

## **Lab 4: Debugging and Pointers**

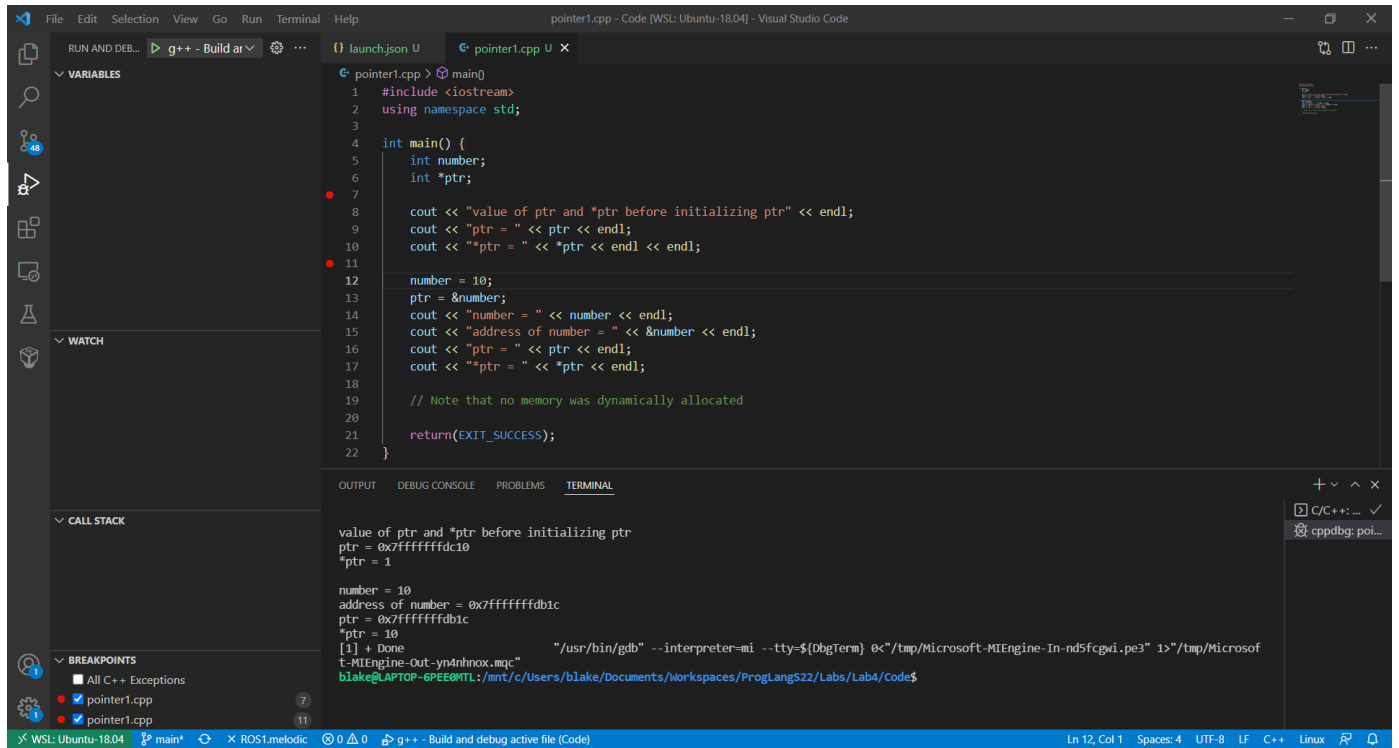
Blake Berry

Bagley College of Engineering, Mississippi State University

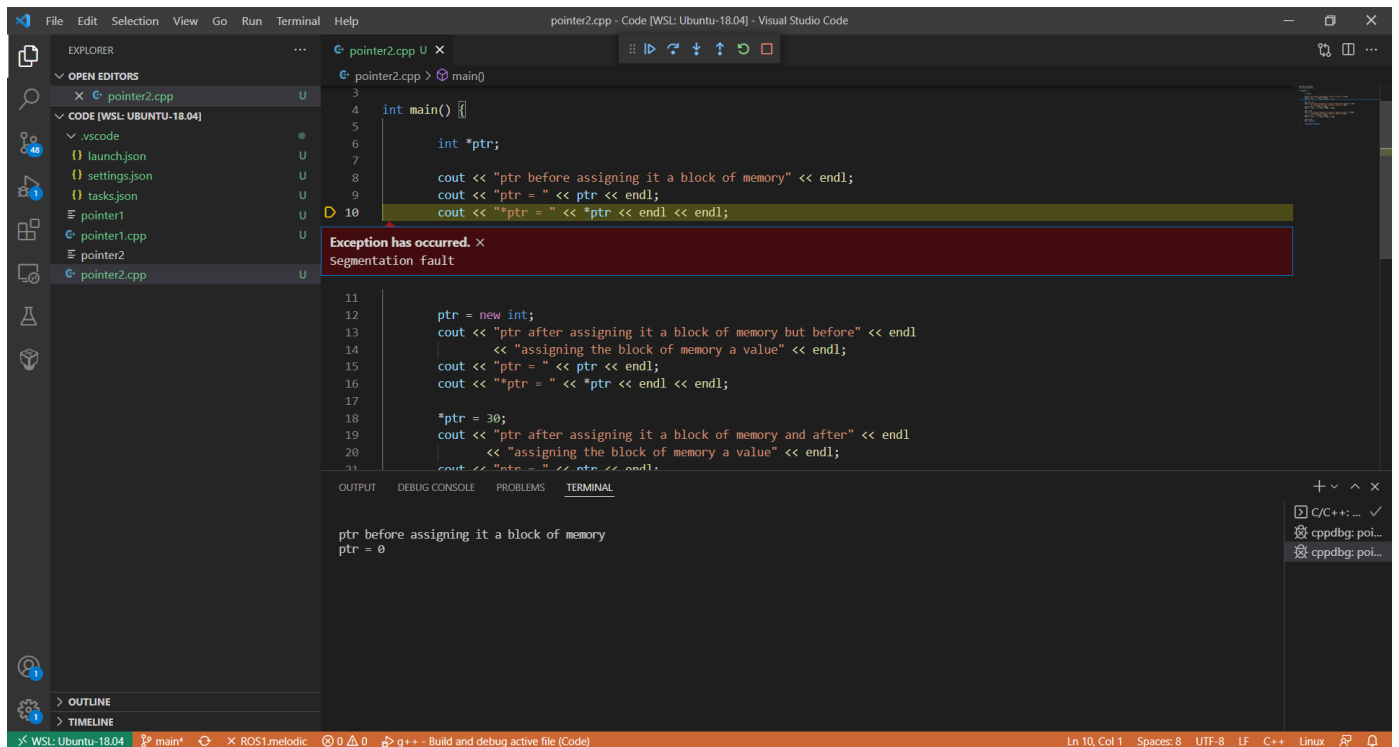
4714: Programming Languages

April 29, 2022

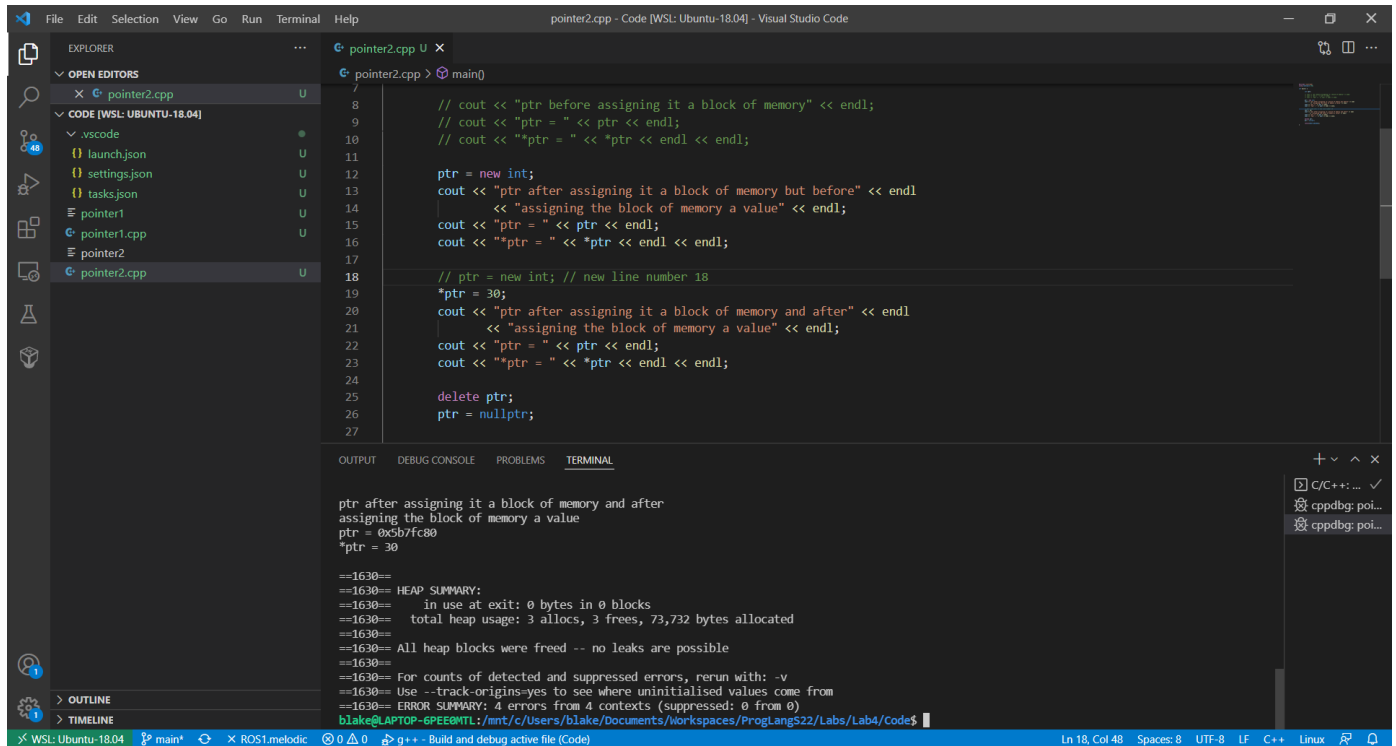
## Debugging: Pointer1 with Printing Garbage



## Debugging: Pointer2 with Segment Fault



## Valgrind: Pointer2 with No Memory Leak



The screenshot shows a Visual Studio Code editor with a C++ file named `pointer2.cpp` open. The file contains a program that allocates memory, assigns a value, and then frees it. The code is as follows:

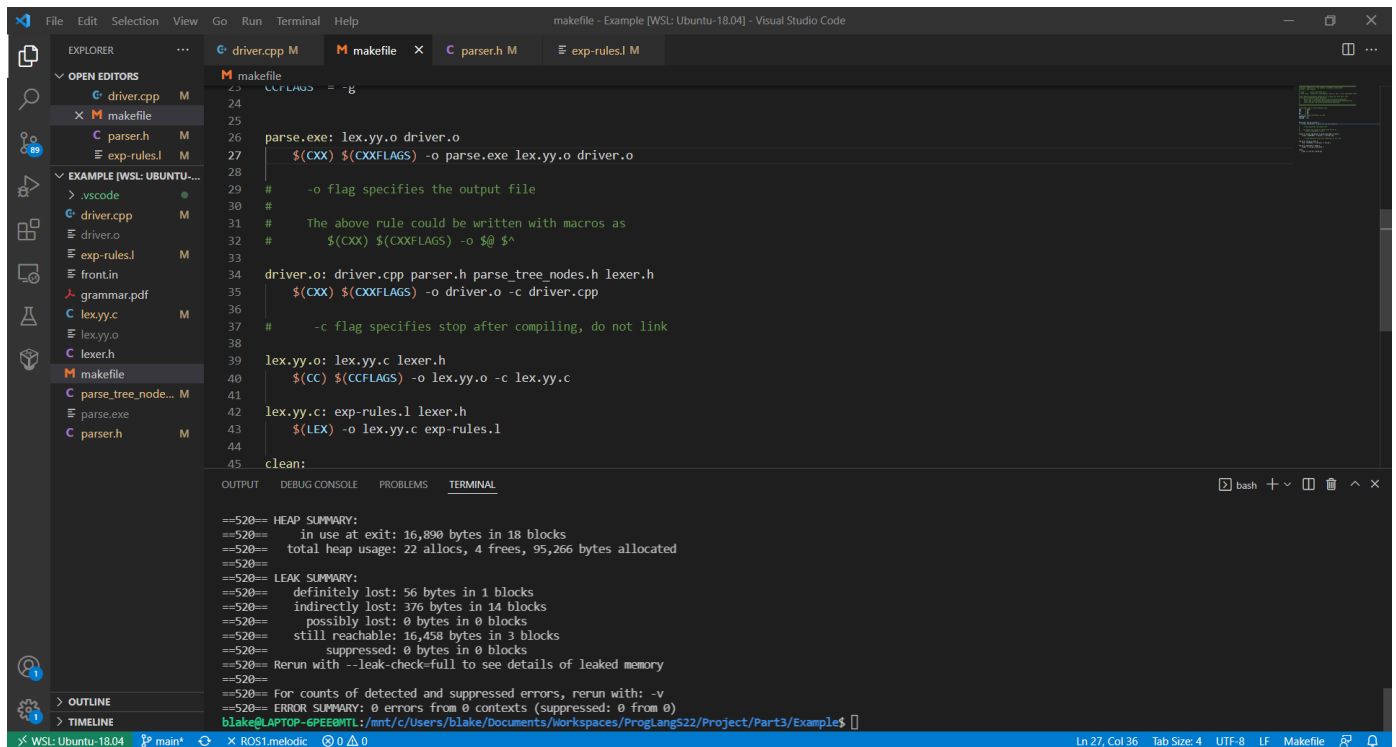
```
1 //
2 // cout << "ptr before assigning it a block of memory" << endl;
3 // cout << "ptr = " << ptr << endl;
4 // cout << "ptr = " << *ptr << endl << endl;
5
6 ptr = new int;
7 cout << "ptr after assigning it a block of memory but before" << endl
8 << "assigning the block of memory a value" << endl;
9 cout << "ptr = " << ptr << endl;
10 cout << "ptr = " << *ptr << endl << endl;
11
12 // ptr = new int; // new line number 18
13 *ptr = 30;
14 cout << "ptr after assigning it a block of memory and after" << endl
15 << "assigning the block of memory a value" << endl;
16 cout << "ptr = " << ptr << endl;
17 cout << "ptr = " << *ptr << endl << endl;
18
19 delete ptr;
20 ptr = nullptr;
21
```

The terminal output shows the program's execution and the Valgrind summary:

```
ptr after assigning it a block of memory and after
assigning the block of memory a value
ptr = 0x5b7fc80
*ptr = 30

==1630==
==1630== HEAP SUMMARY:
==1630==   in use at exit: 0 bytes in 0 blocks
==1630== total heap usage: 3 allocs, 3 frees, 73,732 bytes allocated
==1630==
==1630== All heap blocks were freed -- no leaks are possible
==1630==
==1630== For counts of detected and suppressed errors, rerun with: -v
==1630== Use --track-origins=yes to see where uninitialised values come from
==1630== ERROR SUMMARY: 4 errors from 4 contexts (suppressed: 0 from 0)
blake@LAPTOP-6PEE8MTL: /mnt/c/Users/blake/Documents/workspaces/ProgLang522/Labs/Lab4/Code$
```

## Valgrind: Project Part 3 Example with Memory Leak



The screenshot shows a Visual Studio Code editor with a `makefile` open. The file contains the following rules:

```
1 CCFLAGS = -g
2
3 parse.exe: lex.yy.o driver.o
4 $(CXX) $(CXXFLAGS) -o parse.exe lex.yy.o driver.o
5
6 # -o flag specifies the output file
7 #
8 # The above rule could be written with macros as
9 # $(CXX) $(CXXFLAGS) -o $@ $^
10
11 driver.o: driver.cpp parser.h parse_tree_nodes.h lexer.h
12 $(CXX) $(CXXFLAGS) -o driver.o -c driver.cpp
13
14 # -c flag specifies stop after compiling, do not link
15
16 lex.yy.o: lex.yy.c lexer.h
17 $(CC) $(CCFLAGS) -o lex.yy.o -c lex.yy.c
18
19 lex.yy.c: exp-rules.l lexer.h
20 $(LEX) -o lex.yy.c exp-rules.l
21
22 clean:
```

The terminal output shows the program's execution and the Valgrind summary:

```
==520==
==520== HEAP SUMMARY:
==520==   in use at exit: 16,890 bytes in 18 blocks
==520== total heap usage: 22 allocs, 4 frees, 95,266 bytes allocated
==520==
==520== LEAK SUMMARY:
==520==   definitely lost: 56 bytes in 1 blocks
==520==   indirectly lost: 376 bytes in 14 blocks
==520==   possibly lost: 0 bytes in 0 blocks
==520==   still reachable: 16,458 bytes in 3 blocks
==520==   suppressed: 0 bytes in 0 blocks
==520== Rerun with --leak-check=full to see details of leaked memory
==520==
==520== For counts of detected and suppressed errors, rerun with: -v
==520== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
blake@LAPTOP-6PEE8MTL: /mnt/c/Users/blake/Documents/workspaces/ProgLang522/Project/Part3/Example$
```