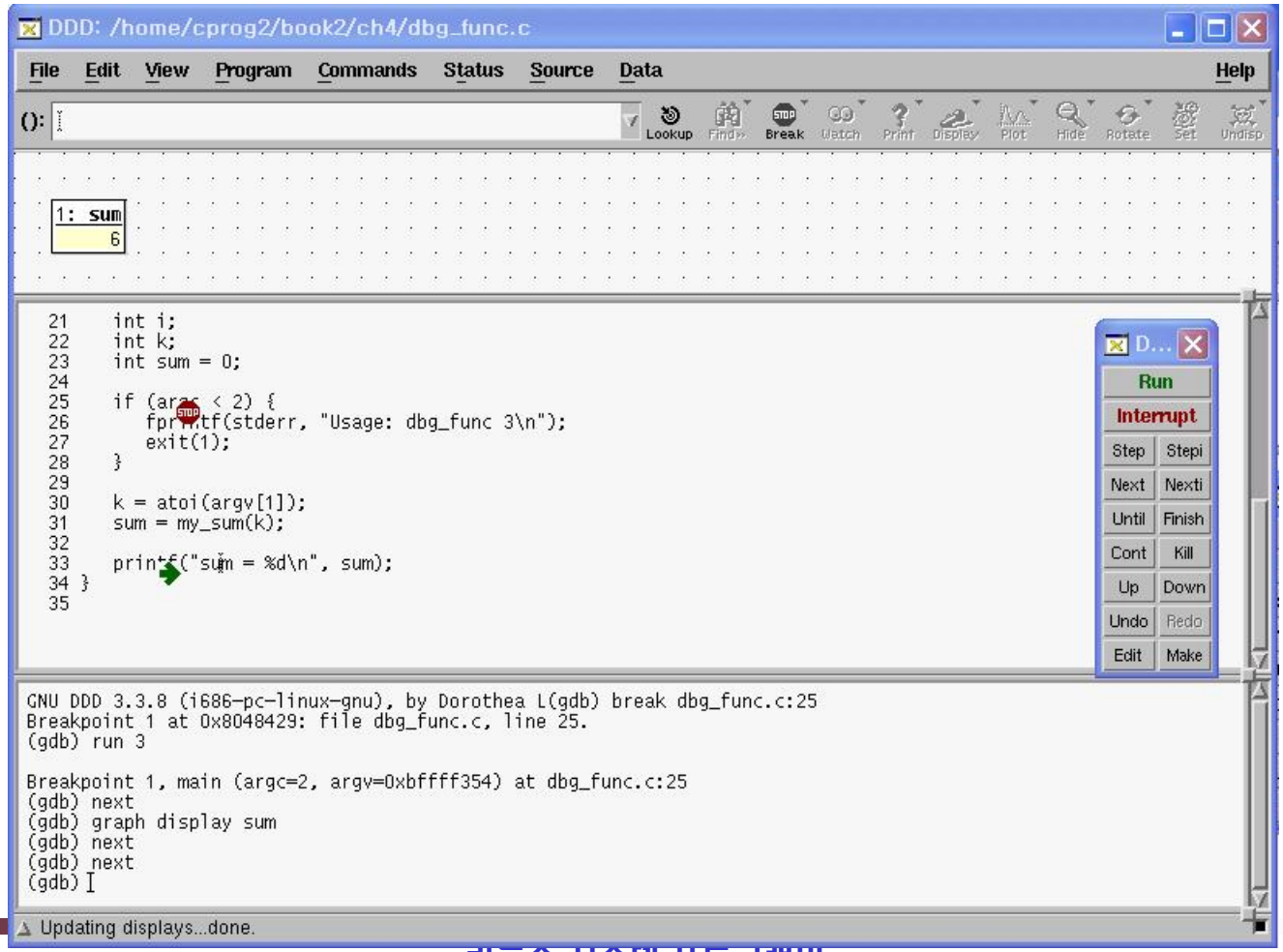
◈ 특정 변수 값 확인 : **Data→Display** 메뉴 선택 후 변수 명 입력



GDB 용 GUI

▣ DDD 사용 예

◈ 단계적 실행 : Step, Next, Cont 명령어 버튼



오류 처리

▣ 오류번호 사용

◈ 많은 라이브러리 함수들이 오류 시 오류번호를 반환

◈ 해당 오류 정보 제공

◈ **<errno.h> 파일 참조 : /usr/include/asm/errno.h**

	오류번호 errno
형식	**#include <errno.h>** **extern int errno;**
기능	가장 최근의 오류값을 나타냄.
오류값	오류값의 예는 다음과 같다. EBADF 잘못된 파일 기술자 EINPROGRESS 진행중인 오퍼레이션 EINVAL 잘못된 프로그램 인자 ENOENT 없는 파일 또는 디렉토리 ERANGE 결과값이 너무 큼 ETIMEDOUT 시간 초과된 오퍼레이션

오류 처리

오류번호 예제 프로그램

```
 1 /*
 2  * 4장 디버깅과 오류 처리
 3  * 파일 이름: errno_show.c
 4  */
 5
 6 #include <stdio.h>
 7 #include <stdlib.h>
 8 #include <errno.h>
 9
10 main(int argc, char *argv[])
11 {
12    FILE *f;
13
14    if (argc < 2) {
15       printf("Usage: errno_show file_name\n");
16       exit(1);
17    }
18
19    if ( (f = fopen(argv[1], "r")) == NULL ) {
20        printf("Cannot open a file \"%s\"... (errno: %d)\n", argv[1], errno);
21       exit(1);
22    }
23
24    printf("Open a file \"%s\".\n", argv[1]);
25
26    fclose(f);
27 }
```

```
[cprog2@seps5 ch4]$ errno_show nofilename
Cannot open a file "nofilename"... (errno: 2)
```

오류 처리

strerror 처리

◈ 발생 오류 번호 **errno** 에 대해 **strerror()** 함수를 사용하여 오류 원인 출력 가능

	오류원인출력 : strerror()
형식	**#include <string.h>** **char *strerror(int errnum);**
기능	오류 원인을 설명하는 문자열을 출력한다.
반환값	인자 errnum에 해당하는 문자열을 반환하거나, 알지 못하는 오류의 경우 unknown error 메시지를 반환한다.

오류 처리

strerror 예제 프로그램

```
 1 /*
 2  * 4장 디버깅과 오류 처리
 3  * 파일 이름: strerror_use.c
 4  */
 5
 6 #include <stdio.h>
 7 #include <stdlib.h>
 8 #include <string.h>
 9 #include <errno.h>
10
11 main(int argc, char *argv[])
12 {
13    FILE *f;
14
15    if (argc < 2) {
16       printf("Usage: strerror_use nofilename\n");
17       exit(1);
18    }
19
20    if ( (f = fopen(argv[1], "r")) == NULL ) {
21       printf("Cannot open a file \"%s\"... (error message: %s)\n",
22             argv[1], strerror(errno));
23       exit(1);
24    }
25
26    printf("Open a file \"%s\".\n", argv[1]);
27
28    fclose(f);
29 }
```

```
[cprog2@seps5 ch4]$ strerror_use  nofilename
Cannot open a file "nofilename"… (error message: No such file or directory)
```

오류 처리

perrno 처리

- **perror()** 함수는 가장 최근의 오류 원인을 문자열로 출력해 줌
- **Perror()** 함수 사용 시 오류 발생 함수를 사용하여 오류 발생 지점을 명시하는 것이 좋다.

	시스템 오류 메시지 출력 : perrno()
형식	**#include <stdio.h>** **void perror(const char *s);**
기능	문자열 s가 NULL이 아니면 문자열 s 다음에 ":" 와 가장 최근의 오류 원인을 설명하는 문자열을 함께 출력한다. 문자열 s가 NULL이면 가장 최근의 오류 원인을 설명하는 문자열만 출력한다.
반환값	없음.

오류 처리

perror 예제 프로그램

```
 1 /*
 2  * 4장 디버깅과 오류 처리
 3  * 파일 이름: perror_use.c
 4  */
 5
 6 #include <stdio.h>
 7 #include <stdlib.h>
 8 #include <string.h>
 9 #include <errno.h>
10
11 main(int argc, char *argv[])
12 {
13    FILE *f;
14
15    if (argc < 2) {
16       printf("Usage: perror_use nofilename\n");
17       exit(1);
18    }
19
20    if ( (f = fopen(argv[1], "r")) == NULL ) {
21       perror("fopen");    // perror(NULL)로 대체하여 실행해 보자.
22       exit(1);
23    }
24
25    printf("Open a file \"%s\".\n", argv[1]);
26
27    fclose(f);
28 }
```

```
[cprog2@seps5 ch4]$ perror_use  nofilename
fopen: No such file or directory
```

오류 처리

▣ 조건부 오류 처리

◈ 특정 수식을 이용한 오류 조건을 검사하여 거짓이면, 오류 메시지를 출력하고 프로그램 종료

	조건부 종료 : assert()
형식	**#include <assert.h>** **void assert(int expression);**
기능	만약 수식 expression이 거짓이면, 표준 출력 stdout으로 오류 메시지를 출력하고 abort()를 호출해 프로그램을 종료한다. 만약 NDEBUG가 정의되어 있으면 실행되지 않는다.
반환값	없음.

오류 처리

조건부 종료 예제 프로그램

```
 1 /*
 2  * 4장 디버깅과 오류 처리
 3  * 파일 이름: assert_test.c
 4  */
 5
 6 #include <stdio.h>
 7 #include <assert.h>
 8 #include <stdlib.h>
 9
10 void foo(int num)
11 {
12    assert( ((num >= 0) && (num <= 100)) );
13
14    printf("foo: num = %d\n", num);
15 }
16
17 main(int argc, char *argv[])
18 {
19    int num;
20
21    if (argc < 2) {
22        fprintf(stderr, "Usage: assert_test aNumber\n(0 <= aNumber <=
100)\n");
23      exit(1);
24    }
25
26    num = atoi(argv[1]);
27    foo(num);
28 }
```

```
[cprog2@seps5 ch4]$ assert_test 10
foo: num = 10
[cprog2@seps5 ch4]$ assert_test 1234
assert_test: assert_test.c:12: foo: Assertion `((num >= 0)
&& (num <= 100))' failed.
중지됨
```