

COMP26020 Programming Languages and Paradigms -- Part 1

---

# C and Memory Safety: Good Practices

# The Problem

- C and C++ have **many benefits** and are still the default languages in several application domains
- C/C++ are also inherently **memory unsafe**

How can we try to avoid as much as possible these dangerous memory errors?

1. Some tools can help (won't fix everything!)
2. **Write good code**

# Tools

- **Enable extra warnings** with compiler flags:
  - `-Wall`
  - `-Wextra`
  - `-pedantic`

More info: <https://gcc.gnu.org/onlinedocs/gcc/Warning-Options.html>

# Tools

- Dynamic Analysis:
  - [Valgrind](#)
  - [Address Sanitizer](#) (ASAN):
    - Add these compiler flags: `-fsanitize=address -fno-omit-frame-pointer`
  - BoundCheck, Purify, etc.
- Static Analysis:
  - [Clang Static Analyser](#)
  - Lint, Coverity, cppcheck, etc.

# Writing Good Code

- The compiler won't catch all mistakes
- The tools have their limitations
- **Writing good code** from the start is super important
  - It will save you a lot of debugging time
  - It will save you from introducing serious security issues

# Undefined Behaviour

## Worst bugs in C/C++: **undefined behaviour**

- "Renders the entire program meaningless if certain rules of the language are violated." (from cppreference.com)
  - program can crash (good!)
  - program can behave weirdly (pretty good!)
  - program can *seem to* behave normally (argh!)
- Common sources include:
  - Reading an uninitialised variable
  - Reading/writing out of the bounds of an array
  - Dereferencing a `NULL (0x0)` pointer
  - Overflow in signed integer arithmetic
  - Dereferencing a freed pointer
  - Freeing a pointer twice
  - etc.

# Array/Buffers Sizes, Integer Overflows

- **Keep track of your arrays' sizes**
- **Be aware of type sizes on the architecture you target to avoid overflows**
  - `sizeof()`
  - Signed overflow: undefined behaviour

# The C standard Library

- Check man pages
- **Never use these functions**, always unsafe:
  - gets (use fgets)
  - getwd (use getcwd)
  - readdir\_r (use readdir)
  - More here: <https://bit.ly/3ef4TBc>

# String Manipulation Functions

- Use the  $n$  methods,
  - `strcpy` -> `strncpy`
  - `sprintf` -> `snprintf`
  - etc.
- Even with the versions with  $n$ , some particularities
  - `strncpy` won't add `\0` at the end of the target buffer

```
char string1[] = "hello, world";
char string2[32] = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx";

strncpy(string2, string1, strlen(string1));
printf("%s\n", string2); // prints "hello, worldxxxxxxxxxxxxxxxxxxxx" 24-good-practices/strncpy-bug.c 
```

# Dynamic Memory Allocation

- Check `malloc` return value
- After `free`, the pointer is invalid
  - Cannot be dereferenced
  - Cannot be **used** (ex: comparison)
- `realloc` returns null upon failure but does not free the old pointer
  - so this: `ptr = realloc(ptr, new_size)` is a leak

# Summary & How to Learn More

- Compile-time errors, easily reproducible runtime crashes and easily detectable program behaviour divergence
  - **Nice bugs**, you can detect them
- Undefined behaviour:
  - **Nasty bugs**, you can fail to notice them and it can lead to serious vulnerabilities
- Solution: write good code + some tools can help
- How to learn more:
  - [https://docs.fedoraproject.org/en-US/Fedora\\_Security\\_Team/1/html/Defensive\\_Coding/index.html](https://docs.fedoraproject.org/en-US/Fedora_Security_Team/1/html/Defensive_Coding/index.html)
  - Secure Coding in C and C++, 2nd edition by Robert C. Seacord

# Feedback form

<https://bit.ly/3CBOipk>

