richardwu.ca

CS 343 Course Notes

Concurrency

Peter Buhr • Fall 2018 • University of Waterloo

Last Revision: September 11, 2018

Table of Contents

Abstract

These notes are intended as a resource for myself; past, present, or future students of this course, and anyone interested in the material. The goal is to provide an end-to-end resource that covers all material discussed in the course displayed in an organized manner. These notes are my interpretation and transcription of the content covered in lectures. The instructor has not verified or confirmed the accuracy of these notes, and any discrepancies, misunderstandings, typos, etc. as these notes relate to course's content is not the responsibility of the instructor. If you spot any errors or would like to contribute, please contact me directly.

1 September 11, 2018

1.1 Advanced control flow

Everything herein pertains to control flow *within* routines:

- Use break guard clauses (early breaks)
- **Avoid** flag variables: instead using an infinite loop with break statements.

  However, in some cases one may use flag variables if absolutely necessary (e.g. memoizing some status that occurs much later and would be hard to modify).
- Use nested control structures with multi-level breaks (with labels)

Rules for gotos:

- No backward breaks/gotos: use a loop's inherent looping capabilities
- No jumping into the middle of code

tl;dr: use gotos for *static* multi-level exit (to simulate labelled breaks/continues).

1.2 Dynamic allocation

Use stack allocation over dynamic allocation whenver possible: e.g. `int arr[size]` as opposed to `int *arr = new int[size]` and `delete [] arr` (although variable-length stack arrays are not part of the C++ standard, use it whenever possible).
However, heap allocation may be necessary if:

- memory needs to persist outside of the scope memory was initialized in
- unbounded input size (e.g. initializing values from `STDIN` into a `vector`)
- array of objects with variable initialization parameters
- when allocation would overflow a small stack

1.3 Control-flow between routines

For *dynamic* multi-level exit (call/return semantics between routines) use a global label variable which is referred to inside subroutines with `goto`s for jumping between multiple function stack frames. Assigning label literals to the variable at various points in time can alter where the subroutines end up jumping to.
`jump_buf`, `setjmp`, `longjmp` initialize, set, and jump to a label variable, respectively, in C.
Traditional approaches to what we described include:

- Return codes. Disadvantage: mixes exception and normal results, and checking code or flag is optional.
- Status flags via global variable (e.g. `errno` in UNIX). Disadvantage: may be modified by other routines (mixed out).
- Fixup routines (or callbacks). Disadvantage: adds overhead with additional function calls.
- `return union`: returning union types (e.g. result or return code). Disadvantage: