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

int main(){
        /**** How to declare variable length arrays     ****/
        // Take n from the user and declare an int array of size 2*n + 3

        // Method 1: directly declare the array with length using a variable
    int n;
    scanf("%d", &n); // Take size of the array from the user
    int a[2*n + 3]; // Nice and simple - allocates space for 2*n + 3 integers
    printf("%ld",sizeof(a)); // Verify that the appropriate number of bytes (should be 8*n + 12)
did get allocated

    // Method 2: malloc (requires stdlib.h)
    int *ptr;

        // malloc requires the total number of bytes required by the array
        // malloc has no idea if we are allocating an array of int or float
        // so malloc returns a pointer of type void* - use typecasting to get it to your desired
type of pointer
    ptr = (int*)malloc(n * sizeof(int));

        // WARNING: if insufficient memory, malloc may return a NULL pointer
        // Always do a null check before using a malloc-ed array
        if(ptr == NULL)
        printf("NULL");
    else{
                // If all goes well, ptr functions just like a regular array
                ptr[3] = 40;
                printf("\n%d",*(ptr+3));
    }

        // If you no longer need this array, release this memory
        // If you are not careful about this, you may run out of memory
        free(ptr);
        // Now the bytes allocated earlier to ptr are free to be used by others (including you
again)

    // WARNING: array returned by malloc usually contains junk
        // Use calloc to allocate memory as well as fill it with zeros
    // NOTE: calloc needs to be used differently
        // Instead of total number of bytes, calloc requires size of array and size of each
element separately
    double *qtr;
    qtr = (double*)calloc(5000, sizeof(double));
        // Since we are done using the array, lets not forget to free it;
    free(qtr);

        // If you find you allocated insufficient memory using malloc or calloc
        // Use realloc to reallocate as much memory as you need
        // realloc will automatically "free" the old memory and allocate new memory
        // It will also fill up initial portion of the new memory with the old memory :)
        ptr = (int*) malloc(3 * sizeof(double));
        int i;
        for(i = 0; i < 3; i++)
                ptr[i] = i;
        // Uh oh I actually need an array of length 5
        int *tmp = (int*) realloc(ptr, 5 * sizeof(int));
        // Good practice to store realloc output into a temporary pointer
        // This way, in case memory is insufficient, we will not overwrite ptr with NULL
        // If we had written ptr = (int*) realloc(ptr, 5 * sizeof(int)); and there was
        // not enough memory, ptr would have been overwritten with NULL

        if(tmp !== NULL){ // tmp is not NULL
                ptr = tmp; // Shiny new array with 5 elements
                // Verify that the 3 old values did get copied :)
                for(i = 0; i < 3; i++)
                        printf("%d ", ptr[i]);
```

```
        }
    // Free the memory
        free(ptr);

    return 0;
}
```