

# *CODE OPTIMIZATION*

# Introduction

- Code optimization improves program performance and reduces resource usage
- Techniques used in compilers transform source code into optimized object code
- This presentation explores code optimization techniques and their implementation in compilers
- Effectiveness of techniques evaluated in terms of performance, code size, and resource usage

```
31 def __init__(self, settings):
32     self.file = None
33     self.fingerprints = set()
34     self.logdups = True
35     self.debug = debug
36     self.logger = logging.getLogger(__name__)
37     if path:
38         self.file = open(os.path.join(path, 'requests.log'),
39                         'a')
40         self.file.seek(0)
41         self.fingerprints.update(self.request_fingerprint(request) for request in requests)
42
43 @classmethod
44 def from_settings(cls, settings):
45     debug = settings.getbool('debug', False)
46     return cls(job_dir(settings), debug)
47
48 def request_seen(self, request):
49     fp = self.request_fingerprint(request)
50     if fp in self.fingerprints:
51         return True
52     self.fingerprints.add(fp)
53     if self.file:
54         self.file.write(fp + os.linesep)
55
56 def request_fingerprint(self, request):
57     return request_fingerprint(request)
```

# Techniques for Code Optimization

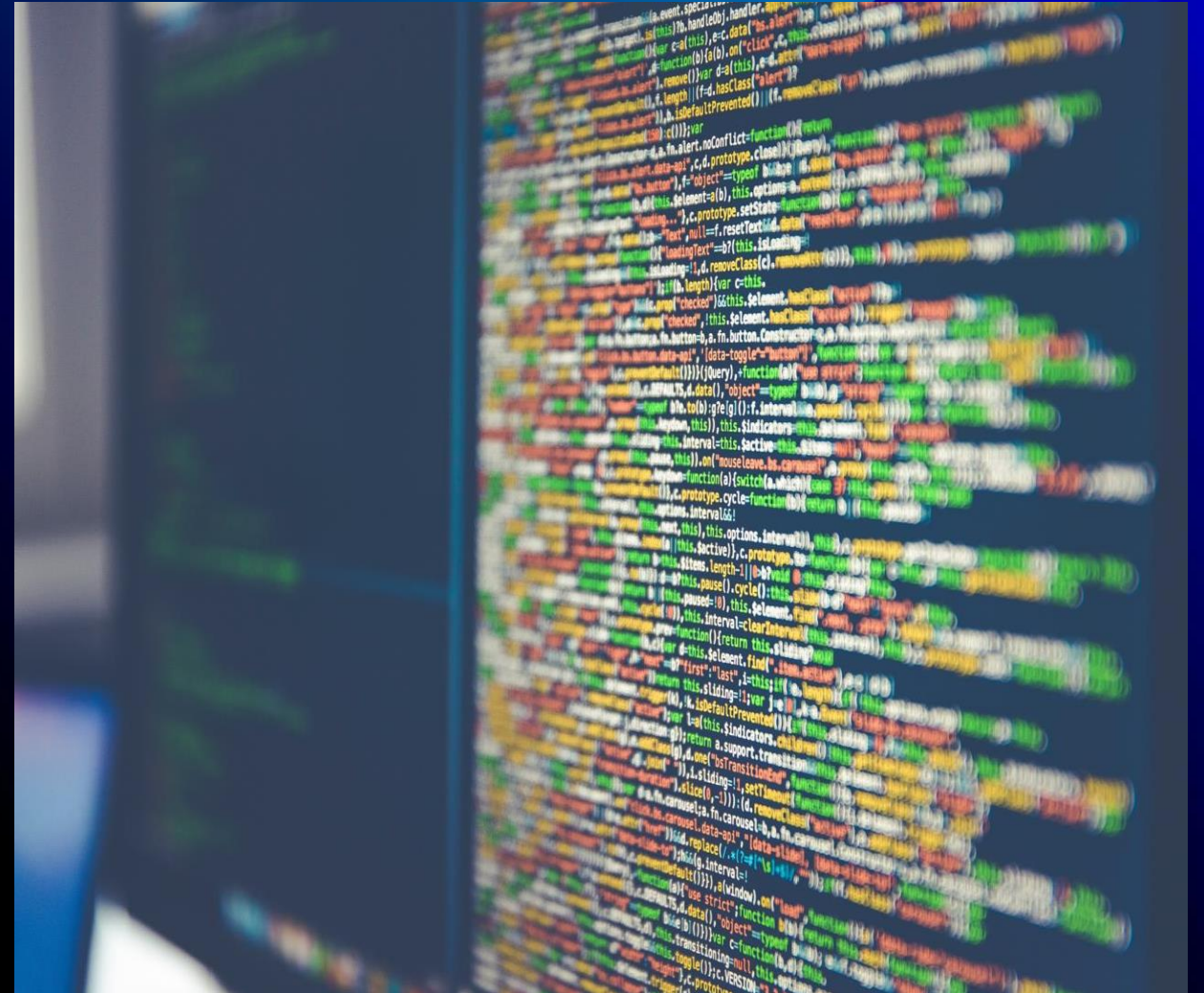
- *Data-flow analysis is a technique for identifying patterns in data values and how they are used in a program. This information is then used to optimize the code.*
- *Loop optimization is a technique for analyzing loops in a program to identify ways to improve their efficiency. This includes techniques like loop unrolling, loop fusion, and loop interchange.*
- *Instruction scheduling is a technique for rearranging instructions in a program to optimize their execution order. This includes techniques like instruction reordering, register allocation, and pipeline scheduling.*





# Evaluation of Optimization techniques

- The effectiveness of optimization techniques is evaluated in terms of performance improvements, code size reduction, and memory usage reduction.
- Performance improvements are typically measured in terms of execution time or throughput.
- Code size reduction is important for reducing memory usage and improving program load times.
- Memory usage reduction is important for improving program efficiency and reducing overall resource usage.



# Combined optimization

- Optimization techniques are often combined to produce more effective results.
- For example, loop optimization and instruction scheduling are often used together to improve program performance.
- Data-flow analysis can be used in combination with other optimization techniques to identify additional optimization opportunities.
- Combining optimization techniques can result in even greater performance improvements, code size reduction, and memory usage reduction than individual techniques alone.





# Conclusion

- Code optimization improves program performance, reduces code size, and minimizes resource usage.
- Optimization techniques can be used individually or in combination to produce optimized code.
- The effectiveness of optimization techniques is evaluated based on performance, code size, and memory usage.
- Careful consideration of trade-offs is necessary to balance optimization with other factors.
- Overall, code optimization is a vital tool for creating efficient and effective software.



*Thank you*

*Done by:*

*G Ashwin*

*V Allen Jerome*