CS4043 Advanced Programming

Omar Usman Khan
omar.khan@nu.edu.pk

Department of Computer Science
National University of Computer & Emerging Sciences, Peshawar

April 26, 2022

Syllabus

1. Course Overview
   - Premise
   - Note on Programming Languages
   - Job Market
2. Build and Analysis Toolchains
   - C/C++ Build Pipeline
   - Debugging and Profiling
3. Swift
   - Overview
   - Strings
   - Collections
   - Collections
   - Loops
   - Functions
   - Object Oriented Swift
   - Design Patterns

Course Overview

Marks Breakdown

- Mid-Term 1: 15%
- Mid-Term 2: 15%
- Final Examination: 50%
- Assignments: 20%

Assignment Notes

- Use Linux based tools
- For C/C++ (Use GCC)
- Marking Scheme (Typical):
  - 0/10: Not submitted by the time I mark them.
  - upto 5/10: submitted, but not working.
  - upto 8/10: submitted, partially working.
  - 10/10: submitted, fully working per requirements.
- No assignment submission through email
- May run through similarity index.

Course Overview (cont.)

Sample Topics

Programming for Performance, Productivity, Algorithms for Performance, Design Patterns, Event Driven Programming, Advanced GUI Development, Graphics 2D, Database/XML Integration, Distributed Computing, Integrated Coding/Testing/Debugging/SE Approaches, DevOp Toolchains: Version Control Systems, Docker, Kubernetes, Jenkins, ...

Complex Software

Production Process

1. Customers & Line of Business
2. Software Development Lifecycles (Design → Develop → Testing → Deploy)

Deployment Platforms

- Standalone Sytems
- Mobile Apps
- Distributed Software
- Big Data & Cloud
- Internet of Things
- ... Metaverse

- Software Complex than ever (Programming Languages, Data Sources, Integrated Data Types, Software Design, Algorithms, Performance, Group based Development, Security, ...)
- Solution: Embrace innovation in Programming Languages, Data Handling Methods, Undertake high level design choices, Software Development Frameworks, Development Methodologies, Execution Environments, ... All qualities of a **Software Architect** (job role appears with slight variations in names)

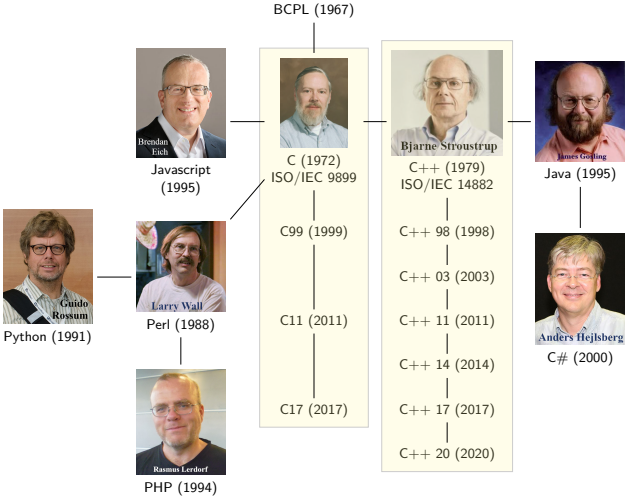
Complex Software (cont.)

Complex Software (cont.)

Course Learning Outcomes

- CLO1 To Develop an in-depth understanding of Programming Language Execution Models and their Run-time Systems
- CLO2 To Understand and Apply Various Design Patterns associated with Software Development Processes
- CLO3 To Understand and Use Toolchains Associated with Production and Deployment of Complex Software

Programming Languages



TIOBE Index (ISO 25010)

Feb 2022	Feb 2021	Change	Programming Language	Ratings	Change
1	3	↑	Python	15.33%	+4.47%
2	1	↓	C	14.08%	-2.26%
3	2	↓	Java	12.13%	+0.84%
4	4		C++	8.01%	+1.13%
5	5		C#	5.37%	+0.93%
6	6		Visual Basic	5.23%	+0.90%
7	7		JavaScript	1.83%	-0.45%
8	8		PHP	1.79%	+0.04%
9	10	↑	Assembly language	1.60%	-0.06%
10	9	↓	SQL	1.55%	-0.18%
11	13	↑	Go	1.23%	-0.05%
12	15	↑	Swift	1.18%	+0.04%
13	11	↓	R	1.11%	-0.45%
14	16	↑	MATLAB	1.03%	-0.03%
15	17	↑	Delphi/Object Pascal	0.90%	-0.12%
16	14	↓	Ruby	0.89%	-0.35%
17	18	↑	Classic Visual Basic	0.83%	-0.18%
18	20	↑	Objective-C	0.81%	-0.08%
19	19		Perl	0.79%	-0.13%
20	12	↓↓	Groovy	0.74%	-0.76%

TIOBE Quality Indicator

Measured by: TiCS

Higher Quality

A

B

C

D

E

F

Lower Quality

C

TQI Score	75.09%
Code Coverage	A B C D **E** F
Abstract Interpretation	**A** B C D E F
Cyclomatic Complexity	A **B** C D E F
Compiler Warnings	A **B** C D E F
Coding Standards	A B **C** D E F
Code Duplication	A B **C** D E F
Fan Out	A B **C** D E F
Security	A B C **D** E F

Measured on Apr 3, 2018

This product has been tested with the utmost care against the TIOBE Quality Indicator 4.0.

By the Way . . .


CODING SPEED
PYTHON
C++
JAVA
EXECUTION SPEED
C++
JAVA
PYTHON
PeopleOfCodes

Frameworks vs Libraries

- Both provide reusable abstractions of code, wrapped in well defined API
- Libraries: Flow of control dictated by caller (software calls library)
- Frameworks: Flow of control dictated by Framework (Framework drives software)

Framework Examples

- General Software: .NET, Android SDK, ...
- GUI Based: Gnome, QT, ...
- Web Based: ASP.Net, ...
- Concurrency: Hadoop Map/Reduce, ...
- ...

Skills in Demand Nationwide (Pakistan)

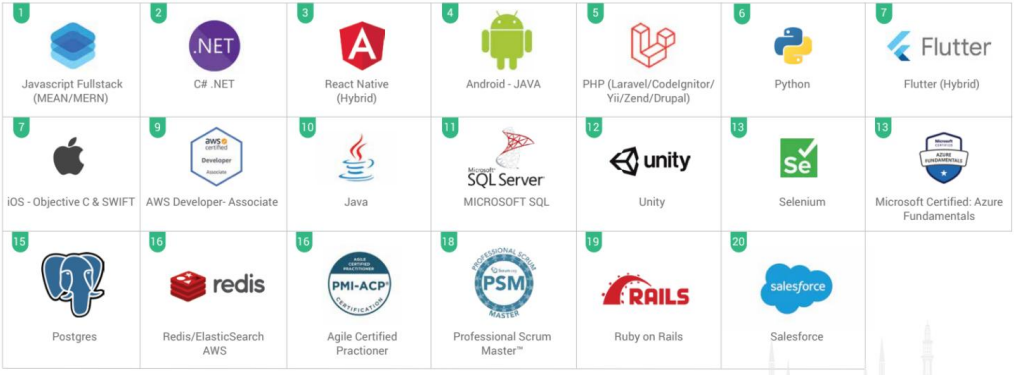


Figure 1: P@sha Top In-Demand Technologies & Tools

Skills in Demand Nationwide (Pakistan) (cont.)

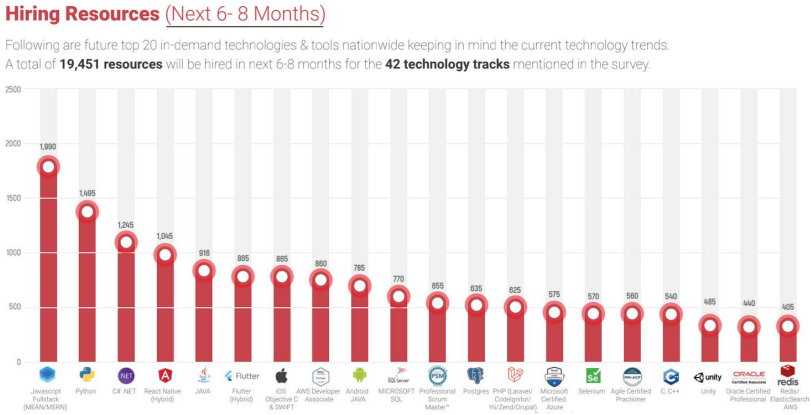


Figure 2: Next 6-8 Months Job Market

Skills in Demand Nationwide (Pakistan) (cont.)

Figure 3: Participating Companies List A

Skills in Demand Nationwide (Pakistan) (cont.)

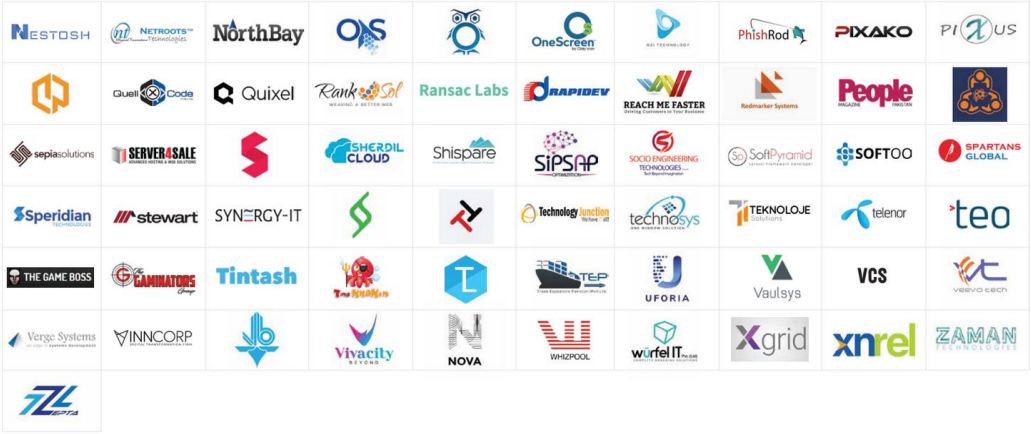
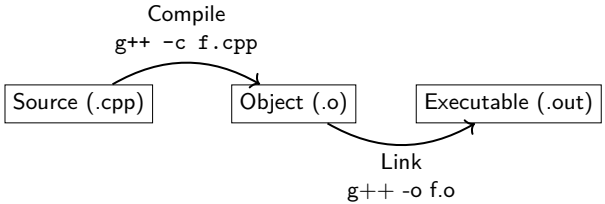


Figure 4: Participating Companies List B

Build Pipelines




```
g++ [options] source.cpp
```

```
gcc [options] source.c
```

```
clang++ [options] source.cpp
```

```
clang [options] source.c
```

```
icpc [options] source.cpp
```

```
icc [options] source.c
```

Sources

Integrated Development Environment

- CodeBlocks, Vim, Sublime Text, IntelliJ, Eclipse, Xcode, Android Studio, Geany, pyCharm, ... [a]
- Whichever you are comfortable with.
- Desirable Features:
  - Auto Complete features.
  - Auto Compilation features.
  - Debugger integration.
  - Code folding.
  - One IDE for all (development, office work)
  - Cross-Sectional Editing.
  - Remote development.
  - Integration with versioning systems
  - ...

[a] pypl.github.io for ranking

Sources (cont.)

Naming and Style Conventions

- cpp, hpp in C++, and c, h in C.
- followCamelCaseNamingConvention
- Folder conventions: *src* for source code, *build* or *bin* for binaries, *lib* for libraries, *doc* for documentation, *test* for unit testing.
- Suffixes: *g_* for global, *s_* for static, *c_* for constant variables.
- ... Much more on **google.github.io/styleguide** (Note: Company specific Guides)
- Can be enforced at commit time, or using static code checkers (e.g. **cpplint**)
- Indentation can be forced using **indent** with args gnu (for Stallman), kr (for Kernighan & Ritchie), linux (for Torvalds), orig (for Berkeley). Check man pages for details.

Sources (cont.)

Comments

- Governed by style guidelines.
- Documentation generators (Latex, HTML, XML, PDF, Man): Doxygen, Sphinx
- Doxygen Step 1: Fix settings
- Doxygen Step 2: Write comments (E.g. below)

```
/**
 * Search whether a given directed edge exists in a graph. Edges are specified as
 * head-tail pairs and must be specified in order.
 * @param G Graph object
 * @param head First incident vertex on edge
 * @param tail Second incident vertex on edge
 * @return Not Found = 0, Found = 1
 */
int directedEdgeExists(struct nGraph *G, int head, int tail) { }
```

- Doxygen Step 3: Generate documentations

Pre-Processing

- Source code transformation by Pre-Processor
- Pre-processor behavior controlled by directives
- File Inclusions:

```
#include <iostream>
#include <boost/tokenizer.hpp>
#include "my_header_file.hpp"
#include "some_directory/my_header_file.hpp"
```

- Macro definitions

```
#define DEBUG_LEVEL 10
#define myMacro(param1, param2) param1+param2
```

- Conditional Compilation Directives

```
#if, #ifdef, or #ifndef directive
#elif
#else
#endif
```

Compilations

Common Arguments

- -c Compile only
- -g Debugging symbols
- -p Performance symbols
- -O*n* Optimization level to *n* (0; none, 3: full)
- -stc Version (e.g. -std=c++20)
- -I Include directory
- -L Library path
- -l*txt* Linking with library (lib*txt*) file
- -Wall enable most warnings
- -Wextra enable extra warnings
- -Werror treat warnings as errors

Compilations (cont.)

ELF64

- Executable and Linkable Format (for all executables, object code, libraries)
- Described in elf.h of Linux kernel
- Learning Benefits: Digital forensics, OS internals, Malware research
- Viewable using `readelf`

ELF File Structure

- ELF Header (64 bytes, -h switch)
- Program Header (-l switch)
- Section Header (-S switch)

ELF File Types

- Relocatable
- Executable
- Shared Object Files

Compilations (cont.)

```
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2s complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              REL (Relocatable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          0 (bytes into file)
  Start of section headers:          384 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           0 (bytes)
  Number of program headers:         0
  Size of section headers:           64 (bytes)
  Number of section headers:         9
  Section header string table index: 1
```

Figure 5: ELF File Header `readelf -h obj.o`

Linking Process

Static Linking

- Code and variables resolved at compile time by interpreter ld, and copied into target application as a stand-alone executable
- .a filename convention
- Benefits: No dependency problems (single executable file)
- Cost: Large file size, Library code possibly loaded multiple times in memory

```
// file get15.c
// gcc -c get15.c


int get15() {
  return 15;
}

// ar -cvr libget15.a get15.o
```

```
// file main.c
// gcc main.c get15.o
// gcc main.c libget15.a
// gcc main.c -lget15

int main() {
  return get15();
}
```

Linking Process (cont.)

Dynamic Linking

- Code and variables resolved at load time by runtime interpreter ld-linux.so.2, and copied into target executable as symbols
- .so file convention
- Benefit: Small file size, library code loaded once in shared memory
- Cost: Dependency management

```
// file get15.c
// gcc -shared get15.c -o libget15.so
```

```
// file main.c
// gcc main.c -lget15
```

Linking Process (cont.)

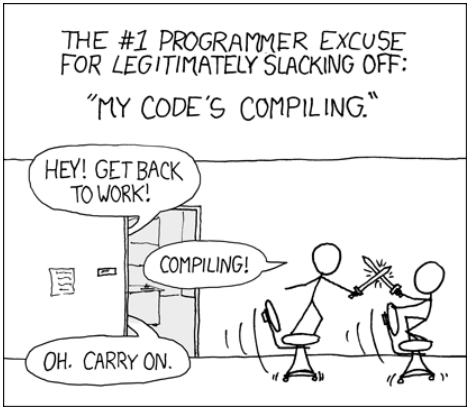
```
# Output: readelf -S a.out
[21] .dynamic          DYNAMIC          0000000000003de8  00002de8
     00000000000001f0  0000000000000010  WA       7     0     8

#Output: readelf -d a.out
Dynamic section at offset 0x2de8 contains 27 entries:
  Tag        Type                         Name/Value
 0x0000000000000001 (NEEDED)             Shared library: [libget15.so]
 0x0000000000000001 (NEEDED)             Shared library: [libc.so.6]
 0x000000000000000c (INIT)               0x1000
# .. . continues
```

- Use symbolic links when dealing with multiple versions

```
gcc -shared -Wl,-soname,libget15.so.1 -o libget15.so.1.0 get15.c
ln -sf libget15.so.1.0 libget15.so.1
ln -sf libget15.so.1.0 libget15.so
```

Automated Builds



- Single-file programs do not work well when code gets large
- Larger programs are split into multiple files

Automated Builds (cont.)

- Retyping commands is wasteful (Use ↑ or `CTRL+R` as shortcut)

GNU Make

- A utility for automatically compiling ("building") executables and libraries from source code.
- Often used for C programs, but not language-specific
- Follows Makefile format

```
myprogram : file1.c file2.c file3.c
	gcc -o myprogram file1.c file2.c file3.c
```

- Launch as `make` for first target, or `make myprogram` with direct target name
- Runs commands only if needed (based on timestamp)

Automated Builds (cont.)

```
all: aprogram

aprogram : foo.o bar.o
	gcc -o aprogram foo.o bar.o

foo.o: foo.c
	gcc -c foo.c

bar.o: bar.c
	gcc -c bar.c
```

- Standard Makefile targets: all, install, clean, distclean, ...
- Standard Variables: CC, CFLAGS, CXX, CXXFLAGS, LDFLAGS, ...

Debugging Using GDB / DDD

Code Debugging

- Step through a program line by line
- Inspect variables and objects as it steps through
- Inspect disassembled code as it steps through
- Inspect call stack as it steps through

Debugging Using GDB / DDD (cont.)

Code Debugging

GNU Debugger GDB

- Compile Time: `gcc -g myCode.c`
- Run Time: `gdb ./a.out`, followed by `run arg1 arg2`
- NCurses based frontend using `gdb ./a.out -tui`, or launching as normal, and issuing `layout src` after inserting any breakpoint.
- Commands:
  - Breakpoints `break file.c:10`, OR `break 10`, OR `break myFunc`
  - Delete breakpoints using `delete` or specifically by name
  - To view code: `list`
  - To view disassembled code: mintinlinebash|disassemble myFunc|
  - Iterate through code: `continue`, `step`, `next`
  - Inspect variables using: `print variableName`

Debugging Using GDB / DDD (cont.)

Code Debugging

GNU Project - Software

DDD v3.3 DataDisplayDebugger

- About DDD
- DDD News
- Getting DDD
- Building DDD
- Documentation
- Alpha Releases
- Reporting Bugs
- Where can I learn more about the debuggers DDD uses?
- Help and Assistance
- References

What is DDD?

GNU DDD is a graphical front-end for command-line debuggers such as GDB, DBX, WDB, Ladebug, JDB, XDB, the Perl debugger, the bash debugger bashdb, the GNU Make debugger remake, or the Python debugger pydb. Besides ``usual'' front-end features such as viewing source texts, DDD has become famous through its interactive graphical data display, where data structures are displayed as graphs.

Debugging for Memory Problems using Valgrind

Memory Profilers

Typical Memory Problems

- Uninitialized Variables
- Read/Write to un-allocated Memory (maybe segfaults generated)
- Deleting or Freeing dynamically created memory twice
- Memory Leaks

```
void f() {
  int *x = malloc(10 * sizeof(int));
  x[10] = 0;
}

int main(int argc, char *argv[]) {
  int n, i;
  f();
  for (i = 0; i < n; i++);
  return 0;
}
```

```
valgrind --leak-check=full ./a.out
```

Debugging for Memory Problems using Valgrind (cont.)

Memory Profilers

```
==23294== Invalid write of size 4
==23294==    at 0x108728: f (in /home/omar/work/codes/c/valgrind/a.out)
==23294==    by 0x108749: main (in /home/omar/work/codes/c/valgrind/a.out)
==23294==  Address 0x5204068 is 0 bytes after a block of size 40 allocated
==23294==    at 0x4C2EF1F: malloc (vg_replace_malloc.c:299)
==23294==    by 0x10871B: f (in /home/omar/work/codes/c/valgrind/a.out)
==23294==    by 0x108749: main (in /home/omar/work/codes/c/valgrind/a.out)
==23294==
==23294== Conditional jump or move depends on uninitialised value(s)
==23294==    at 0x10875D: main (in /home/omar/work/codes/c/valgrind/a.out)
==23294==
==23294== More than 10000000 total errors detected. I am not reporting any more.
==23294==
==23294== HEAP SUMMARY:
==23294==    in use at exit: 40 bytes in 1 blocks
==23294==    total heap usage: 1 allocs, 0 frees, 40 bytes allocated
==23294==
==23294== 40 bytes in 1 blocks are definitely lost in loss record 1 of 1
==23294==    at 0x4C2EF1F: malloc (vg_replace_malloc.c:299)
==23294==    by 0x10871B: f (in /home/omar/work/codes/c/valgrind/a.out)
==23294==    by 0x108749: main (in /home/omar/work/codes/c/valgrind/a.out)
==23294==
```

Debugging for Memory Problems using Valgrind (cont.)

Memory Profilers

```
==23294== LEAK SUMMARY:
==23294==    definitely lost: 40 bytes in 1 blocks
==23294==    indirectly lost: 0 bytes in 0 blocks
==23294==    possibly lost: 0 bytes in 0 blocks
==23294==    still reachable: 0 bytes in 0 blocks
==23294==    suppressed: 0 bytes in 0 blocks
```

Observing Memory Usage Behavior using Massif

Memory Profilers

- Memory usage as a function of allocation/deallocation event on heap (including stack)
- Usage: `valgrind --tool=massif --time-unit=B ./a.out`
- Visualization usage: `massif-visualizer massif.out.pid`

Observing Memory Usage Behavior using Massif (cont.)

Memory Profilers

Performance Measurement

Code Profilers

Profiler

An analysis that may measure usage of instructions in terms of their frequency or duration of execution. The goal of profiling is to aid program optimization.

- Common and Powerful Tools for Profiling Code: time, Fine-grained profiling, callgrind, gprof, oprofile, ...

`time ./a.out`

```
time ./a.out
# output of my program, if any.
real 0m0.003s # Wall clock time
user 0m0.001s # Userspace (accumulated CPU for both library + user space)
sys  0m0.001s # Kernelspace (I/O and/or syscalls)
```

- General Tip: Run for sufficiently long period + allow for relaxation period.

Performance Measurement (cont.)

Code Profilers

Fine Grained profiling

- Multiple Solutions !!!

```
struct timespec start, finish;                    // Defined in time.h
// struct timespec contains tv_sec, and tv_nsec
// struct timeval contains  tv_sec, and tv_usec

/* vs CLOCK_REALTIME (NTP dependent) */
clock_gettime(CLOCK_MONOTONIC, &start);
  /* Code Here */
clock_gettime(CLOCK_MONOTONIC, &finish);

double elapsed = (finish.tv_sec - start.tv_sec);
elapsed += (finish.tv_nsec - start.tv_nsec) / 1e9;
```

Performance Measurement (cont.)

Code Profilers

callgrind

- Part of valgrind,
- Compiler Flags: `gcc -g myCode.c`
- Call using valgrind: `valgrind --tool=callgrind ./a.out`
- View output on Terminal: `callgrind_annotate --auto=yes callgrind.out.NNNN`
- View output in GUI: `kcachegrind callgrind.out.NNNN`

Performance Measurement (cont.)

Code Profilers

gprof

- Compile as: `gcc myCode.c -pg`
- Run as: `./a.out` (will be slower than without profiling), output in `gmon.out`
- View gprof using: `gprof ./a.out`

% time	cumulative seconds	self seconds	calls	self s/call	total s/call	name
39.47	12.01	12.01	1	12.01	21.38	func1
30.79	21.38	9.37	1	9.37	9.37	func2
30.79	30.75	9.37	1	9.37	9.37	new_func1
0.13	30.79	0.04				main

Performance Measurement (cont.)

Code Profilers

```c
#include<stdio.h>

void func1(void)
{   printf("\n Inside func1 \n");
    for(int i = 0; i<0xffffffff; i++);
    new_func1();
    return;
}

void func2(void)
{   printf("\n Inside func2 \n");
    for(int i = 0; i<0xfffffffaa; i++);
    return;
}

void new_func1(void)
{   printf("\n Inside new_func1()\n");
    for(int i = 0;i <0xfffffffee; i++);
    return;
}

int main(void)
{
    printf("\n Inside main()\n");
    for(int i = 0; i<0xffffff; i++);
    func1();
    func2();

    return 0;
}
```

Performance Measurement (cont.)

Code Profilers

Call Graph Output

```
index % time    self  children    called     name
[1]    100.0    0.04   30.75                 main [1]
               12.01    9.37       1/1           func1 [2]
                9.37    0.00       1/1           func2 [3]
-----------------------------------------------
               12.01    9.37       1/1           main [1]
[2]     69.4   12.01    9.37       1         func1 [2]
                9.37    0.00       1/1           new_func1 [4]
-----------------------------------------------
                9.37    0.00       1/1           main [1]
[3]     30.4    9.37    0.00       1         func2 [3]
-----------------------------------------------
                9.37    0.00       1/1           func1 [2]
[4]     30.4    9.37    0.00       1         new_func1 [4]
-----------------------------------------------
```

Performance Measurement (cont.)

Code Profilers

OProfile

- Install it
- Run any code as: `sudo operf ls` (previously it was a kernel module as CONFIG_OPROFILE)
- output written to: oprofile_data in current directory
- View output as: `opreport`
- Cleanup by removing directory

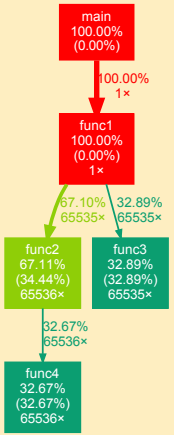
Performance Measurement (cont.)

Code Profilers

gprof2dot

- Can be called with any profiler tools
- Generates interesting graphics for documentation / publications
- Usage:

```
gprof ./a.out | gprof2dot | dot -Teps -o output.eps
```



Code Coverage using Gcov

- Find out different un-used portions of code during various tests.
- Compile using:

  `gcc -Wall -fprofile-arcs -ftest-coverage myCode.c`

  or

  `gcc myCode.c --coverage`
- Run your program
- Call `gcov myCode.c`

```
for (i = 1; i < 10; i++)
  {
    if (i % 3 == 0)
      printf ("%d is divisible by 3\n", i);
    if (i % 11 == 0)
      printf ("%d is divisible by 11\n", i);
  }
```

Code Coverage using Gcov (cont.)

Coverage output

```
      10:    8:  for (i = 1; i < 10; i++)
       -:    9:    {
       9:   10:      if (i % 3 == 0)
       3:   11:        printf ("%d is divisible by 3\n", i);
       9:   12:      if (i % 11 == 0)
   #####:   13:        printf ("%d is divisible by 11\n", i);
       -:   14:    }
```

Cross Compilation

- Compiler support required to merge object files of different languages
- Why? Performance and Native Calls

Wrapper Libraries

- C extension modules (for Python, Cython)
- Java extension modules (for Python, Jython)
- Mex files (for Matlab)
- Swift - Objective C extensions
- ...

Cross Compilation (cont.)

Example 1 (C Code)

```python
from ctypes import *

so_file="./fputs.so"
myCFunctions=CDLL(so_file)

myCFunctions.myfputs(b"Hello", b"write.txt")
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int myfputs(char *s, char *f) {
  FILE *fp = fopen(f, "w");
  int ret = fputs(s, fp);
  fclose(fp);
  return ret;
}
```

- Overview
- Strings
- Collections
- Collections
- Loops
- Functions
- Object Oriented Swift
- Design Patterns

Introduction

- Introduced by Apple in 2014 (Open Sourced in 2015)
- swift.org, github.com/apple
- Swift 1 (2014), Swift 2 (2015), Swift 3 (2016), Swift 4 (2017), Swift 5 (2019)
- Objective C ← Swift
- Primarily designed for Apple Ecosystem

```
/**
 * Filename: hello.swift
 * Run as: swift hello.swift
 * Or:     swiftc hello.swift && ./hello
 */

print("Hello, World!")
var str = "Hello, World!"

var x = 1                           // var definition
if (x == 1) {                       // brackets optional
    print("x == \(x)")              // \( ) for value of variable
}

let speedOfLight = 300000           // constant definition
```

Introduction (cont.)

```
var pi          = 3.14          // Inferenced as double
var z           = true          // Inferenced as bool

var redirect = ""
print(x, y, str, separator:", ", terminator:"", to: &redirect)      // Replaces defaults
```

Compiler Safety Features

- Curly Braces required for Loops and conditional statements
- Assignment operator = will not work inside conditional statements (if, while, etc.). E.g. `if (x = 1) {}`
- Data Type Safety with Type Inference. E.g. `str = "Hello, World!"; str = 1` not allowed.

Introduction (cont.)

Installation

- Dependencies: git cmake ninja-build clang python uuid-dev libicu-dev icu-devtools libedit-dev libxml2-dev libsqlite3-dev swig libpython-dev python-six libncurses5-dev pkg-config libcurl4-openssl- dev systemtap-sdt-dev tzdata rsync
- Source: github.com/apple/swift.git, Manjaro: swift-bin-development (or swift-bin), Gentoo: dev-lang/swift
- Windows: Use Windows Sub-sytem for Linux (powershell)

Introduction (cont.)

Creating Packages

```
swift package init --type=executable
#Creating executable package: packagetest
#Creating Package.swift
#Creating README.md
#Creating .gitignore
#Creating Sources/
#Creating Sources/packagetest/main.swift
#Creating Tests/
#Creating Tests/packagetestTests/
#Creating Tests/packagetestTests/packagetestTests.swift

swift build
#Building for debugging...
#[6/6] Linking packagetest
#Build complete! (2.71s)

swift run
#Building for debugging...
#Build complete! (0.29s)
#Hello, world!
```

Introduction (cont.)

Data Types

```
var str = "Hello"              // Inferenced as string
var pi  = 3.14                 // Inferenced as double
var x   = 1                    // Inferenced as int
var z   = true                 // Inferenced as bool
var sol = 300_000              // Inferenced as int with underscore ignored

var ppi = pi + x               // Not allowed as different types
var ppi = pi + Double(x)       // Allowed

var x                          // Throws Error: type annotation missing in pattern
var x: Int                     // Does not Throw Error

var x: Int8 = 95
x = 0b1011111                  // Specify as binary
x = 0o137                      // Specify as octal
x = 0x5f                       // Specify as hexadecimal
```

Introduction (cont.)

```
let π = 3.14159
let 你好 = "你好世界"
let 🐶🐮 = "dogcow"
```

Introduction (cont.)

Explicit Type Definitions

```
var x:Float = 3.14
var y:Float = "Hello"          // Will give error
var z:String = "Hello"         // Is fine
```

Type	Minimum	Maximum
Int8	-128	127
Int16	-32,768	32,767
Int32	-2,147,483,648	2,147,483,647
Int64	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
Int	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
UInt8	0	255
UInt16	0	65,535
UInt32	0	4,294,967,295
UInt64	0	18,446,744,073,709,551,615
UInt	0	18,446,744,073,709,551,615

Introduction (cont.)

Cocoa

- Provides an abstraction layer of the iOS operating system, and includes Cocoa API
- Written in Objective C

Optional Types

NULL

- Non-existent address for C pointers
- `void *`
- E.g. `int *x = NULL`, defined in `stddef.h`

nil

- Objective C: Non-existent objects (nil) and classes (Nil).
- Swift: An absent / non-existent value (anything can be nil)

```
var myVar1 = 3
var myVar2 = "a String"
var myVar3: String = "another string"
myVar1 = nil // error: error: 'nil' cannot be assigned to type 'Int'
myVar2 = nil // error: error: 'nil' cannot be assigned to type 'String'
myVar3 = nil // error: error: 'nil' cannot be assigned to type 'String'
```

- Variables by default in swift are *Non-Optional*, i.e. they must contain a valid, non-vil value.

Optional Types (cont.)

Optional Type

```
var myVar1: Int? = 3                        // var myVar1: Optional<Int>
var myVar2: String? = "a String"            // var myVar2: Optional<String>
var myVar3: String? = "another string"      // var myVar3: Optional<String>
myVar1 = nil // no error
myVar2 = nil // no error
myVar3 = nil // no error
```

Strings

- Usage of ", """, and #"

```
var astr = "conventional string"
var bstr = """
string across
multiple lines
"""
var cstr = #"Components of a "Raw String" with var \#(astr)."#
```

- Concatenations

```
var dstr = astr + bstr        // Concatenations
var estr = "\(astr) \(bstr)"
```

- Case conversions using `astr.lowercased()` and `astr.uppercased()` member methods.
- Comparisons using equality, prefix equality, suffix equality, and isEmpty methods.

```
var astr = "advanced programming"

print(astr == "advanced programming")
print(astr.hasPrefix("adv"))
print(astr.hasSuffix("ing"))
print(astr.isEmpty)
```

- String replacements (Apple Platforms Only)

Strings (cont.)

```
var astr = "one, to, three, four"
var bstr = astr.replacingOccurences(of:"to", with:"two")
```

Substring Type

- Not same as String type. Needs to be casted back.

```
var path = "/one/two/three/four"
let startIndex = path.index(path.startIndex, offsetBy: 4)   // Index(_rawBits: 262401)
let endIndex   = path.index(path.startIndex, offsetBy: 14)  // Index(_rawBits: 917761)
let apath      = path[startIndex ..< endIndex]              // /two/three
let bpath      = path[..<startIndex]                        // /one
let cpath      = path[endIndex...]                          // /four
let newStr     = String(apath)                              // /two/three
```

Strings (cont.)

Note on Operators

Type	C/C++	Swift
Assignment	=	=
Comparison	==, !=, >, <, <=, >=	==, !=, >, <, <=, >=
Arithmetic	+, -, *, /	+, -, *, /
Compound Arithmetic	+=, -=, *=, /=	+=, -=, *=, /=
Remainder	%	%
Closed range		(a...b)
Half range		(a..<b)
Ternary	?:	?:
Logical	!, &&, ||	!, &&, ||

Collections

Tuples

- Grouping multiple types into single compound type

```
var atuple               = ("Khyber Zalmy", 4, 0) // Unnamed Tuple
var (city, wins, losses) = atuple                 // members casted to vars
```

- Access members as `atuple.i`, where $i = 0, 1, \ldots$.

```
var atuple               = (city: "Khyber Zalmy", wins: 4, losses: 0) // Named Tuple
```

- Access members as `atuple.city`, ...

Collections

Arrays

Declarations

```
                var myArray1 = [3, 3, 3, 3, 3, 3]
                var myArray1 = [Int](repeating: 3, count: 6)
  Empty Array   var myArray2 = [Int](), or
                var myArray3: [Int] = []
Multiple Types  var myArray4: [Any] = [1, "two"]
    High Dims   var myArray4 = [[1, 2], [3,4]]
                var myArray5 = [[Int]]()
```

Collections (cont.)

Arrays

Accessing Members

```
var array1 = [1, 2, 3, 4, 5, 6]
var array2 = [[1, 2], [3, 4]]

print(array1[0])            // 1
print(array1.first)         // 1
print(array1.last)          // 6
print(array1[9])            // Index out of Bound Error
print(array1.isEmpty)       // False

print(array2[0])            // [1, 2]
print(array2[0][0])         // 1
print(array2[0].first)      // 1
```

Collections (cont.)

Arrays

Counting Members

```
var array1 = [1, 2, 3, 4, 5, 6]
var array2 = [[1, 2], [3, 4], [5,6]]

print(array1.count)        // 6
print(array2.count)        // 3
print(array2[0].count)     // 2
```

Shuffle and Shuffled

- Inplace shuffling: `array1.shuffle()`
- Out of Place shuffling: `var array3 = array1.shuffled()`

Collections (cont.)

Arrays

Inserting / Appending into Arrays

```
var array1 = [1, 2, 3, 4, 5, 6]

array1.append(7)          // 1, 2, 3, 4, 5, 6, 7
array1 += [8, 9]          // 1, 2, 3, 4, 5, 6, 7, 8, 9

array1.insert(10, at:3)   // 1, 2, 10, 3, 4, 5, 6, 7, 8, 9
```

Removing from Arrays

```
var array1 = [1, 2, 3, 4, 5, 6]

array1.removeLast()       // 1, 2, 3, 4, 5
array1.remove(at:2)       // 1, 3, 4, 5
array1.removeAll()
```

Collections (cont.)

Arrays

Functions on Arrays

```
var array1 = [3, 2, 1, 6, 5, 4]

/* In Place Variations */
array1.sort()                    // 1, 2, 3, 4, 5, 6
array1.sort() { $1 < $0 }        // 1, 2, 3, 4, 5, 6 through Closure
array1.sort(by: <)               // 1, 2, 3, 4, 5, 6 through by method

array1.sort() { $1 > $0 }        // 6, 5, 4, 3, 2, 1 through Closure
array1.sort(by: >)               // 6, 5, 4, 3, 2, 1 through by method

/* Out of Place Variations */
var array2 = array1.sorted()
```

Collections (cont.)

Arrays

Filtering on Arrays

```
var array1 = [3, 2, 1, 6, 5, 4]

var array2 = array1.filter{$0 > 2 && $0 < 5}  // 3, 4
```

Creating Maps for Arrays

```
var array1 = [3, 2, 1, 6, 5, 4]

var array2 = array1.map{$0 * 2}  // 6, 4, 2, 12, 10, 8
```

Collections (cont.)

Arrays

Foreach for Arrays

```
var array1 = [3, 2, 1, 6, 5, 4]
array1.forEach{ print($0) }

for item in array1 {
  print(item)
}
```

Collections (cont.)

Arrays

Difference between Arrays

```
var cities1 = ["London", "Paris", "Seattle", "Boston", "Moscow"]
var cities2 = ["London", "Paris", "Tulsa", "Boston", "Tokyo"]

let diff = cities2.difference(from: cities1)

for change in diff {
  print(change)
}
/*
remove(offset: 4, element: "Moscow", associatedWith: nil)
remove(offset: 2, element: "Seattle", associatedWith: nil)
insert(offset: 2, element: "Tulsa", associatedWith: nil)
insert(offset: 4, element: "Tokyo", associatedWith: nil)
*/
```

Collections

Dictionaries

Definition 1 (Dictionary)

- Container storing multiple key-value pairs.
- All keys and values are of same type
- Key for unique identification (Items looked up by key rather than index position)

```
let countries1 = ["US":"United States","UK":"United Kingdom"]
var countries2 = [1: "United States", 2: "United Kingdom"]

var countries3 = [String: String]()
var countries4 = [Int: String]()

countries3["PK"] = "Pakistan"
countries4[123]  = "Pakistan"

print(countries1["US"])              // United States
print(countries2[1])                 // United States

print(countries1.count)              // 2
print(countries3.isEmpty)            // True
```

Collections (cont.)

Dictionaries

```
countries1["UK"] = "Great Britain"
var original1 = countries1.updateValue("United Kingdom", forKey: "UK")
var original2 = countries1.updateValue("Germany", forKey: "DE")

print(countries1["UK"])                 // United Kingdom
print(original1)                        // Great Britain
print(countries1["DE"])                 // Germany
print(original2)                        // nil

countries1["DE"] = nil                                      // Removal method 1
var original3 = countries1.removeValue(forKey: "UK")        // Removal method 2

print(countries1["UK"])                 // nil
print(original3)                        // United Kingdom
```

Conditionals

```
int var1 = 7, var2 = 6;
if (var1 > var6) {
  // code
}
```

```
let var1 = 7, var2 = 6;
if var1 > var6 {
  // code
}
```

```
if (var1 > var6) {
  // code
} else {
  // code
}
```

```
if var1 > var6 {
  // code
} else {
  // code
}
```

Conditionals (cont.)

```
if (var1 > var2) {
  // code
} else if (var1 < var2) {
  // code
}
```

```
if var1 > var6 {
  // code
} else if var1 < var2 {
  // code
}
```

Guards

```
int x = 9
if (x > 10) {
  // Functional code
} else {
  // Error Condition
}
```

```
var x = 9
guard x > 10 else {
  // Error Condition
  return
}
// Functional code
```

Conditionals (cont.)

```
switch(x) {
  case 1:  // block code
           break;
  case 2:  // block code
           break;
  case 3:
  case 4:
  case 5:
  case 6:
  case 7:  if (y == 3) {
             // block code
           }
           break;
  default: // block code
           break;
}
```

```
switch x {
  case 1: // block code
  case 2: // block code
  case 3, 4, 5, 6, 7 where y == 3: // block code
  case 3...7 where y == 3: // block code
  default: // block code
}

let myDog = ("Alsation", 4)
switch myDog {
  case ("Terrier", let age):  // block code
  case ("Alsation", let age): // block code
}

switch myDog {
  case(_, 0...4): print("Your dog is young")
  case(_, 5...):  print("Your dog is old")
}
```

Loops

For-In

```
for index in 1...5 {
  print(index)
}

var countries = ["USA","UK", "PK"]
for item in countries {
  print(item)
}

let countries = ["US":"United States","UK":"United Kingdom", "PK":"Pakistan"]
for (abbreviation, name) in countries {
  print("\(abbreviation) and \(name)")
}
```

Loops (cont.)

Filter with For

```
let countries = ["US":"United States","UK":"United Kingdom", "PK":"Pakistan"]

for case let ("US", name) in countries {
  print(name)
}
```

While Variations

```
var ran = 0 while ran < 7 {
  ran = Int.random(in: 1..<20)
}
```

```
var ran: Int repeat {
  ran = Int.random(in: 1..<20)
} while ran < 7
```

Loops (cont.)

Filter with Where

```
for number in 1...30 {
  if number % 3 == 0 {
    print(number)
  }
}
```

```
for number in 1...30 where number % 3 == 0 {
  print(number)
}
```

Control Transfers

```
for i in 1...10 {
  if i % 2 == 0 {
    continue
  }
  print("\(i) is odd")
}
```

```
for i in 1...10 {
  if i % 2 == 0 {
    break
  }
  print("\(i) is odd")
}
```

```
switch x {
  case "baseball": fallthrough
  case "Baseball":
  case "basketball": fallthrough
  case "Basketball":
  default:
}
```

Functions

Basic Format

```
func sayHello1(name: String) -> Void {
  let retString = "Hello " + name
  print(retString)
}
```

- Called as:
  `sayHello1(name: "Omar")`

```
func sayHello2(name: String) -> String {
  let retString = "Hello " + name
  return retString
}
```

- Called as:

```
var msg = sayHello2(name: "Omar")
_       = sayHello2(name: "Omar")
```

Functions (cont.)

Ommitting Labels

```
func sayHello1(_ name: String) -> Void {
  let retString = "Hello " + name
  print(retString)
}
sayHello1("Omar")
```

Default Parameters

```
func sayHello2(name: String, greetings: String = "Bonjour") -> String {
  let retString = "\(greetings) \(name)"
  return retString
}
```

Functions (cont.)

Returning Multiple Values

```
func sayHello2() -> [String] {
  let retValues = ["FAST", "NUCES"]
  return retValues
}
func sayHello3() -> (name1: String, name2: String, aValue: Int) {
  let retValues = ("FAST", "NUCES", 1.0)
  return retValues
}
```

Functions (cont.)

Returning Optional Types

```
func sayHello4() -> String? {
  return nil
}
func sayHello5() -> (name1: String?, name2: String?, aValue: Int?) {
  let retValues = (nil, nil, nil)
  return retValues
}
func sayHello6() -> (name1: String, name2: String, aValue: Int)? {
  let retValues = nil
  return retValues
}
```

Functions (cont.)

Variadic Parameters

```
func sayHello7() -> (greetings: String = "Hello", names: String...) {
  for name in names {
    print("\(greetings) \(name)")
  }
}
sayHello7(names: "Omar", "Usman")
```

Functions (cont.)

Pass by Reference using inout

```
func sayHello8() -> (name1: inout String, name2: inout String) {
    let tmp = name1
    name1 = name2
    name2 = tmp
}
var one = "one"
var two = "two"
sayHello8(name1: &one, name2: &two)
```

Classes & Structures

```
class MyClass {
  var c      = 0
  var d: Int = 0
  var e      = ""
}
var myClass  = MyClass()
myClass.c    = 1
myClass.d    = 2
myClass.e    = "Hello"
```

```
struct MyStruct {
  var c: Int
  var d: Int
  var e: String
}
var myStruct = MyStruct(c: 1, d: 2, e: "Hello")
```

Classes & Structures (cont.)

```
class MyClass {
  var c       = 0
  var d: Int = 0
  var e       = ""

  init(c: Int, d: Int, e: String) {
    self.c = c
    self.d = d
    self.e = e
  }
}

var myClass  = MyClass(c: 1, d: 2, e: "Hello")
```

- Note: Failable Initializers also possible using init?

Classes & Structures (cont.)

Computed Types

```
struct EmployeeStruct {
  var empName = ""
  var salaryYear = 0.0

  var salaryWeek: Double {
    get {
      self.salaryYear/52
    }
    set(newValue) {
      self.salaryYear = newValue*52
    }
  }
}

var emp = EmployeeStruct(empName: "Omar", salaryYear: 100_000)

print(salaryWeek)              // 1923.07
salaryWeek = 500
print(salaryYear)              // 26000
```

Classes & Structures (cont.)

Property Observers (willSet and didSet)

```
var salaryYear: Double = 0.0 {
  willSet(newSalary) {
    print("About to set salaryYear \(self.salaryYear) to \(newSalary)")
  }
  didSet {
    print("Just changed the value")
  }
}
```

- Cannot be used with get and set observers
- Definition type is required

Classes & Structures (cont.)

Methods

```
struct EmployeeStruct {
  var empFirstName = ""
  var empLastName  = ""

  func fullName() -> String {
    self.empFirstName + self.empLastName
  }
}

var emp = EmployeeStruct(empFirstName: "Omar", empLastName: "Usman")
print(emp.fullName())
```

Classes & Structures (cont.)

Mutating Functions

```
mutating func giveRaise(amount: Double) {
  self.salaryYear += amount
}
```

Inheritance

- Basic OO development concept
- Allowing classes to derive another class with set of characteristics
- Separates classes from structures

```
class Plant {
  var height = 0.0
  var age = 0
}

class Tree: Plant {
  func growHeight(inches: Double) {
    height += inches
  }
}

var pine = Tree ()
pine.height = 4
pine.growHeight(inches: 4)
```

- **Multiple Inheritance not allowed**
  Classes can have Multiple Sub Classes
  Class can have only one Super Class

Inheritance (cont.)

Overriding

```
class Plant {
  var height = 0.0
  var age = 0
  func getDetails() -> String {
    return "Plant Details"
  }
}

class Tree: Plant {
  func growHeight(inches: Double) {
    height += inches
  }
  func getDetails() -> String {
    return "Tree Details"
  }
}

var plant = Plant ()
var tree  = Tree()
print("Plant: \(plant.getDetails())")
print("Tree:   \(tree.getDetails())")
```

Inheritance (cont.)

Overriding: Calling Super Class

```
class Plant {
  var height = 0.0
  var age = 0
  func getDetails() -> String {
    return "Height: \(height), Age: \(age)"
  }
}
```

```
class Tree: Plant {
  func growHeight(inches: Double) {
    height += inches
  }
  override func getDetails() -> String {
    let details = super.getDetails()
    return "Tree: \(details)"
  }
}
```

- `override` applies to stored and computed properties also.
- **Override Prevention** using `final` keyword before items definition.

Protocol

Protocol

Used to describe Implementations (methods and properties) of a type without actually providing any implementation.

```
protocol MyProtocol1 {}
protocol MyProtocol2 {}

struct MyStruct: MyProtocol1, MyProtocol2 {}
class  MyClass:  MyProtocol1, MyProtocol2 {}
```

- Mention Super Classes before Protocols when both are used.

Protocol (cont.)

```
protocol NamingConvention {
  var firstName: String { get set }
  var lastName:  String { get set }

  func fullName() -> String
}

class  scientist:  NamingConvention {
  var firstName = "Albert"
  var lastName  = "Einstein"

  func fullName() -> String {
    return "\(firstName) \(lastName)"
  }
}
```

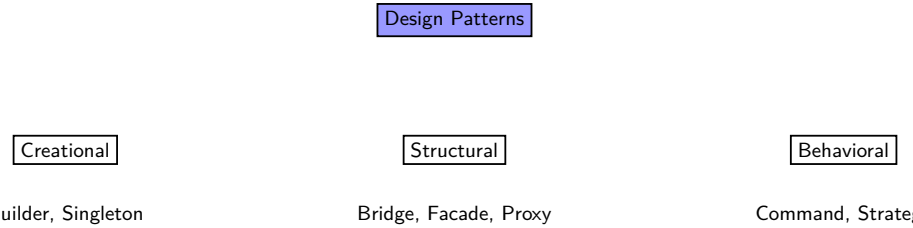
Design Patterns

- Software Engineering (Analysis and Design); Usually plenty of Repetitions in Design depending on type of analysis
- **Design Patterns**: Evolution of best practices for common problems.
- Role of Software Architect
- Not the same as algorithms

Benefits

- All the good practices for common problems (evolutionary process)
- Consistent code writing (generic / organizational)
- Code Reuse

Design Patterns (cont.)



Creational Design Patterns

Singleton

- Only one instance of a class active; i.e. maintains a single object for entire lifetime of the object.
- Does not work with structs (value type vs reference type)
- Drawbacks: Race conditions in parallel setups

```
class MySingleton {
  static let sharedInstance = MySingleton()
  var number                = 0
  private init() { }
}

var A = MySingleton.sharedInstance
var B = MySingleton.sharedInstance

A.number = 111
print(B.number)
```

Creational Design Patterns

Builder

```
struct BurgerOld {
  var name: String,  patties: Int,  bacon: Bool,   cheese: Bool, pickles: Bool
  var ketchup: Bool, mustard: Bool, lettuce: Bool, tomato: Bool
  init(name: String,  patties: Int,  bacon: Bool,   cheese: Bool, pickles:Bool,
       ketchup: Bool, mustard: Bool, lettuce: Bool, tomato: Bool) {
    self.name = name
    self.patties = patties
    self.bacon = bacon
    self.cheese = cheese
    self.pickles = pickles
    self.ketchup = ketchup
    self.mustard = mustard
    self.lettuce = lettuce
    self.tomato = tomato
  }
}
```

Creational Design Patterns (cont.)

Builder

- **Issues**: Items can be forgotten, Too much user side code, How to make different variations of a complex object
- **Solution** Assemble parts of the object through a builder object.

```
protocol BurgerBuilder {
  var ketchup: Bool { get }
  var tomato: Bool { get }
  var mustard: Bool { get }
  var lettuce: Bool { get }
  var pickles: Bool { get }
  var cheese: Bool { get }
  var patties: Int { get }
  var bacon: Bool { get }
}
```

- Create Burger Builders

```
class BurgerStandard   : BurgerBuilder { }
class BurgerVegetarian : BurgerBuilder { }
```

Creational Design Patterns (cont.)

Builder

- Create simplified new Class

```
struct BurgerNew {
  var name: String, patties: Int, bacon: Bool, cheese: Bool, pickles: Bool
  var ketchup: Bool, mustard: Bool, lettuce: Bool, tomato: Bool
  init(builder: BurgerBuilder) {
    self.name = name
    self.patties = patties
    self.bacon = bacon
    self.cheese = cheese
    self.pickles = pickles
    self.ketchup = ketchup
    self.mustard = mustard
    self.lettuce = lettuce
    self.tomato = tomato
  }
}
```

Creational Design Patterns (cont.)

Builder

- Create objects

```
var myBurger       = BurgerNew(builder: BurgerStandard())
var myCheeseBurger = BurgerNew(builder: BurgerVegetarian())
```

Structural Design Patterns

Bridge Pattern

- Performs abstraction across two layers (so that implementaitons may vary slightly)
- Control Refactoring
- Controls Class/Combinatorial Explosion

```
class sendMessage {
  var messageString: String

  init(messageString: String) {
    self.messageString = messageString
  }

  func sendmessage() {
    // send message functionality
  }
}
```

New Requirements

- Prepare Connection
- Encrypt Message
- Send as Email
- Send as SMS
- Pre-Process / Verify message
- ...

Structural Design Patterns (cont.)

Bridge Pattern

```
protocol Message {
  var messageString: String { get set }
  init(messageString: String)
  func prepareMessage()
}
```

```
class PlainTextMessage: Message {
  var messageString: String
  init(messageString: String) {
    self.messageString = messageString
  }
  func prepareMessage() {
    //Nothing to do
  }
}

class DESEncryptedMessage: Message {
  var messageString: String
  init(messageString: String) {
    self.messageString = messageString
  }
  func prepareMessage() {
    // Encrypt message here
    self.messageString = "DES: " + self.messageString
  }
}
```

Structural Design Patterns (cont.)

Bridge Pattern

```
protocol Sender {
  func sendMessage(message: Message)
}
```

```
class EmailSender: Sender {
  func sendMessage(message: Message) {
    print("Sending through E-Mail:")
    print("\(message.messageString)")
  }
}

class SMSSender: Sender {
  func sendMessage(message: Message) {
    print("Sending through SMS:")
    print("\(message.messageString)")
  }
}
```

Usage

```
var myMessage = PlainTextMessage(messageString: "Plain Text Message")
myMessage.prepareMessage()

var sender    = SMSSender()
sender.sendMessage(message: myMessage)
```

Structural Design Patterns (cont.)

Bridge Pattern

Message Bridge

```
struct MessagingBridge {
  static func sendMessage(message: Message, sender: Sender) {
    var sender = sender
    message.prepareMessage()
    sender.message = message
    sender.sendMessage()
  }
}
```