Part A.)

A1.)

1. Unix
2. Android
3. Palm Piolet
4. Queueing Theory
5. NeXT
6. LINUX
7. stdio.h
8. First come first serve
9. Heap
10. Ready & Running
11. preemptive
12. context switching
13. an integer
14. uniform memory access
15. quantum
16. the right of
17. 0
18. optional
19. RAM
20. malloc
21. Hugepages
22. kernel, daemon
23. 4kB
24. The buddy blocks combine into a larger free block.
25. 2
26. Clean
27. unit
28. speed for less memory
29. ‘world\0’
30. used

A2.)

1. DOS/360
2. Windows
3. Linux
4. Quality Operating Systems
5. 5
6. Multi-User
7. esc
8. The low priority processes will be held and will then resume after the higher priority process finishes execution.

9. Heap
10. Program Counter
11. 32
12. Smaller
13. Print character
14. Processors
15. Kernel
16. I
17. 0
18. Hit-or-miss
19. Dirty
20. Old pages
21. 000
22. 2
23. First-fit
24. Two 2^0 size blocks
25. Double
26. Lookaside
27. Page in Page out
28. Grow up
29. Time
30. Least recently used

A3.)

1. Monitor
2. Linux
3. Linux
4. Boot
5. Motorola 68000
6. Preemptive Multitasking
7. Time-zone
8. Burst Rate
9. Security, memory allocation is not predictable
10. Doubly linked list
11. Any number that is large
12. Waiting queue
13. %d\t%d
14. migration
15. Priority
16. q
17. The child runs first after fork
18. It is worth zero
19. Thrashing
20. RES
21. Stack

22. A process is killed
23. Internal Fragmentation
24. 2MiB (4KiB * 512 = 2048KiB)
25. Stays the same (size of IPT is proportional to size of physical memory)
26. L2
27. Protection
28. On disk
29. Length
30. Page fault rates

A4.)

1. GM-NAA I/O
2. Kolibri Operating System
3. Engine Control Unit
4. Clothes washing line
5. Ctrl-Z
6. Multithreading
7. Rate Monotonic Scheduling
8. Overflow the stack
9. SCHED_FIFO
10. Aging
11. Pointer to current register set
12. A string, with spaces appended to make it of length 20
13. Memory Stall
14. When the amount of threads exceeds the amount of registers on a barrel processor.
15. ls -a
16. 21301,0
17. Central Processor
18. Prepaging
19. VIRT
20. Decrease
21. Encoding
22. Free Frame
23. 2^0, 2^1, 2^2
24. Sharing
25. It has the page
26. Page lookup to find a page.
27. 11
28. Buffer overflow attack
29. Interrupted Programming

Part B.)

B1.) My phone uses IOS. IOS does use virtual memory but does not include swap memory. Apps are restricted to the available RAM.
B2.) The first OS that I ever used was Windows XP. This windows desktop had 512MB of RAM. I do not know how much swap space that it had.

B3.) The RAM from the physical hardware is shared between operating systems on a hypervisor. A type 1 hypervisor is safer because it does not depend upon an underlying operating system.

B4.)

- Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') - *(79)*
- Out-of-bounds Write - *(787)*
- Improper Input Validation - *(20)*
- Out-of-bounds Read - *(125)*
- Improper Restriction of Operations within the Bounds of a Memory Buffer - *(119)*
- Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection') - *(89)*
- Exposure of Sensitive Information to an Unauthorized Actor - *(200)*
- Use After Free - *(416)*
- Cross-Site Request Forgery (CSRF) - *(352)*
- Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection') - *(78)*
- Integer Overflow or Wraparound - *(190)*
- Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') - *(22)*
- NULL Pointer Dereference - *(476)*
- Improper Authentication - *(287)*
- Unrestricted Upload of File with Dangerous Type - *(434)*
- Incorrect Permission Assignment for Critical Resource - *(732)*
- Improper Control of Generation of Code ('Code Injection') - *(94)*
- Insufficiently Protected Credentials - *(522)*
- Improper Restriction of XML External Entity Reference - *(611)*
- Use of Hard-coded Credentials - *(798)*
- Deserialization of Untrusted Data - *(502)*
- Improper Privilege Management - *(269)*
- Uncontrolled Resource Consumption - *(400)*

- Missing Authentication for Critical Function - *(306)*
- Missing Authorization - *(862)*

Out-of-bounds Read: As a programmer, I would assume all input as malicious. As a product manager, I would want to consider all potentially relevant properties. As a CTO, I would recommend validating correct calculations for any argument. As an NST policymaker, I would recommend not exclusively looking at malicious or malformed inputs.

NULL Pointer Dereference: As a programmer, I would check that the results of all functions return a value and verify it is non-null before acting. As a product manager, I would ensure that all pointers modified are sanity checked. If I were a CTO or NST policymaker, I would choose a language that is not susceptible to these issues.

B5.)

1) 5 ENTER 2 + ENTER 7 *

2) 38899.00

3.) 31, 02, 51, 07, 61, 74, 13 00, 13 00

4.)

MIPS Assembly Branch Example. Branches if $t0 == $t1

beq $t0, $t1, Target
goto Loop

5.) These commands imply that the calculator can have functions that return a value, unlike the HP25.

Part C.)

C1., C2., C3.)

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

int main(void)
{
    // A string to hold my name
    const char name[]  = "Nathan Mack";

    // Allocates memory for my name string
    int nameLength = strlen(name) + 1;
    char* nameMem = malloc(nameLength);

    // Control flow to ensure allocation of memory
    if (nameMem == NULL)
    {
        printf("Couldn't allocate memory!");
    }

    else
    {
        strcpy(nameMem, name);
        printf("Name = %s\n", nameMem);
    }

    // Prints every char in my name with index
    for(int i=0; i<strlen(name); i++)
    {
        printf("[%d] %c\n", i, nameMem[i]);
    }

    free(nameMem);
}
```

```
nam116@eecslab-1:~/csds338/hw2$ ./a.out
Name = Nathan Mack
[0] N
[1] a
[2] t
[3] h
[4] a
[5] n
[6]
[7] M
[8] a
[9] c
[10] k
```

C4.)

Copying a pointer to a string

```c
#include <stdio.h>

int main(void)
{
    char emptyString1[] = "";
    char testString [] = "testString";

    *emptyString1 = testString;
    printf("%s", emptyString1);
}
```

Strcopy copies a string to a location in memory without size specification. Similar to the pointer example above. Strncopy copies to a location in memory with a specific size parammeter. The danger here is that a termination character does not need to be specified.

C5.)

Code demonstrating fork, exec, wait

```c
#include <stdio.h>

/* This program forks and and the prints whether the process is
 *   - the child (the return value of fork() is 0), or
 *   - the parent (the return value of fork() is not zero)
 *
 * When this was run 100 times on the computer the author is
 * on, only twice did the parent process execute before the
 * child process executed.
 *
 * Note, if you juxtapose two strings, the compiler automatically
 * concatenates the two, e.g., "Hello " "world!"
 */

int main( void ) {
        char *argv[3] = {"Command-line", ".", NULL};

        int pid = fork();

        if ( pid == 0 ) {
                printf("%d\n", (int)getpid());
                execvp( "find", argv );
        }

        /* Put the parent to sleep for 2 seconds--let the child finished executing */
        wait( 2 );

        printf("%d\n", (int)getpid());
        return 0;
}
```

Code Output, 8666 is the child process, 8665 is the parent

```
nam116@eecslab-1:~/csds338/hw2$ ./a.out
8666
.
./c5.c
./a.out
./c4.c
./c1.c
8665
```

Part D.)

(A) = Equiprobable
(B) = Strong low numbered page bias
(C) = Bias for 3 < k < 10 page numbers

NOTE: The page numbers for my code range for 1 to 10

FIFO Page Replacement Algorithm for 1000 Pages w/ 5 page capacity

Page Faults (A): 504
Page Faults (B): 75
Page Faults (C): 294

Clock Page Replacement Algorithm for 1000 Pages w/ 5 page capacity

Page Faults (A): 479
Page Faults (B): 97
Page Faults (C): 287

Refer to:
page_replacement_fifo.py
page_replacement_clock.py