

# EECS 22: Advanced C Programming

## Lecture 15

Rainer Dömer

doemer@uci.edu

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

## Lecture 15: Overview

- Dynamic Data Structures
  - Graphical Data Structure Display in `ddd`
  - Double-linked List
  - Element Insertion
  - Example: Student records
    - `Student.h`
    - `StudentList.h`
      - New `InsertStudentBefore`
      - New `InsertStudentAfter`
    - `StudentList.c`
      - New `InsertStudentBefore`
      - New `InsertStudentAfter`
    - `Makefile`

## Graphical Data Structure Display

- Data Display Debugger `ddd`
  - Graphical frontend for GDB
    - Requires *X* forwarding client (e.g. *Xming* in addition to *Putty*)
  - Displays separate windows
    - Menu bar and command buttons
    - Graphical display area for *pointers* and *data structures!*
    - Source code (and assembly code)
    - Command line interface
  - Can display data structures built by pointers as graphs
    - Very useful tool to visualize and debug dynamic data structures
  - Example: `StudentList.c`

```
% vi StudentList.c
% make StudentList
gcc -Wall -ansi -g -c Student.c -o Student.o
gcc -DMAIN -Wall -ansi -g StudentList.c Student.o -o StudentList
% ddd StudentList
```

EECS22: Advanced C Programming, Lecture 15

(c) 2012 R. Doemer

3

## Graphical Data Structure Display

EECS22: Advanced C Programming, Lecture 15

(c) 2012 R. Doemer

4

## Dynamic Data Structures

- Double-Linked List
  - Example: List of Student Records

Length	42
First	•
Last	•

List	•
Next	•
Prev	•
Student	•

ID	1001
Name	"Jane Doe"
Grade	'A'

ID	1002
Name	"John Doe"
Grade	'C'

ID	1003
Name	"Jim Doe"
Grade	'B'

ID	1042
Name	"Z End"
Grade	'A'

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

EECS22: Advanced C Programming, Lecture 15 (c) 2012 R. Doemer 5

## Dynamic Data Structures

- Double-Linked List
  - Example: List of Student Records

Length	42
First	•
Last	•

List	•
Next	•
Prev	•
Student	•

ID	1001
Name	"Jane Doe"
Grade	'A'

ID	1002
Name	"John Doe"
Grade	'C'

ID	1003
Name	"Jim Doe"
Grade	'B'

```

typedef struct Student
    STUDENT;
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;
};

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

EECS22: Advanced C Programming, Lecture 15 (c) 2012 R. Doemer 6

## 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 */

```

EECS22: Advanced C Programming, Lecture 15

(c) 2012 R. Doemer

7

## 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 15

(c) 2012 R. Doemer

8

## 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);

/* prepend/append a student at beginning/end of list */
void PrependStudent(SLIST *l, STUDENT *s);
void AppendStudent(SLIST *l, STUDENT *s);

/* insert a student before/after an existing one */
void InsertStudentBefore(SENTRY *e, STUDENT *s);
void InsertStudentAfter(SENTRY *e, STUDENT *s);

/* remove the first/last student from the list */
STUDENT *RemoveFirstStudent(SLIST *l);
STUDENT *RemoveLastStudent(SLIST *l);

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

#endif /* STUDENT_LIST_H */

/* EOF */

```

EECS22: Advanced C Programming, Lecture 15

(c) 2012 R. Doemer

9

## Dynamic Data Structures

- Example `studentList.c` (part 1/4)

```

/* StudentList.c: maintaining lists of student records */
/* author: Rainer Doemer */
/* modifications: */
/* 11/10/11 RD added InsertStudentBefore, InsertStudentAfter */
/* 11/08/11 RD version for double-linked lists */
/* 11/03/11 RD initial version */

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

/* unmodified functions omitted here for brevity */

...

```

EECS22: Advanced C Programming, Lecture 15

(c) 2012 R. Doemer

10

## Dynamic Data Structures

- Example `studentList.c` (part 2/4)

```

...

/* insert a student before an existing one */
void InsertStudentBefore(SENTRY *e, STUDENT *s)
{
    SENTRY *New;
    New = NewStudentEntry(s);
    New->List = e->List;
    New->Next = e;
    New->Prev = e->Prev;
    e->Prev = New;
    if (New->Prev)
    { New->Prev->Next = New;
    }
    else
    { assert(New->List->First == e);
      New->List->First = New;
    }
    New->List->Length++;
} /* end of InsertStudentBefore */

...

```

EECS22: Advanced C Programming, Lecture 15

(c) 2012 R. Doemer

11

## Dynamic Data Structures

- Example `studentList.c` (part 3/4)

```

...

/* insert a student after an existing one */
void InsertStudentAfter(SENTRY *e, STUDENT *s)
{
    SENTRY *New;
    New = NewStudentEntry(s);
    New->List = e->List;
    New->Next = e->Next;
    New->Prev = e;
    e->Next = New;
    if (New->Next)
    { New->Next->Prev = New;
    }
    else
    { assert(New->List->Last == e);
      New->List->Last = New;
    }
    New->List->Length++;
} /* end of InsertStudentAfter */

...

```

EECS22: Advanced C Programming, Lecture 15

(c) 2012 R. Doemer

12

## Dynamic Data Structures

- Example `StudentList.c` (part 4/4)

```

#ifdef MAIN
int main(void)
STUDENT *s = NULL;
SLIST *l = NULL;
SENTRY *e = NULL;
l = NewStudentList();
s = NewStudent(1001, "Jane Doe", 'A');
AppendStudent(l, s);
e = l->First;
assert(e->Student == s);
s = NewStudent(1002, "John Doe", 'C');
InsertStudentAfter(e, s);
s = NewStudent(1000, "New Kid", 'F');
InsertStudentBefore(e, s);

PrintStudentList(l);

DeleteStudentList(l);
l = NULL;
return 0;
} /* end of main */
#endif /* MAIN */
/* EOF */

```

EECS22: Advanced C Programming, Lecture 15

(c) 2012 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 $(DEBUG) -c
LFLAGS = -Wall -ansi $(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 15

(c) 2012 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

```

## Dynamic Data Structures

- Example **Session**

```

% vi StudentList.c
% vi Makefile
% make
gcc -Wall -ansi -g -c Student.c -o Student.o
gcc -DMAIN -Wall -ansi -g StudentList.c Student.o -o StudentList
% valgrind StudentList
==5908== Memcheck, a memory error detector
Student ID: 1000
Student Name: New Kid
Student Grade: F
Student ID: 1001
Student Name: Jane Doe
Student Grade: A
Student ID: 1002
Student Name: John Doe
Student Grade: C
==5908== HEAP SUMMARY:
==5908==   in use at exit: 0 bytes in 0 blocks
==5908== total heap usage: 7 allocs, 7 frees, 264 bytes allocated
==5908== All heap blocks were freed -- no leaks are possible
==5908== ERROR SUMMARY: 0 errors from 0 contexts
%

```