Data Type Specification

- char: 1 byte
- int: 4 bytes
- short: 2 bytes
- long: 8 bytes
- long long: 8 bytes
- float: 4 bytes
- double: 8 bytes
- signed int: is int
- unsigned int: or just unsigned
1. What is the output?

typedef struct {
    char *word;
    int num;
} Unit;
    {"Hull", 4}, {"Winthrop", 8} };
Unit *up1, *up2, *up3;

up1 = &list[3];
up2 = up3 = list;

printf("1 \[%d\]n", up3->num + up1->num);

up3 = &(*(up3 + 1));

printf("2 \[%s\]n", (++up2)->word + 7);
printf("3 \[%c\]n", *(*(list + 3)).word + 1);

list[up1 - up3].word = "Utah";
list[up1 - up3].num = up1->num - (*(list + 2)).num;

printf("4 \[%x]\n", 5 * up1->num);
printf("5 \[%s\]n", list[sizeof(short)].word);

Answer:
1 [14]
2 [er]
3 [X]
4 [28]
5 [Utah]
2. What is the output on a big-endian machine? What is the output on a little-endian machine?

```c
typedef union {
    char str[4];
    uint32_t num;
} Endian;
Endian end;

end.str[0] = 'A';
end.str[1] = 'B';
end.str[2] = 'C';
end.str[3] = 'D';

printf("%s\n", end.str);
printf("%#x\n", end.num);
```

Answer:

Big-endian machine:
ABCD
0x41424344

Little-endian machine:
ABCD
0x44434241
3. We can use a binary search tree to keep track the word frequencies in a text. The convention is that the left branch holds lexicographically smaller words, and the right larger. Your task is to implement `addWord()`. If the word exists in the tree, you should increment its count. If it is a new word, you should allocate a new tree node, allocate enough memory to store the word, make a copy of it, set its count to 1, and insert the new node in the proper place. Additionally, draw the tree after the words from `Gettysburg[]` are added.

```c
char *Gettysburg[] = { "government", "of", "the", "people", "by", "the", "people", "for", "the", "people", "shall", "not", "perish", "from", "the", "earth" };

typedef struct tree {
    char *word;
    int count;
    struct tree *left, *right;
} Tree;

Tree *addWord(Tree *ptr, const char word[]) {
}

Tree root = NULL;
for (int i = 0; i < 16; i++)
    root = addWord(root, Gettysburg[i]);
```
4. Write a variadic function that generates a matrix of random numbers. The first argument is the dimension of the matrix. If it is zero, the matrix is a number. If it is one, the matrix is a one-dimensional vector, and the second argument is its length. If it is two, the matrix is two-dimensional, and the second and third arguments are its sizes. Finish the following code.

double *rndMat(int dim, ...) {
    int dim1, dim2, i, j;
    double *mat = NULL;
    va_list ap;

    va_start(ap, dim);
    if (dim == 0) {
        mat = (double*)malloc(sizeof(double));
        mat[0] = drand48();
    }
    else if (dim == 1) {
        dim1 = va_arg(ap, int);
        mat = (double*)malloc(sizeof(double) * dim1);
        for (i = 0; i < dim1; i++)
            mat[i] = drand48();
    }
    else { /*dim == 2*/
        double **mat1 = NULL;
        dim1 = va_arg(ap, int);
        dim2 = va_arg(ap, int);
        mat1 = (double**) malloc(sizeof(double *) * dim1);
        for (i = 0; i < dim1; i++) {
            mat1[i] = (double*) malloc(sizeof(double) * dim2);
            for (j = 0; j < dim2; j++)
                mat1[i][j] = drand48();
        }
        mat = (double *) mat1;
    }
    return mat;
}
5. What are defined? Explain in words. If a definition is wrong, say “it is wrong.”

1. float *ptr[5](int);
2. float *(ptr[5])(int);
3. float (*ptr)[5](int);
4. float (**ptr)[5](int);
5. float (*)(ptr[5])(int);
6. float (*ptr[5])(int);
7. float *(*ptr[5])(int);

Answer:


These are all illegal declarations such that they put functions to array

Meaning: ptr is an array of 5 pointers to functions that take an int argument return a float

Meaning: same as the previous one

Meaning: ptr is an array of 5 pointers to functions that take an int argument return a pointer to a float