ECE220 Lab5

```
int count_steps_recursive(int n)
{
    if (n < 0)
        return 0;
    else if (n == 0)
        return 1;

    return count_steps_recursive(n - 1) +
           count_steps_recursive(n - 2) +
           count_steps_recursive(n - 3);
}
```

Brain Teaser - Recursion

Design a recursive algorithm

- A child going up a staircase with n steps, can hop up 1, 2, or 3 steps at a time. How many ways can the child reach the top?

Solution

- Recursively call *count_steps_recursive* with decremented count.

```c
int count_steps_dp(int n)
{
    if (n < 0)
        return 0;
    else if (n < 2)
        return 1;
    else if (n == 2)
        return 2;

    int *arr = (int*)malloc((n + 1) * sizeof(int));
    arr[0] = 1, arr[1] = 1, arr[2] = 2;

    for (int i = 3; i <= n; i++)
        arr[i] = arr[i-1] + arr[i-2] + arr[i-3];

    int steps = arr[n];
    free(arr);
    return steps;
}
```

Brain Teaser – Dynamic Programming

Can we improve algorithm using concepts learned two weeks ago?

- Hint: pointers, arrays

Solution

- Create an array to store counts for each step.

Dynamic vs static memory allocation

- Dynamic
  - malloc: int *arr = (*int)malloc(size_t size);
- Static
  - VLAs: int arr[size];
  - alloca: int *arr = (*int)malloc(size_t size);

```
int count_steps_vars(int n)
{
    if (n < 0)
        return 0;
    else if (n < 2)
        return 1;
    else if (n == 2)
        return 2;

    int a = 1, b = 1, c = 2;
    for (int i = 3; i <= n; i++)
    {
        int sum = a + b + c;
        a = b;
        b = c;
        c = sum;
    }

    return c;
}
```

Brain Teaser – Further improvements

Can we further improve algorithm?

Solution:

- Replace array with variables.
- Array will be stored in cache/RAM
- Variables will be stored in registers

Computer Memory Hierarchy (2012)

1. Registers: 1 clock cycle
2. L1 cache: 0.5 ns
3. L2 cache: 7 ns
4. RAM: 100 ns
5. SSD: 150 μs

option	optimization level	execution time	code size	memory usage	compile time
-O0	optimization for compilation time (default)	+	+	-	-
-O1 or -O	optimization for code size and execution time	-	-	+	+
-O2	optimization more for code size and execution time	--		+	++
-O3	optimization more for code size and execution time	---		+	+++
-Os	optimization for code size		--		++
-Ofast	O3 with fast none accurate math calculations	---		+	+++

Brain Teaser – Measuring Performance

Compilation:

- gcc –Ox main.c

Elapsed times for –O0 and $n = 35$:

- count_steps_recursive: 10.23 s
- count_steps_dynamic: 2e-6 s
- count_steps_vars: 1e-6 s

Elapsed times for -0fast and $n = 35$:

- count_steps_recursive: 8.02 s
- count_steps_dynamic: 3e-6 s
- count_steps_vars: 1e-6 s

```c
int main()
{
    // Function table
    int (*count_steps[3])(int) = { count_steps_recursive, count_steps_dp,
                                   count_steps_vars };

    // Time each function
    for (int i = 0; i < 3; i++)
    {
        clock_t begin = clock();

        count_steps[i](35);

        double func_time = (double)(clock() - begin) / CLOCKS_PER_SEC;
        printf("Elapsed time for %d - %f\n", i, func_time);
    }

    return 0;
}
```

Measuring Execution Time

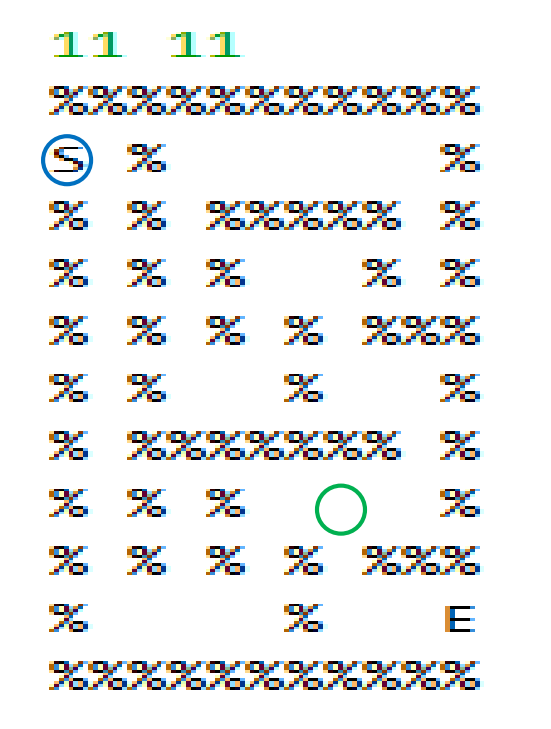
Solving Sudoku

	4	9			3		7	
	2		5	4		3		
			8	9			4	
9		5		2			8	1
4	7			8		9		6
	8			7	2			
		6		5	8		1	
	5		1			8	6	

1	4	9	2	3	3		7	
	2		5	4		3		
			8	9			4	
9		5		2			8	1
4	7			8		9		6
	8			7	2			
		6		5	8		1	
	5		1			8	6	

1	4	9	2	6	3	5	7	_
	2		5	4		3		
			8	9			4	
9		5		2			8	1
4	7			8		9		6
	8			7	2			
		6		5	8		1	
	5		1			8	6	

MP6 – Mazes



Solving mazes

- Start at **S** and end at **E**

Ideas on how to solve?

Recursion

- Depth first search algorithm
  - At each cell, check which surrounding four cells are available
  - Blue circle has one path
  - Green circle has three paths
  - Mark cells to prevent infinite recursion
- Basic algorithm on the wiki

Lab6 – Vector Data Structure

Arrays
- Statically allocated in sense that size is fixed

Vectors
- Dynamically allocated data structure (at least in higher-level languages)

Methods:
- vector_t *createVector(int initialSize);
- void destroyVector(vector_t *vector);
- void resize(vector_t *vector);
- void push_back(vector_t *vector, int element);
- int pop_back(vector_t *vector);
- int access(vector_t *vector, int index);

Valgrind
- Memory debugging, memory leak, detection, and profiling tool.
- Check if you have any memory leaks.