

School of Computer Science and Software Engineering
Clayton Campus, Monash University

CSE1303 Part A
Summer Semester, 2002

Tutorial 3: Lists, ADTs and Dynamic Memory Solutions

Exercise 1.

```
#define MAXLIST 20

struct ElementRec
{
    float item;
    int   nextIndex;
};

typedef struct ElementRec Element;

struct ArrayLinkedListRec
{
    int     start;
    int     free;
    Element entries[MAXLIST];
};

typedef struct ArrayLinkedListRec List;
```

Exercise 2. `myItems = (Item*) malloc(10*sizeof(Item));`

Exercise 3.

```
/* Make a duplicate of s */
char*
strdup(char* s)
{
    char* newStr = (char*) malloc(strlen(s) + 1);

    if (newStr != NULL)
    {
        strcpy(newStr, s);
    }

    return newStr;
}
```

Exercise 4.

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

#define IN 1 /* inside a word */
#define OUT 0 /* outside a word */

/* Count lines, words, and characters in the input */

main()
{
    int* cPtr = (int*) malloc(sizeof(int));
    int* statePtr = (int*) malloc(sizeof(int));
    long* nlPtr = (long*) malloc(sizeof(long));
    long* nwPtr = (long*) malloc(sizeof(long));
    long* ncPtr = (long*) malloc(sizeof(long));

    if (cPtr == NULL || *statePtr == NULL || *nlPtr == NULL
        || *nwPtr == NULL || *ncPtr == NULL)
    {
        fprintf(stderr, "Out of memory\n");
        exit(1);
    }

    *statePtr = OUT;
    *nlPtr = *nwPtr = *ncPtr = 0;

    *cPtr = getchar();
    while (*cPtr != EOF)
    {
        (*ncPtr)++;
        if (*cPtr == '\n')
        {
            (*nlPtr)++;
        }

        if (*cPtr == ' ' || *cPtr == '\n' || *cPtr == '\t')
        {
            *statePtr = OUT;
        }
        else
        {
            *statePtr = IN;
            (*nwPtr)++;
        }
    }
}
```

```

    *cPtr = getchar();
}

printf("%ld %ld %ld\n", *nlPtr, *nwPtr, *ncPtr);

free(cPtr);
free(statePtr);
free(nlPtr);
free(nwPtr);
free(ncPtr);
}

```

Exercise 6.

```

struct StackRec
{
    int    top;
    int    maxSize;
    float* entry;
};

typedef struct StackRec  Stack;

/*
 * Initialize the max size of the stack to be n
 * and the stack to be empty.
 */

void
initializeStack(Stack* stackPtr, int n)
{
    stackPtr->entry = (float*)malloc(n*sizeof(float));

    if (stackPtr->entry == NULL)
    {
        fprintf(stderr, "Not enough memory\n");
        exit(1);
    }

    stackPtr->maxSize = n;
    stackPtr->top = -1;
}

```