APS 105: Computer Fundamentals

Tutorial #4
Summer 2025

Problem 1: Simple Array (Winter 2024 Final Exam, Q1)

Write a single C statement that declares a 1D array of 100 integers named *arr*, where the first three elements are set to 1, 3 and 10 and the remaining 97 elements are set to 0.

Example solution:

```c
int arr[100] = {1, 3, 10};
```

Problem 2: Dynamic Allocation of Arrays (Fall 2015 Final Exam, Q2)

Write one C statement that use *malloc* to dynamically allocate an array of elements of type *double*. The allocated array should be called *list*.

Example solution:

```c
double *list = (double *)malloc(1000 * sizeof(double));
```

Problem 3: 2D Arrays and Functions

Write a C function called *borderSum* that adds all the border elements of the top-left 2D square matrix inside a 2D square array. For example, for the array below, if n is 3, we should add $1, 2, 3, 7, 11, 10, 9, 5$ and return 48. If n is 0, the function should return 0. If n is 1, the function should return the top-left element, which is 1 for the array below.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

The header of the *borderSum* function is provided below, where 26 is the number of rows and columns in the 2D array mat, and n is the size of the square matrix to which we need to get the sum of its border. You can safely assume $n \leq 26$.

```
int borderSum(int mat[][26], int n){
```

Example solution:

```
int borderSum(int mat[][26], int n) {
  int sum = 0, row, col;

  if (n == 0) {
    return 0;
  }   else if (n == 1) {
      sum = mat [0][0];
  }   else if (n == 2) { //unnecessary else-if, but fine if it is there
      sum = mat [0][0] + mat [0][1] + mat [1][0] + mat [1][1];
  }   else {
      for (col = 0; col < n; col++) {
      sum += mat[0][col] + mat[n - 1][col];
      }

      for (row = 1; row < n - 1; row++) {
      sum += mat[row][0] + mat[row][n - 1];
    }
  }
  return sum;
}
```

Problem 4: Working with Arrays and Pointers (Winter 2024 Final Exam, Q3)

In the box provided below, write the output generated after the following program is completely executed.

```
#include <stdio.h>

int main() {
  int i = 1, *first;
  int myArray[] = {0, 1, 2, 3, 7, 11, 15};

  while (myArray[i] < 3) {
    *( myArray + i) = 3;
    i++;
  }

  *( myArray + i) = 0;
  first = myArray + i + 1;
  myArray[0] = *first;
  *first = myArray[i];

  printf("i = %d, *first = %d\n", i, *first);
```

```
  printf("myArray has ");
  for (int i = 0; i < 7; i++) {
  printf("%d, ", myArray[i]);
  }

  return 0;
}
```

Example solution:

```
i = 3, *first = 0
myArray has 7, 3, 3, 0, 0, 11, 15,
```

Problem 5: Dynamic Memory Allocation (Winter 2023 Midterm, Q9)

Create a code segment in C that takes a dynamically allocated array and adds another element to it. Use the following variables (assume they are already defined).

1. *int newNumber*, an integer value to be added into the array
2. *int numbersInArray*, a non-negative value of how many elements are presently in the array.
3. *int *pArray*, a pointer to the array which is in dynamically allocated memory. The array already exists and has one or more elements. The code segment must add *newNumber* to the present array, leaving *pArray* pointing to the new array with *newNumber* in it. Since the present array size is not large enough, you will have to allocate a new space and move the present values plus *newNumber* to this space. Avoid memory leaks.

Example solution:

```
int *extArray = (int *)malloc(sizeof(int) * (numInArray + 1));
extArray[0] = newNum; // add at first since easier

for (int i = 1; i <= numInArray; i++){
  extArray[i] = pArray[i - 1]; // transfer old stuff
}

free(pArray);
pArray = extArray; // reassign pointer to new array space
```

Problem 6: Count Letters (Winter 2019 Final Exam, Q9)

Write a function, countLetters, that counts the number occurrences of each alphabetical letter found in a string. The function has two parameters: a string (*i.e.* char *) and an array of integers. The string should not be modified and can be any length. You may

assume that the string is null-terminated and all letters are lower case. The string may contain characters that are not part of the alphabet (*e.g.*, '0', '1', '!', '&', *etc*). The integer array count has a size of 26, one for each letter in the alphabet.

The first index corresponds to the letter 'a', the second index to the letter 'b', and so on. You may assume that, initially, all 26 elements in the array have a value of zero. You must abide by the following constraints. Failure to meet a constraint will result in a grade of zero for this question.

1. You cannot modify the characters inside the string.
2. You cannot create any other data structures (e.g., array, linked list, etc),
3. Your function must only access valid indices of the array.
4. You cannot call any functions in your implementation.

An example of one run of the program is below. You only need to implement the countLetters function. Assume a main function (that reads in a string, calls your function, and outputs the integer array) already exists.

```
Enter a sentence: hello, world
The frequency that each lower case letter appears is:
d: 1
e: 1
h: 1
l: 3
o: 2
r: 1
w: 1
```

Function Prototype:

```
void countLetters(char *s, int count[])
```

Example solution:

```
void countLetters(char *s, int count[]) {
  int i = 0;
  int whichLetter;

  while (s[i] != '\0') {
    if (s[i] >= 'a' && s[i] <= 'z') {
      whichLetter = s[i] - 'a';
       count[whichLetter]++;
    }
    i++;
  }
}
```

Problem 7: Copy Characters (Winter 2018 Final Exam, Q11)

Write a C function called preamble that takes two parameters: a string *str* and an int-type integer n. The function will then return a new string that is dynamically allocated, and that contains at most the first n characters in the string str. For example, if *str* is "Toronto", and *n* is 3, then the function will return "Tor" (the first three characters in "Toronto"). If *str* is "Toronto" and n is 8, then the function will return "Toronto". If *str* is NULL, the function will also return NULL.

Function Prototype:

```
char *preamble(char *str, int n)
```

Example solution:

```
char* preamble(char* str, int n) {
  if (str == NULL)
    return NULL;

  char* newString = (char*)malloc((n + 1) * sizeof(char));
  int i;
  for (i = 0; str[i] != '\0' && i < n; i++) {
    newString[i] = str[i];
  }
  newString[i] = '\0';

  return newString;
}
```