8. COMPOUND (HIGHER LEVEL)
DATA TYPES 4 --- LINKED LISTS AND UNIONS

1) Linked records/structures
2) Linked lists, examples in Ada and C
3) Processing linked lists

One of the most useful applications of records is to define organisations of data in which distinct items are linked together using pointers or reference values.

The simplest general form of linked records is a linked list (more complex forms include trees of various types).

In a linked list each record includes a pointer to a following record.

The final pointer is a null pointer (defined using the reserved word NULL in both Ada and C) indicating that the last record in the linked list has been reached.

C LINKED LIST EXAMPLE

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

typedef struct date {  
    int day;  
    char month[9],  
    int year;  
    struct date *next;  
} DATE_T, *DATE_PTR_T;

DATE_PTR_T createBirthdayStruct(int, int, int);

void main(void) {  
    DATE_PTR_T birthdayPtr=NULL, newPtr=NULL;
    birthdayPtr=createBirthdayStruct(15,"October",  
    1991);
    newPtr=createBirthdayStruct(15,"May",1993);
    newPtr->next=birthdayPtr;
    birthdayPtr=newPtr;
    newPtr=createBirthdayStruct(21,"April",1957);
    newPtr->next=birthdayPtr;
}
```
DATE_PTR_T createDayStruct(int day, int month, int year) {
    DATE_PTR_T newPtr;
    if ((newPtr = (DATE_PTR_T) malloc(sizeof(DATE_T))) == NULL) {
        printf("Insufficient space\n");
        exit(-1);
    }
    newPtr->day = day;
    strcpy(newPtr->month, month);
    newPtr->year = year;
    newPtr->next = NULL;
    return(newPtr);
}

SUMMARY

1) Linked records/structures
2) Linked lists, examples in Ada and C
3) Processing linked lists