

# EECS 22: Advanced C Programming

## Lecture 14 (TuTh)

Rainer Dömer

doemer@uci.edu

The Henry Samueli School of Engineering  
Electrical Engineering and Computer Science  
University of California, Irvine

## Overview

- Dynamic Data Structures
  - Linked List
  - Double-linked List
  - Example: List of student records
    - `Student.h`
    - `Student.c`
    - `StudentList.h`
    - `StudentList.c`
    - `Makefile`

## Dynamic Data Structures

- Arrays
  - Static: size fixed at compile time
  - Dynamic: size fixed at time of allocation
  - Arrays cannot grow or shrink after allocation!
- Linked Lists
  - Dynamic: list length is flexible at run time
  - At program run time, list elements can be...
    - Added (allocated),
    - Removed (deleted), and
    - Moved (re-linked), as needed!
  - Example: Single-linked list
    - Each list element contains a pointer to the next element

```
struct ListItem
{
    struct ListItem *Next;
    defined_type    Data;
};
```

EECS22: Advanced C Programming, Lecture 19 (c) 2017 R. Doemer 3

## Dynamic Data Structures

- Double-Linked List (with header)
  - Example: List of Student Records
  - 1. Empty list

Length	0
First	●
Last	●

EECS22: Advanced C Programming, Lecture 19 (c) 2017 R. Doemer 4

## Dynamic Data Structures

- Double-Linked List (with header)
  - Example: List of Student Records

The diagram shows a header structure with three fields: Length (value 1), First (points to the first node), and Last (points to the first node). Below the header is a node structure with four fields: List, Next, Prev, and Student. The Student field points to a student record structure with three fields: ID (1002), Name ("John Doe"), and Grade ('C').

1. Empty list
2. Add a student

EECS22: Advanced C Programming, Lecture 19
(c) 2017 R. Doemer
5

## Dynamic Data Structures

- Double-Linked List (with header)
  - Example: List of Student Records

The diagram shows a header structure with three fields: Length (value 2), First (points to the first node), and Last (points to the second node). There are two nodes. The first node has Next pointing to the second node and Student pointing to a record with ID 1002, Name "John Doe", and Grade 'C'. The second node has Prev pointing to the first node and Student pointing to a record with ID 1003, Name "Jim Doe", and Grade 'B'.

1. Empty list
2. Add a student
3. Append a student

EECS22: Advanced C Programming, Lecture 19
(c) 2017 R. Doemer
6

## Dynamic Data Structures

- Double-Linked List (with header)
  - Example: List of Student Records

The diagram shows a header node with fields Length (3), First, and Last. Below it are three list nodes, each with fields List, Next, Prev, and Student. The First pointer of the header points to the first list node, and the Last pointer points to the last list node. The Next pointer of one list node points to the next, and the Prev pointer of one list node points to the previous. Each list node's Student pointer points to a student record table with fields ID, Name, and Grade.

ID	1001	1002	1003
Name	"Jane Doe"	"John Doe"	"Jim Doe"
Grade	'A'	'C'	'B'

1. Empty list
2. Add a student
3. Append a student
4. Prepend a student

EECS22: Advanced C Programming, Lecture 19
(c) 2017 R. Doemer
7

## Dynamic Data Structures

- Double-Linked List (with header)
  - Example: List of Student Records

The diagram shows a header node with fields Length (42), First, and Last. Below it are 42 list nodes, each with fields List, Next, Prev, and Student. The First pointer of the header points to the first list node, and the Last pointer points to the last list node. The Next pointer of one list node points to the next, and the Prev pointer of one list node points to the previous. Each list node's Student pointer points to a student record table with fields ID, Name, and Grade.

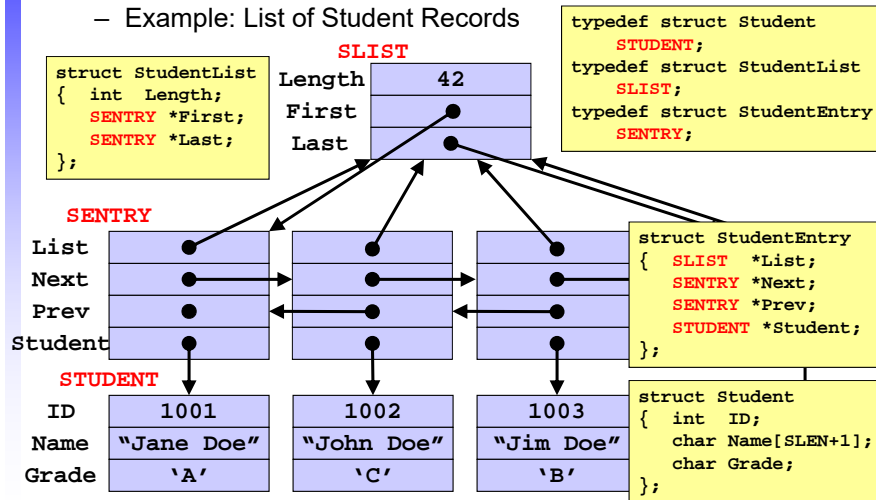
ID	1001	1002	1003	...	1042
Name	"Jane Doe"	"John Doe"	"Jim Doe"	...	"Z End"
Grade	'A'	'C'	'B'	...	'A'

1. Empty list
2. Add a student
3. Append a student
4. Prepend a student
5. ...

EECS22: Advanced C Programming, Lecture 19
(c) 2017 R. Doemer
8

## Dynamic Data Structures

- Double-Linked List (with header)
  - Example: List of Student Records



## Dynamic Data Structures

- Example `student.h`

```

/* Student.h: header file for student records */

#ifndef STUDENT_H
#define STUDENT_H

#define SLEN 40

struct Student
{
    int ID;
    char Name[SLEN+1];
    char Grade;
};
typedef struct Student STUDENT;

/* allocate a new student record */
STUDENT *NewStudent(int ID, char *Name, char Grade);

/* delete a student record */
void DeleteStudent(STUDENT *s);

/* print a student record */
void PrintStudent(STUDENT *s);

#endif /* STUDENT_H */
  
```

## Dynamic Data Structures

- Example `student.c` (part 1/3)

```

/* Student.c: maintaining student records */
#include "Student.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <assert.h>

/* allocate a new student record */
STUDENT *NewStudent(int ID, char *Name, char Grade)
{
    STUDENT *s;
    s = malloc(sizeof(STUDENT));
    if (!s)
    {
        perror("Out of memory! Aborting...");
        exit(10);
    } /* fi */
    s->ID = ID;
    strncpy(s->Name, Name, SLEN);
    s->Name[SLEN] = '\0';
    s->Grade = Grade;
    return s;
} /* end of NewStudent */
...

```

EECS22: Advanced C Programming, Lecture 19

(c) 2017 R. Doemer

11

## Dynamic Data Structures

- Example `student.c` (part 2/3)

```

...

/* delete a student record */
void DeleteStudent(STUDENT *s)
{
    assert(s);
    free(s);
} /* end of DeleteStudent */

/* print a student record */
void PrintStudent(STUDENT *s)
{
    assert(s);
    printf("Student ID:    %d\n", s->ID);
    printf("Student Name:  %s\n", s->Name);
    printf("Student Grade: %c\n", s->Grade);
} /* end of PrintStudent */

...

```

EECS22: Advanced C Programming, Lecture 19

(c) 2017 R. Doemer

12

## Dynamic Data Structures

- Example `student.c` (part 3/3)

```

...
#ifdef MAIN /* test the student record functions */
int main(void)
{
    STUDENT *s1 = NULL, *s2 = NULL;
    printf("Creating 2 student records...\n");
    s1 = NewStudent(1001, "Jane Doe", 'A');
    s2 = NewStudent(1002, "John Doe", 'C');

    printf("Printing the student records...\n");
    PrintStudent(s1);
    PrintStudent(s2);

    printf("Deleting the student records...\n");
    DeleteStudent(s1);
    s1 = NULL;
    DeleteStudent(s2);
    s2 = NULL;

    printf("Done.\n");
    return 0;
} /* end of main */
#endif /* MAIN */
/* EOF */

```

EECS22: Advanced C Programming, Lecture 19

(c) 2017 R. Doemer

13

## Dynamic Data Structures

- Example `Makefile` (part 1/2)

```

# Makefile: Student Records

# macro definitions
CC = gcc
DEBUG = -g
#DEBUG = -O2
CFLAGS = -Wall -ansi -std=c99 $(DEBUG) -c
LFLAGS = -Wall -ansi -std=c99 $(DEBUG)
MAIN = -DMAIN

# dummy targets
all: student StudentList

test: all
    valgrind ./Student
    valgrind ./StudentList

clean:
    rm -f *.o
    rm -f Student StudentList

...

```

EECS22: Advanced C Programming, Lecture 19

(c) 2017 R. Doemer

14

## Dynamic Data Structures

- Example **Makefile** (part 2/2)

```

...
# compilation rules
Student.o: Student.c Student.h
    $(CC) $(CFLAGS) Student.c -o Student.o

Student: Student.c Student.h
    $(CC) $(MAIN) $(LFLAGS) Student.c -o Student

StudentList.o: StudentList.c StudentList.h Student.h
    $(CC) $(CFLAGS) Student.c -o StudentList.o

StudentList: StudentList.c StudentList.h Student.h Student.o
    $(CC) $(MAIN) $(LFLAGS) StudentList.c Student.o \
        -o StudentList

# EOF

```

EECS22: Advanced C Programming, Lecture 19

(c) 2017 R. Doemer

15

## Dynamic Data Structures

- Example **studentList.h** (part 1/2)

```

/* StudentList.h: header file for lists of student records */
#ifndef STUDENT_LIST_H
#define STUDENT_LIST_H

#include "Student.h"

typedef struct StudentList SLIST;
typedef struct StudentEntry SENTRY;

struct StudentList
{
    int Length;
    SENTRY *First;
    SENTRY *Last;
};

struct StudentEntry
{
    SLIST *List;
    SENTRY *Next;
    SENTRY *Prev;
    STUDENT *Student;
};

...

```

EECS22: Advanced C Programming, Lecture 19

(c) 2017 R. Doemer

16



## Dynamic Data Structures

- Example `studentList.h` (part 2/2)

```

...
/* allocate a new student list */
SLIST *NewStudentList(void);

/* delete a student list (and all entries) */
void DeleteStudentList(SLIST *l);

/* print a student list */
void PrintStudentList(SLIST *l);

/* ...more functions to add/remove students... */
#endif /* STUDENT_LIST_H */
/* EOF */

```

## Dynamic Data Structures

- Example `studentList.c` (part 1/5)

```

/* StudentList.c: maintaining lists of student records */

#include "StudentList.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <assert.h>

/* allocate a new student entry */
static SENTRY *NewStudentEntry(STUDENT *s)
{
    SENTRY *e;
    e = malloc(sizeof(SENTRY));
    if (! e)
        { perror("Out of memory! Aborting...");
          exit(10);
        } /* fi */
    e->List = NULL;
    e->Next = NULL;
    e->Prev = NULL;
    e->Student = s;
    return e;
} /* end of NewStudentEntry */
...

```

## Dynamic Data Structures

- Example `studentList.c` (part 2/5)

```

.../* delete a student entry */
static STUDENT *DeleteStudentEntry(SENTRY *e)
{
    STUDENT *s;
    assert(e);
    s = e->Student;
    free(e);
    return s;
} /* end of DeleteStudentEntry */

/* allocate a new student list */
SLIST *NewStudentList(void)
{
    SLIST *l;
    l = malloc(sizeof(SLIST));
    if (! l)
        { perror("Out of memory! Aborting...");
          exit(10);
        } /* fi */
    l->Length = 0;
    l->First = NULL;
    l->Last = NULL;
    return l;
} /* end of NewStudentList */
...

```

EECS22: Advanced C Programming, Lecture 19

(c) 2017 R. Doemer 19

## Dynamic Data Structures

- Example `studentList.c` (part 3/5)

```

...
/* delete a student list (and all entries) */
void DeleteStudentList(SLIST *l)
{
    SENTRY *e, *n;
    STUDENT *s;
    ...

    assert(l);
    e = l->First;
    while(e)
        { n = e->Next;
          s = DeleteStudentEntry(e);
          DeleteStudent(s);
          e = n;
        }
    free(l);
} /* end of DeleteStudentList */
...

```

EECS22: Advanced C Programming, Lecture 19

(c) 2017 R. Doemer

20

## Dynamic Data Structures

- Example `studentList.c` (part 4/5)

```

...

/* print a student list */
void PrintStudentList(SLIST *l)
{
    SENTRY *e;
    assert(l);
    e = l->First;
    while(e)
    { PrintStudent(e->Student);
      e = e->Next;
    }
} /* end of PrintStudentList */

...

```

## Dynamic Data Structures

- Example `studentList.c` (part 5/5)

```

#ifdef MAIN
int main(void)
{
    SLIST *l = NULL;
    printf("Creating a student list...\n");
    l = NewStudentList();

    printf("Printing the empty student list...\n");
    PrintStudentList(l);

    printf("Deleting the student list...\n");
    DeleteStudentList(l);
    l = NULL;

    return 0;
} /* end of main */
#endif /* MAIN */
/* EOF */

```

## Dynamic Data Structures

- Example Session

```
% vi StudentList.c
% vi Makefile
% make StudentList
gcc -Wall -ansi -std=c99 -g -c Student.c -o Student.o
gcc -DMAIN -Wall -ansi -std=c99 -g StudentList.c Student.o
-o StudentList
StudentList.c:10: warning: 'NewStudentEntry' defined but not used
% valgrind ./StudentList
==5908== Memcheck, a memory error detector
Creating a student list...
Printing the empty student list...
Deleting the student list...
==5908== HEAP SUMMARY:
==5908==    in use at exit: 0 bytes in 0 blocks
==5908== total heap usage: 1 allocs, 1 frees, 24 bytes allocated
==5908== All heap blocks were freed -- no leaks are possible
==5908== ERROR SUMMARY: 0 errors from 0 contexts
%
```