7 Debugging

Contents

7.1 Systematic Localization of Errors

What can be done if a program generates wrong results or simply crashes?

1. Formulate a **hypothesis** which part of the program might be wrong.
2. Try to confirm this hypothesis by some test.
   a) If the test succeeds (i. e. hypothesis confirmed): Narrow down the error by additional hypotheses.
   b) If the test fails: Try to find a different hypothesis.

7.1.1 Printing the Program State

example: We assume that the content of a pair of variables becomes inconsistent at a certain position of the source code:

Add a conditional output statement:

```
#ifdef DEBUG
  cerr << "x1 = " << x1 << ", x2 = " << x2 << endl;
#endif
```

- This method is frequently used as a short test.
- Often it is worth leaving those lines in the source code for possible later problems.

7.1.2 Preventing Errors during Implementation

- **Assertions** can be used to validate the status of the program at a certain point:

  → **`assert`** in C and C++ (next section)

- In **test-driven programming**, for each new function one first generates a test case.

  → **`CppUnit`** in the lecture “Tools”

7.2 Assertions

The macro **`assert(condition)`** checks if some condition is fulfilled. It is inherited from C and defined within the header file **`cassert`**:

C++/Assert/assert.cc

```
5  #include <cassert>
6  #include <iostream>
7
8  using namespace std;
9
10 // computes the quotient a/b
11 double divide(int d1, int d2)
12 {
13   assert(d2 != 0);
14   return static_cast<double>(d1) / static_cast<double>(d2);
15 }
16
17 int main()
18 {
19   int d1{3}, d2{0};
20   double q = divide(d1, d2);
21   cout << "q = " << q << endl;
22 }
```

If **`condition`** is equal to zero (i. e. false) a message is written to the standard error device and **`abort()`** is called, terminating the program execution.

The specifics of the message shown depend on the particular library implementation, but according to the standard it shall at least include the **condition** whose assertion failed, the name of the source file, and the line number where it happened.

example:

```
wmai20 ~/ModProg/Lecture/Sources/C++/Assert > assert
assert: assert.cc:13: double divide(int, int): Assertion `d2 != 0' failed.
Aborted (core dumped)
```

This macro is disabled if, at the moment of including **<cassert>**, a macro with the name **NDEBUG** has already been defined.

7.3 Debuggers

For larger programs, errors can usually be located much faster using a **debugger** rather than by throwing **fprintf**s or **cerr**s into the source code.

Example

Consider the following (wrong!) bubblesort program:

Debugging/BUGS/sorter.cc

```
10 void bubsort(int* a, int size)
11 {
12   for (int i{size - 1}; i > 0; --i)
13     for (int j{0}; j < i; ++j)
14       if (a[j] > a[j + 1])
15         swap(a[j], a[j + 1]);
16 }
17
18 int main(int argc, char** argv)
19 {
20   int* a{new int[argc - 1]};
21
22   for (int i{0}; i < argc - 1; ++i)
23     a[i] = atoi(argv[i + 1]);
24
25   bubsort(a, argc);
26
27   for (int i{0}; i < argc - 1; ++i)
28     cout << a[i] << ' ';
29   cout << '\n';
30
31   delete[] a;
32 }
```

Sometimes, it prints wrong results:

```
~/Tools/Lecture/Sources/Debugging/BUGS > sorter 7 6 5 4 3 2
2 3 4 5 6 7
~/Tools/Lecture/Sources/Debugging/BUGS > sorter 7 6 5 4
0 4 5 6
```


7.3.1 GDB

Before we show graphical debuggers, we give a short introduction to **GDB (GNU Project Debugger)**.

URL: https://www.sourceware.org/gdb/

documentation: *Debugging with GDB*, in html- and pdf-format at
https://www.sourceware.org/gdb/documentation/

Examining the Bubblesort Program

- At first, the program has to be compiled with the option **-g** which adds additional debugging information to the binary. Optimization should be switched off.

```
wmai20 …/Tools/Lecture/Sources/Debugging/BUGS > make
…
g++ -Wall -g -c sorter.cc -o sorter_dbg.o
g++ sorter_dbg.o -o sorter_dbg
```

- The debugging session is started with **gdb sorter_dbg**.
- After each GDB command the source code line to be executed next is displayed (if program is run).

- overview of GDB commands:

command	short form	what it does
break 25	**b 25**	set a breakpoint at line 25
break main	**b main**	set a breakpoint at first code line of function **main**
break sorter.cc:main	**b …**	set a breakpoint in the specified source file
break main if argc > 3	**b …**	set a conditional breakpoint
run	**r**	execute program, stop at next breakpoint if present
run 7 6 5	**r 7 6 5**	passing command line parameters
next	**n**	execute next command or complete function call
next 8	**n 8**	next 8 commands
step	**s**	like **next**, but step into functions
until	**u**	run until a source code line with a greater number than the current line is reached (useful to exit loops)
finish	**fin**	run until the current function returns
continue	**c**	continue until next breakpoint or end of program
print *argv[i + 2]	**p …**	print content of a variable or result of an expression
print arr[n]@m	**p …**	print array elements **arr[n]**, …, **arr[n + m - 1]**
display i	**disp i**	print variable or expression after each GDB command
watch a[2]	**wa a[2]**	stop whenever the value of an expression changes
where	**whe**	print current position in all active functions (backtrace)
quit	**q**	quit debugger