



UNIVERSITY OF WATERLOO  
FACULTY OF ENGINEERING  
Department of Electrical & Computer Engineering

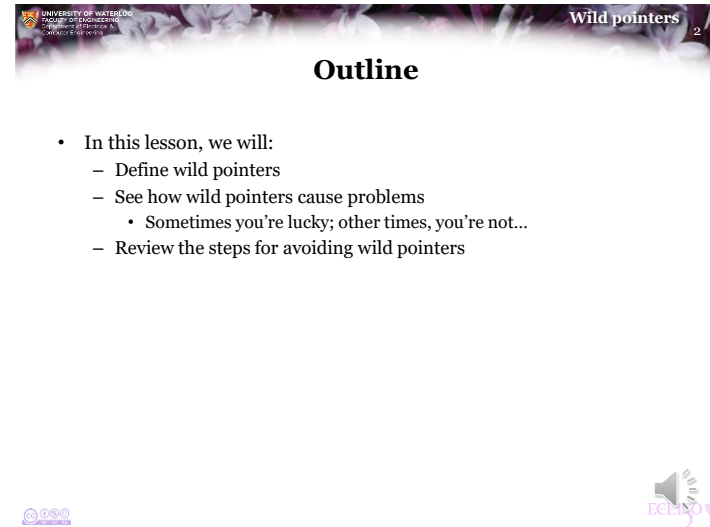
ECE 150 *Fundamentals of Programming*

# Wild pointers

CC BY NC SA

Douglas Wilhelm Harder, M.Math. LEL.  
Prof. Hiren Patel, Ph.D., P.Eng.  
Prof. Werner Dieltz, Ph.D.

© 2018 by Douglas Wilhelm Harder and Hiren Patel. All rights reserved.



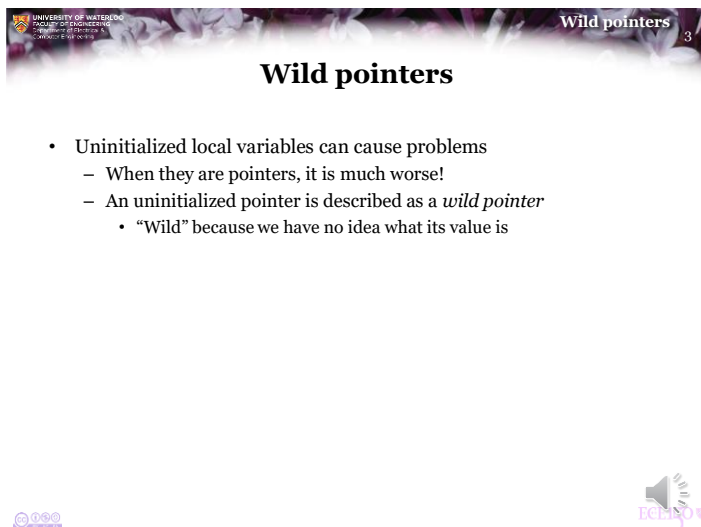
UNIVERSITY OF WATERLOO  
FACULTY OF ENGINEERING  
Department of Electrical & Computer Engineering

Wild pointers 2

## Outline

- In this lesson, we will:
  - Define wild pointers
  - See how wild pointers cause problems
    - Sometimes you're lucky; other times, you're not...
  - Review the steps for avoiding wild pointers

CC BY NC SA



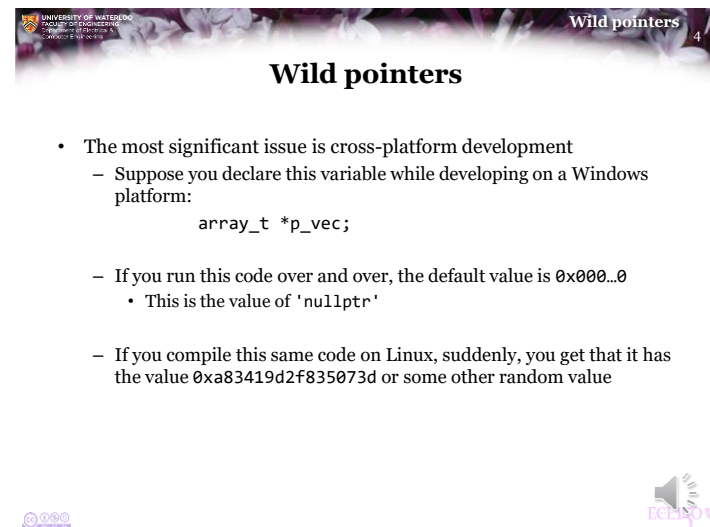
UNIVERSITY OF WATERLOO  
FACULTY OF ENGINEERING  
Department of Electrical & Computer Engineering

Wild pointers 3

## Wild pointers

- Uninitialized local variables can cause problems
  - When they are pointers, it is much worse!
  - An uninitialized pointer is described as a *wild pointer*
    - "Wild" because we have no idea what its value is

CC BY NC SA



UNIVERSITY OF WATERLOO  
FACULTY OF ENGINEERING  
Department of Electrical & Computer Engineering

Wild pointers 4

## Wild pointers

- The most significant issue is cross-platform development
  - Suppose you declare this variable while developing on a Windows platform:
 

```
array_t *p_vec;
```
  - If you run this code over and over, the default value is `0x000...0`
    - This is the value of `'nullptr'`
  - If you compile this same code on Linux, suddenly, you get that it has the value `0xa83419d2f835073d` or some other random value

CC BY NC SA

UNIVERSITY OF WATERLOO  
FACULTY OF ENGINEERING  
FACULTY OF INFORMATION TECHNOLOGY

## Wild pointers, if you are lucky...

- Consider this code:

```
#include <iostream>
int main();
void f();

int main() {
    std::cout << "In main()..." << std::endl;
    f();
    return 0;
}

void f() {
    int *p_value;
    std::cout << "In f(): " << p_value << std::endl;
    *p_value = 42;
}
```

Output:

```
In main()...
In f(): 0x7f51abb16b9e
Segmentation fault (core dumped)
```



UNIVERSITY OF WATERLOO  
FACULTY OF ENGINEERING  
FACULTY OF INFORMATION TECHNOLOGY

## Wild pointers, if you are lucky...

- Whatever memory is at 0x7f51abb16b9e, it is almost certainly not assigned to your program
  - When you tried to access it, the operating system terminated your program
  - Your program cannot access memory not allocated by the operating system to your program



UNIVERSITY OF WATERLOO  
FACULTY OF ENGINEERING  
FACULTY OF INFORMATION TECHNOLOGY

## Wild pointers, if you aren't lucky...

- Consider this code:

```
#include <iostream>
int main();
void init();
void no_init();

int main() {
    init();
    no_init();

    return 0;
}
```



UNIVERSITY OF WATERLOO  
FACULTY OF ENGINEERING  
FACULTY OF INFORMATION TECHNOLOGY

## Wild pointers, if you aren't lucky...

```
void init() {
    int *p_value{new int{42}};
    std::cout << "In init(): " << p_value
              << std::endl;
    // Use 'p_value'...
    delete p_value;
    // Why bother setting 'p_value' to nullptr?
}

void no_init() {
    int *p_local;
    std::cout << "In no_init(): " << p_local
              << std::endl;
    *p_local = 91;
}
```

Output:

```
In init(): 0x1ca3010
In no_init(): 0x1ca3010
Your results may differ...
```





## Wild pointers, if you aren't lucky...

- Why did this code execute?
  - In the first function, the local variable was assigned new memory:
 

```
int *p_value{new int{42}};
```
  - At the end of the function, the memory was deallocated but not reset to 'nullptr'
    - After all, the variable is immediately going out of scope
 

```
delete p_value;
```
    - The operating system is, however, lazy and leaves that memory allocated to you as you may request more memory in the future
  - In the second function, the local variable occupies the same location in memory on the call stack
    - The value from the first function call is still there



## Avoiding wild pointers

- To avoid wild pointers
  - Always initialize all pointers, even if it is to 'nullptr'
 

```
int *p_value{};
int *p_value{nullptr};
```
  - Benefit:
    - Assigning to the memory location 0x0 will always terminate your program, guaranteed
  - When memory is being deallocated, always set the pointer to 'nullptr' even if the local variable is going out of scope
 

```
delete p_value;
p_value = nullptr;
```



## Summary

- Following this lesson, you now
  - Understand that wild pointers are uninitialized pointers
  - Know that accessing wild pointers will result in two consequences:
    - The operating system may terminate your program, or
    - You may use memory that has been allocated to you for other purposes
  - Understand that:
    - The first problem is easy to track down
    - The second is difficult or impossible to track
  - Know that you must always initialize all pointers and set them to 'nullptr' after any deallocation



## References

- [1] Wikipedia: [https://en.wikipedia.org/wiki/Dangling\\_pointer](https://en.wikipedia.org/wiki/Dangling_pointer)
- This page also covers wild pointers





## Colophon

These slides were prepared using the Georgia typeface. Mathematical equations use Times New Roman, and source code is presented using Consolas.

The photographs of lilacs in bloom appearing on the title slide and accenting the top of each other slide were taken at the Royal Botanical Gardens on May 27, 2018 by Douglas Wilhelm Harder. Please see

<https://www.rbg.ca/>

for more information.



## Disclaimer

These slides are provided for the ECE 150 *Fundamentals of Programming* course taught at the University of Waterloo. The material in it reflects the authors' best judgment in light of the information available to them at the time of preparation. Any reliance on these course slides by any party for any other purpose are the responsibility of such parties. The authors accept no responsibility for damages, if any, suffered by any party as a result of decisions made or actions based on these course slides for any other purpose than that for which it was intended.

