Implementation of algorithms and data structures

5. seminar

Jirka Fink

https://ktiml.mff.cuni.cz/~fink/

Department of Theoretical Computer Science and Mathematical Logic
Faculty of Mathematics and Physics
Charles University in Prague

Winter semestr 2025/26

Last change 26. listopadu 2025

Exception handling

Description

Exception handling is the process of responding to the occurrence of exceptions – anomalous or exceptional conditions requiring special processing – during the execution of a program.

Exception handling

Description

Exception handling is the process of responding to the occurrence of exceptions – anomalous or exceptional conditions requiring special processing – during the execution of a program.

Examples of exceptions

- Index out of range of an array
- Retrieve the first element of an empty list
- Searched element does not exist
- Square root of a negative number
- Determine a circle from three collinear points
- File access without sufficient permission
- Error in reading data
- Invalid user input
- Insufficient amount of memory

Exception handling

Methods of handling exceptions

- Ignore, leads to an undefined behavior
  `vector::operator[]`, `list::pop_front`
- Throw an exception, call longjmp
  `vector::at`
- Call a function
  `set_terminate`, signal handling
- Set a status variable, e.g. `errno`
  `fopen`, `scanf`
- Return an invalid value, e.g. NULL, a dummy object
  `find`
- Return a valid value
  `double sqrt(double x){ if(x < 0)return 0; ... }`
- Return a pair of a status and a result
  `pair<bool, int> find(x){ return exist(x)? make_pair(true, search(x)): make_pair(false, 0); }`
- Use goto to an error label
- Terminate the program, e.g. using functions `abort` or `assert`

Exception handling: Improper and rare solutions

Return a valid value

- A calling function cannot recognize an error
- The error is propagated which makes it harder to find

Exception handling: Improper and rare solutions

Return a valid value

- A calling function cannot recognize an error
- The error is propagated which makes it harder to find

Set a status variable

- Hard to manage
- Requires detailed documentation
- Data race conditions

Exception handling: Improper and rare solutions

Return a valid value

- A calling function cannot recognize an error
- The error is propagated which makes it harder to find

Set a status variable

- Hard to manage
- Requires detailed documentation
- Data race conditions

Use goto to an error label

- Can be used only inside a function
- In rare situations, it may simplify a complex if-else statements

Exception handling: Improper and rare solutions

Return a valid value

- A calling function cannot recognize an error
- The error is propagated which makes it harder to find

Set a status variable

- Hard to manage
- Requires detailed documentation
- Data race conditions

Use goto to an error label

- Can be used only inside a function
- In rare situations, it may simplify a complex if-else statements

Call a function

Only for asynchronous or external events, e.g. signal handling.

Exception handling: Invalid arguments in public methods (bugs)

Throw an exception

- Safe
- Checking conditions delays computation
- Requires catching exceptions
- Requires documentation of used exceptions

Exception handling: Invalid arguments in public methods (bugs)

Throw an exception

- Safe
- Checking conditions delays computation
- Requires catching exceptions
- Requires documentation of used exceptions

Assert in a debug mode, ignore in a release mode

- Fast (in release mode)
- May lead to an undefined behavior
- Requires documentation of conditions on arguments

Exception handling: Unavoidable situations

Return an invalid value

Prefer when

- the calling function is expected to handle
- the exception is a natural output (e.g. no element satisfies a given condition)

If an invalid values does not exist, return a pair with a status.

Exception handling: Unavoidable situations

Return an invalid value

Prefer when

- the calling function is expected to handle
- the exception is a natural output (e.g. no element satisfies a given condition)

If an invalid values does not exist, return a pair with a status.

Throw an exception

Prefer when

- the exception may jump through many functions
- the situation is very rare
- many different types of errors may occur

Exception handling: Unavoidable situations

Assert in a debug mode, ignore in a release mode

- Fast
- Easy to debug
- Cause undefined behavior in release mode

Exception handling: Unavoidable situations

Assert in a debug mode, ignore in a release mode

- Fast
- Easy to debug
- Cause undefined behavior in release mode

A general rule

- Write documentation
- Always follow a project policy!

Example of an issue

Consider a function `push_back` inserting into a `vector<MyClass>` when its array is full. Reallocation moves all element into a new memory, but move constructor throws an exception. What the function `push_back` should do?

Exception handling: Exception safety

Example of an issue

Consider a function `push_back` inserting into a `vector<MyClass>` when its array is full. Reallocation moves all element into a new memory, but move constructor throws an exception. What the function `push_back` should do?

Exception safety

- No-throw guarantee: Operations are guaranteed to succeed and satisfy all requirements even in exceptional situations. If an exception occurs, it will be handled internally and not observed by clients.
- Strong exception safety: Operations can fail, but failed operations are guaranteed to have no side effects, so all data retains their original values.
- Basic exception safety: Partial execution of failed operations can cause side effects, but all invariants are preserved and there are no resource leaks (including memory leaks). Any stored data will contain valid values, even if they differ from what they were before the exception.
- No exception safety: No guarantees are made.