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

ECE 150 *Fundamentals of Programming*

The call stack

ECE150

Douglas Wilhelm Harder, M.Math.
Prof. Hiren Patel, Ph.D.
Prof. Werner Dietl, Ph.D.

Outline

- In this lesson, we will:
  - Describe instructions and constants
  - Give an overview of main memory
  - Look at how a program is loaded into main memory
    - Observe where instructions, constants and local variables are stored
  - Learn about the *call stack* to store local variables

Main memory

- Up to now, we have authored, compiled and executed programs
  - Question: How does this work?

Example

- What happens to a program when it is compiled?

```
#include <iostream>

int main();

int main() {
    double x{};
    std::cout << "Enter a value 'x': ";
    std::cin >> x;

    double pi{3.1415926535897932};

    double result{x - pi};

    if ( result < 0 ) {
        result = -result;
    }

    std::cout << "|x - pi| = " << result << std::endl;

    return 0;
}
```

Example

- A program is converted into a sequence of instructions that the computer can execute
  - These instructions are stored in a *file* within a *file system*
    - They are often called *executable files* or *executables*
  - File systems are generally stored in persistent memory
    - A hard-disk drive
    - A solid-state drive
    - Some form of optical memory
  - It may also be stored as firmware in flash ROM
  - Each instruction has its own *address* in that memory

Instructions

Address	Instruction (rendered into English)
1954	Assign 'x' the value 0.0
1955	Call routine to print a string at Address 1968
1956	Call routine to input a double, saving the value to 'x'
1957	Assign 'pi' the value at Address 1969
1958	Subtract 'pi' from 'x'
1959	Assign 'result' the previous calculation
1960	If 'result' is not negative, jump to Address 1963
1961	Negate 'result'
1962	Assign 'result' the previous calculation
1963	Call routine to print a string at Address 1970
1964	Call routine to print a double 'x'
1965	Call routine to print an end-of-line character
1966	Set return value to 0
1967	Return
1968	"Enter a value 'x': "
1969	3.1415926535897932
1970	"|x - pi| = "

Main memory

- When you run a program,

  the instructions and constants are loaded into main memory
  - Main memory is volatile
    - It usually disappears when the computer is turned off
  - It is much faster to access values stored in main memory than it is access anything in persistent memory
  - The processor then starts executing one instruction at a time

Main memory

- In main memory, we now have
  - Instructions
  - Constants (literals)

- Question: Where are the local variables stored?

1954	Assign 'x' the value 0.0
1955	Call routine to print a string at Address 1968
1956	Call routine to input a double, saving the value to 'x'
1957	Assign 'pi' the value at Address 1969
1958	Subtract 'pi' from 'x'
1959	Assign 'result' the previous calculation
1960	If 'result' is not negative, jump to Address 1963
1961	Negate 'result'
1962	Assign 'result' the previous calculation
1963	Call routine to print a string at Address 1970
1964	Call routine to print a double 'x'
1965	Call routine to print an end-of-line character
1966	Set return value to 0
1967	Exit
1968	"Enter a value 'x': "
1969	3.1415926535897932
1970	"|x - pi| = "

Local variables

- One idea is to include the memory for the local variables together with the instructions and constants
  - Problem:
    - We would have to reserve memory for all local variables even if it is unlikely that they will ever be used
    - We will not be able to perform recursive algorithms

- The most common strategy is to place local variables elsewhere in memory
  - The most preferred place is at the *end* of main memory

Local variables

- Thus, our main memory is broken into three sections
  - Instructions
  - Constants
  - Local variables (the *call stack*)
- All the memory in between will be used for:
  - Function calls
  - Requests for additional memory

⋮	
60	
61	`result`
62	`pi`
63	`x`

```
0   Assign 'x' the value 0.0
1   Call routine to print a string at Address 13
2   Call routine to access a double, saving to 'x'
3   Assign 'pi' the value at Address 14
4   Subtract 'pi' from 'x'
5   Assign 'result' the previous calculation
6   Test if 'result' is not negative, jump to Address 9
7   Negate 'result'
8   Assign 'result' the previous calculation
9   Call routine to print a string at Address 15
10  Call routine to print a double 'x'
11  Call routine to print an end-of-line character
12  Set return value to 0
13  Exit
14  "Enter an integer 'n': "
15  3.1415926535897932
16  "|n - pi| = "
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61  result
62  pi
63  x
```

The purpose of the call stack

- Function management:
  Tracks function calls and returns, using a last-in–first-out order
- Local storage:
  Stores local variables and parameters for each function call
- Execution order:
  Manages the order of function execution and ensures each function completes before the previous one returns
- Error handling:
  Provides stack traces for debugging by showing the sequence of function calls at the point of error
- Supports recursion:
  Handles recursive function calls by managing each call's state with separate stack frames.
- Memory management:
  Efficiently allocates and reclaims memory for function calls, preventing leaks.

Summary

- Following this lesson, you now:
  - Know programs are a sequences of instructions built by the compiler
  - Understand they must be loaded into main memory to run them
  - Know that instructions and constants are stored in separate blocks
  - Know that local variables are stored at the other end of memory
    - They are stored in what is called the *call stack*
  - Are aware that the remaining memory can also be used by the running program

References

[1] https://en.wikipedia.org/wiki/Call_stack

Acknowledgments

Proof read by Dr. Thomas McConkey and Charlie Liu.

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.