



# Debugger

---

Presented by 李明璋

2012/05/08

---





# The Definition of Bug

---

- Part of the code which would result in an error, fault or malfunctioning of the program.





# Common Bugs

---

- Bugs with pointers and memory
  - Memory leaks
    - The allocated memory is not freed subsequently.
  - Free the already freed resource

```
int main(){
    char* str = ( char* ) malloc( sizeof(char) * 10 );
    if ( global == 0 ) free( str );
    /* Doing Something here ... */
    free ( str );
    return 0;
}
```





# Common Bugs

---

- Bugs with pointers and memory
  - Memory leaks
    - The allocated memory is not freed subsequently.
  - Free the already freed resource
  - NULL-pointer dereferencing
    - Improper initialization

```
int main(){  
    char* str = NULL;  
    strcpy( str, "def" );  
    printf( "%s", str );  
    return 0;  
}
```

```
int main(){  
    char* str = "abc";  
    strcpy( str, "def" );  
    printf( "%s", str );  
    return 0;  
}
```





# Common Bugs

---

- Error of scanf()

```
int main(){
    int num;
    char* str = (char*)malloc(sizeof(char) * SIZE);
    scanf("%d", &num);    // require to pass address to scanf()
    scanf("%s", &str);    // Not need & here, str points to variable itself
    printf("%d %s\n", num, str);
    return 0;
}
```





# Common Bugs

---

- Using '=' instead of '=='

```
int main(){
    int i;
    int a = 0;
    for(i = 0; i < 10; i++) a += i;
    if(a = 0) a = 1000;
    printf("%d", a);
    return 0;
}
```

Output:

0





# Common Bugs

---

- Loop error

```
int x = 5;
while(x > 0); → Infinite loop
    x--;
```

Infinite loop



```
int main() {
    int a = 0;
    while(a < 10){
        printf("%d\n", a);
        if (a = 5) printf("a equals 5!\n");
        a++;
    }
    return 0;
}
```





# Debugger

---

- A debugger is a computer program used to test and debug other programs.
- The code to be examined might alternatively be running on an instruction set simulator (ISS), a technique that allows great power in its ability to halt when specific conditions are encountered.







# Debugger (GDB)

---

- GDB – GNU Debugger
- Compiler your program with ‘-g’ for debugging
  - gcc -g [your program] -o [executable file]

```
lmc66@cm118:~/DSA/GDB$ gcc -g hello.c -o hello
```





# Debugger (GDB)

- Start GDB
  - gdb
  - gdb [executable file]

```
lmc66@cml18:~/DSA/GDB$ gdb hello
GNU gdb (GDB) 7.4-debian
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /auto/master00/lmc66/DSA/GDB/hello...done.
(gdb) █
```

↓  
GDB window





# Debugger (GDB)

- Load your program on GDB
  - file [executable file]
- Run your program
  - start (開始執行程式，並停留在main入口處)
  - run (開始執行程式，直到遭遇break point)

```
Reading symbols from /auto/master00/lmc66/DSA/GDB/hello...done.  
(gdb) start  
Temporary breakpoint 1 at 0x40067f: file hello.c, line 38.  
Starting program: /auto/master00/lmc66/DSA/GDB/hello  
  
Temporary breakpoint 1, main () at hello.c:38  
38      struct Node* head = (struct Node*)malloc(sizeof(struct Node));  
(gdb)
```

Stop at the  
beginning of 'main'





# Debugger (GDB)

- Set breakpoints
  - break [location]
  - break [function]

The command “**list**” is used to list part of the code of your program.

```
(gdb) list
39     head = NULL,
40
41     int input;
42     while(scanf("%d", &input)!=EOF){
43         struct Node* new = (struct Node*)malloc(sizeof(struct Node));
44         new->next = NULL;
45         new->data = input;
46         head = InsertSorted(head, new);
47     }
48
(gdb) break 43
Breakpoint 1 at 0x400697: file hello.c, line 43.
(gdb) run
Starting program: /auto/master00/lmc66/DSA/GDB/hello
1
Breakpoint 1, main () at hello.c:43
43     struct Node* new = (struct Node*)malloc(sizeof(struct Node));
(gdb) █
```

Stop at the line 43





# Debugger (GDB)

---

- Set breakpoints
  - break [function / location] if [condition]

```
(gdb) break InsertSorted if head->data < new->data  
Breakpoint 3 at 0x4005b0: file hello.c, line 10.
```

Stop at the function  
'InsertSorted' if (head->data  
< new->data)





# Debugger (GDB)

- Print the value of some variables

- print [variable]

Print the value of variable 'new->data', and the value is 10.

```
(gdb) break 50
Breakpoint 1 at 0x40069b: file InsertSort.c, line 50.
(gdb) run
Starting program: /auto/master00/lmc66/DSA/GDB/InsertSort
10
Breakpoint 1, main () at InsertSort.c:50
50      head = InsertSorted(head, new);
(gdb) print new->data
$1 = 10
(gdb)
```

- Continue executing your program

- continue (c)

Continue executing

```
(gdb) start
Temporary breakpoint 1 at 0x40067f: file hello.c, line 38.
Starting program: /auto/master00/lmc66/DSA/GDB/hello

Temporary breakpoint 1, main () at hello.c:38
38      struct Node* head = (struct Node*)malloc(sizeof(struct Node));
(gdb) c
Continuing.
```





# Debugger (GDB)

- Set watchpoints
  - watch [variable]

```
Temporary breakpoint 1, main () at hello.c:38
38      struct Node* head = (struct Node*)malloc(sizeof(struct Node));
(gdb) watch head
Hardware watchpoint 2: head
(gdb) continue
Continuing.
Hardware watchpoint 2: head
Old value = (struct Node *) 0x0
New value = (struct Node *) 0x601010
main () at hello.c:39
39      head = NULL;
(gdb) █
```

The value of variable  
'head' has been changed.





# Debugger (GDB)

- Set value of variable
  - set [variable] = [value]

```
(gdb) run
Starting program: /auto/master00/lmc66/DSA/GDB/hello
10
Breakpoint 1, InsertSorted (head=0x0, new=0x601030) at hello.c:10
10     struct Node* traverse = head;
(gdb) c
Continuing.
25
Breakpoint 1, InsertSorted (head=0x601030, new=0x601050) at hello.c:10
10     struct Node* traverse = head;
(gdb) set new->data = 1
(gdb) c
Continuing.
10
1
```

Set the value of new->data as 1.







# Debugger (GDB)

- List breakpoints and watchpoints
  - list break

```
(gdb) info break
Num    Type          Disp Enb Address          What
2      breakpoint    keep y 0x0000000000400697 in main at hello.c:43
3      breakpoint    keep y 0x00000000004005b0 in InsertSorted at hello.c:10
4      hw watchpoint keep y                               head
(gdb)
```

The list of all breakpoints and watchpoints.

- Delete breakpoint or watchpoint
  - delete [point number]

Delete the watchpoint which Num is 4.

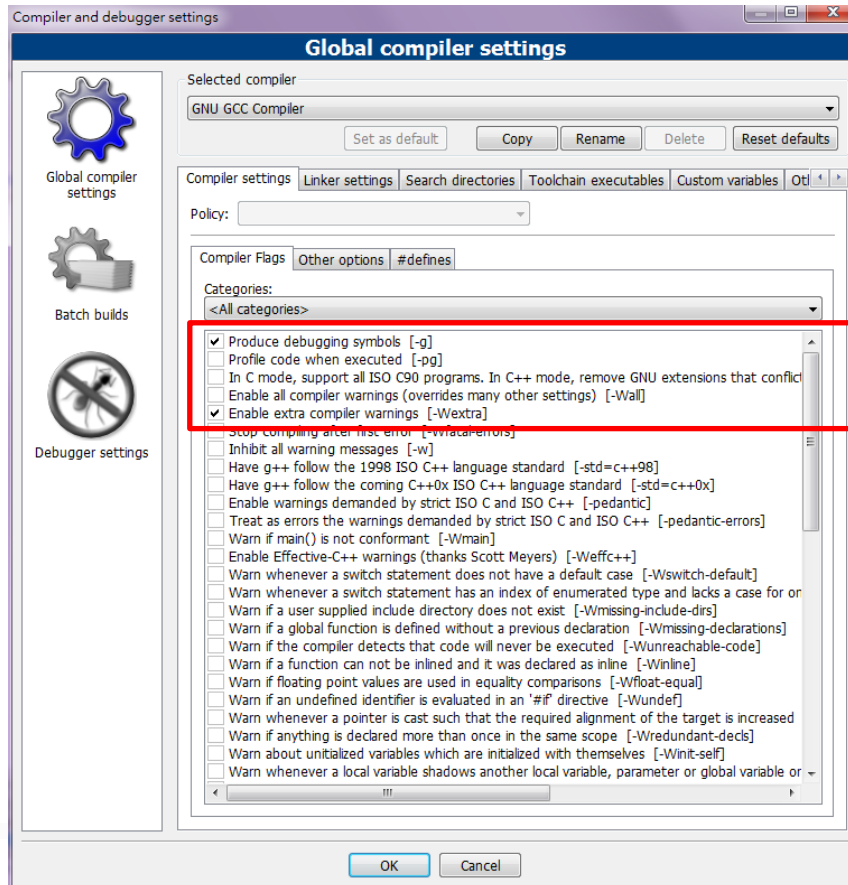
```
(gdb) delete 4
(gdb) info break
Num    Type          Disp Enb Address          What
2      breakpoint    keep y 0x0000000000400697 in main at hello.c:43
3      breakpoint    keep y 0x00000000004005b0 in InsertSorted at hello.c:10
(gdb)
```





# Debugger (Code Blocks)

- Settings → Compiler and debugger...





# Debugger (Code Blocks)

- Add breakpoint

The screenshot shows the Code::Blocks IDE interface. The title bar reads "main.c [testing] - Code::Blocks 10.05". The menu bar includes File, Edit, View, Search, Project, Build, Debug, wxSmith, Tools, Plugins, Settings, and Help. The toolbar contains various icons for file operations and debugging. The "Build target" dropdown is set to "Debug". The "Management" panel on the left shows a project tree with "Workspace", "testing", "Sources", and "main.c". The main editor window displays the following C code:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 int main(){
6     int i, j = 0;
7     char* str = "Hello DSA!";
8     for(i = 0; i < (int)strlen(str); i++){
9         j += i * i;
10    }
11    printf("%d\n", j);
12    return 0;
13 }
14
```

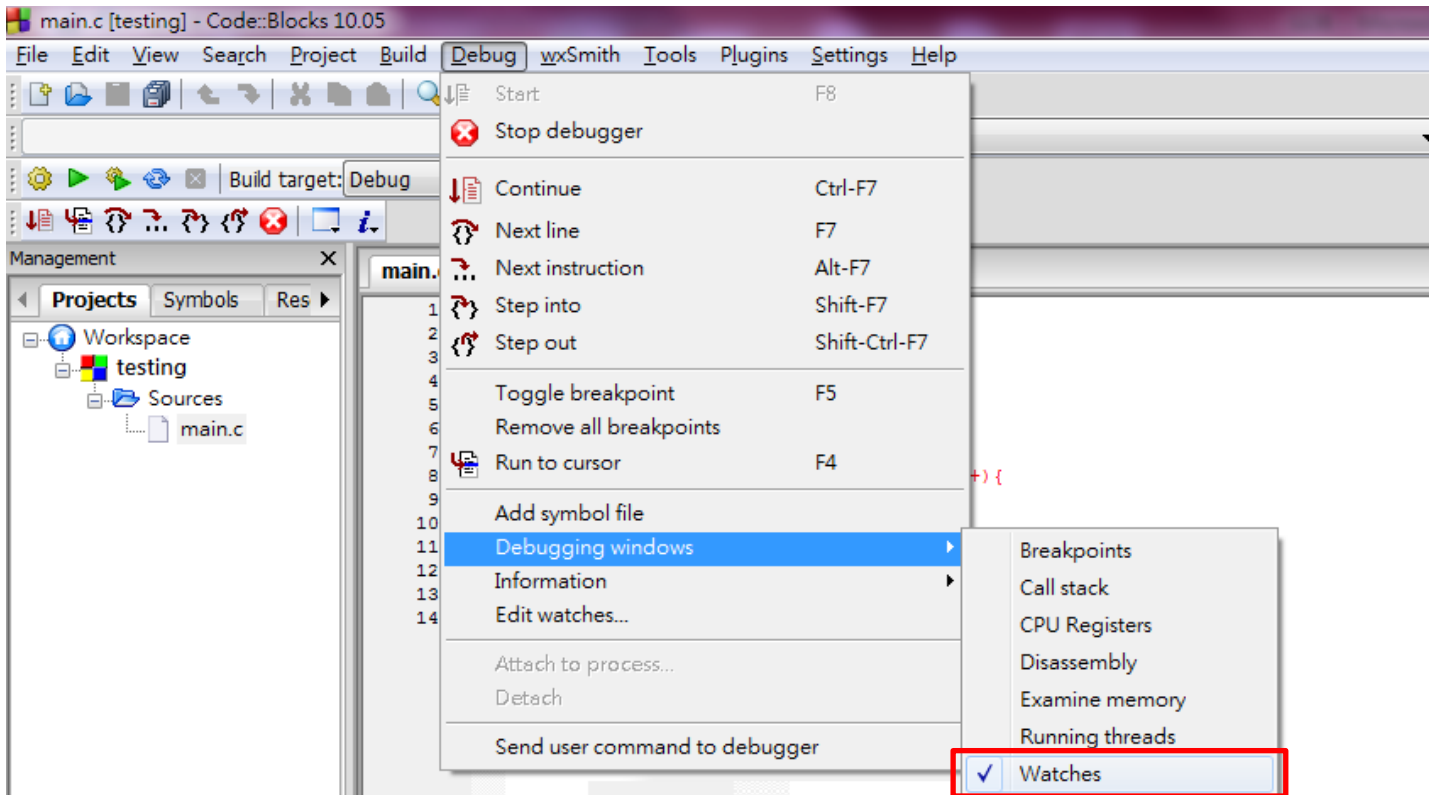
A red box highlights line 9, where a red dot indicates a breakpoint has been set. The dropdown menu at the top right shows "main() : int".





# Debugger (Code Blocks)

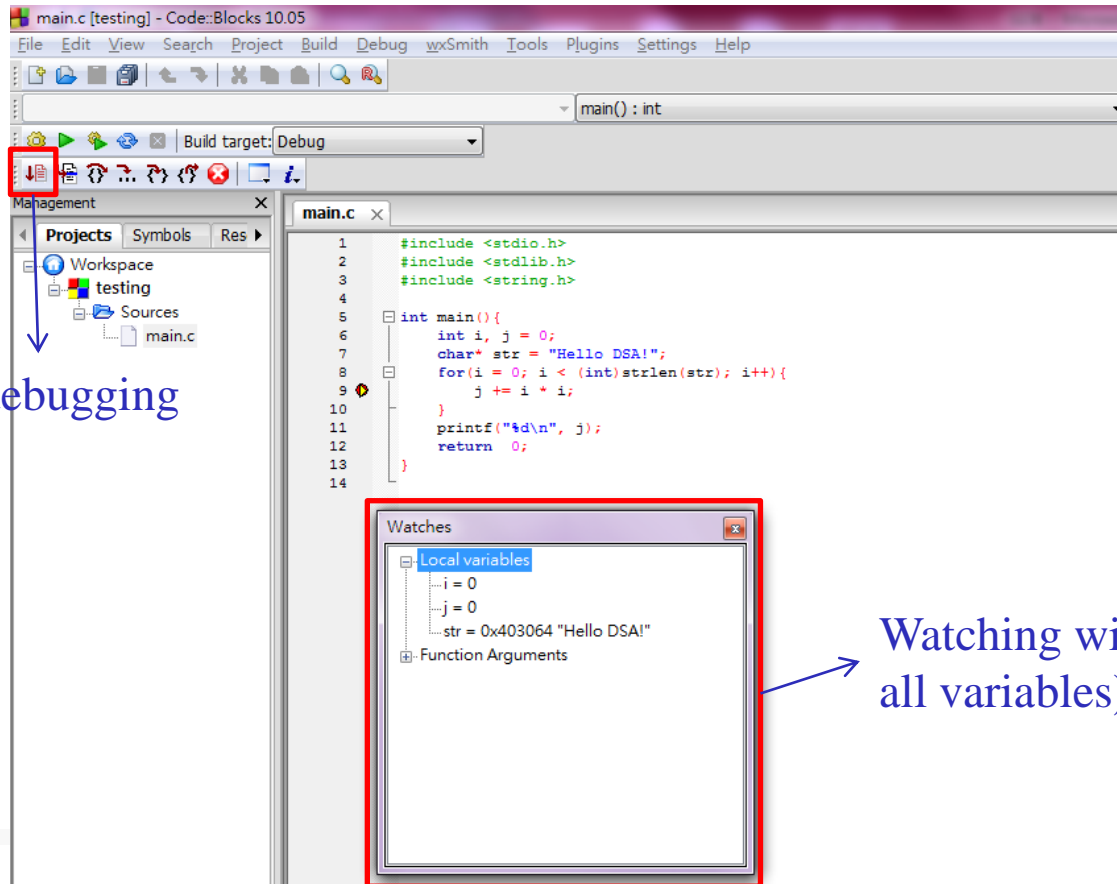
- Open debugging window





# Debugger (Code Blocks)

- Start debugging



Press to start debugging

Watching window (the state of all variables)





# References

---

- GDB manual
  - <http://sourceware.org/gdb/current/onlinedocs/gdb>
- Code::Blocks Debugger manual
  - <http://ez2learn.com/index.php/c-tutorials/codeblocks-tutorials/206-codeblocks-debugger>
- Dev C++ Debugger manual
  - <http://ez2learn.com/index.php/c-tutorials/dev-c-/203-dev-cdebugger>

