

# Programming tools

Linda Marshall

Department of Computer Science  
University of Pretoria

14 September 2021

A programming tool or software development tool is a computer program that software developers use to:

- create,
- debug,
- maintain, and/or
- otherwise support other programs and applications

## Basic tools

- Source code editor
- Compiler
- Interpreter
- Debugger
- Profiler

There are many forms of software tools, for example:

- Debuggers, e.g. the GNU Debugger (GDB) [2]
- Memory leak detection, e.g. Valgrind [3]
- Documentation generators
- Project Management/Code sharing, e.g. GitHub, Stack Overflow.
- Source code editors, either standalone or part of an IDE (e.g. Eclipse)

*NOTE: Some employers will check your GitHub contributions.*

## The GNU Debugger (GDB)

- works with Ada, C, C++, Java, Assembler etc
- On Linux - `sudo apt-get install gdb`
- GDB tutorial: <http://www.gdbtutorial.com/tutorial> (Accessed on 13 Sep 2021)
- [https://www.cs.swarthmore.edu/~newhall/unixhelp/howto\\_gdb.php](https://www.cs.swarthmore.edu/~newhall/unixhelp/howto_gdb.php)

- Compile and build your program with debugging symbols  
`g++ -g main.cpp`  
If you have multiple files, make sure you compile each with the `-g` flag
- Run the program using GDB  
`gdb ./main`
- Use GDB commands to analyse and debug the program
- Exit GDB by typing `quit`

## A few GDB Commands to get you started

- Set a breakpoint
  - Set a breakpoint for main: `break main`
  - Set for a specific file at a specific line: `break filename.c:line`Breakpoints can be deleted, disabled and enabled.
- Run the program: `run`
- Step through the code
  - Step into functions: `s`
  - Step over functions: `n`Adding a number allows you to step the specified number of lines.
- Watchpoints
  - Set watchpoint on a variable: `watch variable`
  - List watchpoints: `info break`
- Interrogate variables
  - Show local variables in current frame: `info locals`
  - Show the contents of a local variable: `p variable`
- List the next few lines: `list`
- Examine memory: `x address`

## “Hello World” on steroids!

```
// Create a type
typedef HelloWorld <HelloWorld_OutputPolicy_WriteToCout ,
                  HelloWorld_LanguagePolicy_English>  my_hello_world_type;

// Create an object of the type
my_hello_world_type hello_world;

// Call the run method of the object
hello_world.run();
```



```
template< typename output_policy , typename language_policy>
class HelloWorld : public output_policy , public language_policy {
    using output_policy::print;
    using language_policy::message;
public:
    //behaviour method
    void run() {
        //two policy methods
        std::string msg = message();
        print( msg );
    }
};
```

```
class HelloWorld_OutputPolicy_WriteToCout {  
protected:  
    template< typename message_type >  
    void print( message_type message ) {  
        std::cout << message << std::endl;  
    }  
};  
  
class HelloWorld_LanguagePolicy_English {  
protected:  
    std::string message() {  
        return "Hello , World!";  
    }  
};
```

- Used for memory debugging, memory leak detection and profiling.
- On Linux - `sudo apt-get install valgrind`
- Video:  
<https://www.youtube.com/watch?v=A5Rc4Awda0A>

- As with GDB, compile the program with the `-g` flag. Memcheck will provide error messages with line numbers.
- Run valgrind with the `--leak-check=yes` flag set

### Example 1 - Simple program

```
int main() {  
    int* value = new int(5);  
  
    cout << *value << endl;  
  
    return 0;  
}
```

### Valgrind result with no flags

```
==58986== HEAP SUMMARY:  
==58986==      in use at exit: 17,741 bytes in 158 blocks  
==58986==    total heap usage: 172 allocs, 14 frees, 22,485 bytes allocated  
==58986==  
==58986== LEAK SUMMARY:  
==58986==      definitely lost: 4 bytes in 1 blocks  
==58986==      indirectly lost: 0 bytes in 0 blocks  
==58986==      possibly lost: 0 bytes in 0 blocks  
==58986==      still reachable: 4,096 bytes in 1 blocks  
==58986==             suppressed: 13,641 bytes in 156 blocks
```

### Example 1 - Simple program

```
int main() {  
    int* value = new int(5);  
  
    cout << *value << endl;  
  
    return 0;  
}
```

### Valgrind result with `-leak-check=yes`

```
...  
==59075==    by 0x100000CF8: main (SimpleProgram.cpp:13)  
==59075==  
==59075== LEAK SUMMARY:  
==59075==    definitely lost: 4 bytes in 1 blocks  
==59075==    indirectly lost: 0 bytes in 0 blocks  
==59075==    possibly lost: 0 bytes in 0 blocks  
==59075==    still reachable: 4,096 bytes in 1 blocks  
==59075==           suppressed: 13,641 bytes in 156 blocks
```

The memory allocated by the variable in line 13 has not been deallocated.

## Example 2 - Simple class

```
template <typename T>
class MyClass {
public:
    MyClass(T val) {
        value = new T(val);
    }
    friend ostream &operator<<(ostream &output,
                               const MyClass<T> &val ) {
        output << val;
        return output;
    }
private:
    T* value;
};
```

```
int main() {  
    MyClass<int> obj(5);  
    cout << &obj << endl;  
    return 0;  
}
```

```
==59524==      by 0x100000C9C: main (SimpleClass.cpp:35)  
==59524==  
==59524== LEAK SUMMARY:  
==59524==      definitely lost: 4 bytes in 1 blocks  
==59524==      indirectly lost: 0 bytes in 0 blocks  
==59524==      possibly lost: 0 bytes in 0 blocks  
==59524==      still reachable: 4,096 bytes in 1 blocks  
==59524==                suppressed: 13,641 bytes in 156 blocks
```

The last line of the class definition highlighted as the problem.



## Example 2 - Simple class Add a destructor

```
virtual MyClass::~~MyClass() { // Not in original program
    delete value;
}
```

```
==59569== HEAP SUMMARY:
==59569==      in use at exit: 17,737 bytes in 157 blocks
==59569==    total heap usage: 172 allocs, 15 frees, 22,485 bytes allocated
==59569==
==59569== LEAK SUMMARY:
==59569==    definitely lost: 0 bytes in 0 blocks
==59569==    indirectly lost: 0 bytes in 0 blocks
==59569==    possibly lost: 0 bytes in 0 blocks
==59569==    still reachable: 4,096 bytes in 1 blocks
==59569==         suppressed: 13,641 bytes in 156 blocks
==59569== Reachable blocks (those to which a pointer was found) are not shown.
==59569== To see them, rerun with: --leak-check=full --show-leak-kinds=all
```