

EECS 22: Advanced C Programming

Lecture 16

Rainer Dömer

doemer@uci.edu

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

Part 1: Overview

- Dynamic Data Structures
 - Double-linked list
 - Element insertion before another list element
 - Element insertion after another list element
 - Element deletion (left for exercise at home)
 - Example: Student records
 - `StudentList.h`, `StudentList.c`
 - New `InsertStudentBefore`
 - New `InsertStudentAfter`
- Data Display Debugger
 - Graphical data structure display in `ddd`

Dynamic Data Structures

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

Length	0
First	●
Last	●

1. Empty list

EECS22: Advanced C Programming, Lecture 21

(c) 2017 R. Doemer

3

Dynamic Data Structures

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

Length	1
First	●
Last	●

1. Empty list
2. Append a record

List	●
Next	●
Prev	●
Student	●

ID	1002
Name	"Jim Doe"
Grade	'B'

EECS22: Advanced C Programming, Lecture 21

(c) 2017 R. Doemer

4

Dynamic Data Structures

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

1. Empty list
2. Append a record
3. Insert after a record

EECS22: Advanced C Programming, Lecture 21
(c) 2017 R. Doemer
5

Dynamic Data Structures

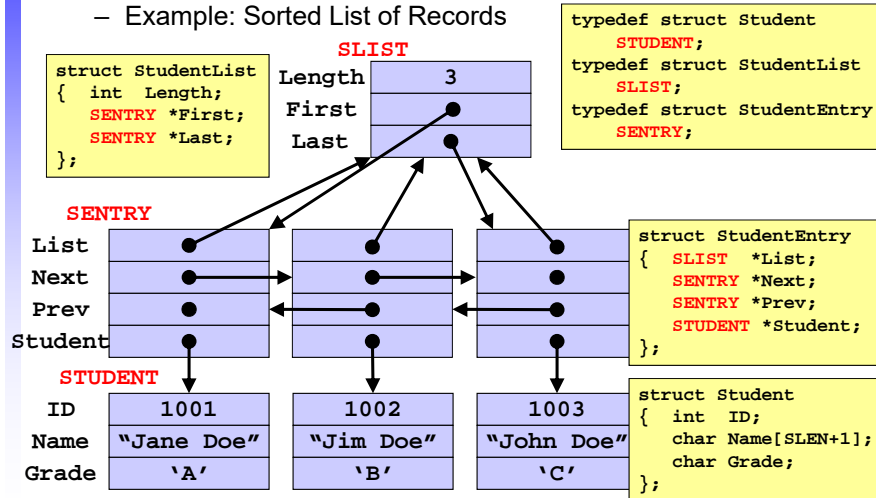
- Double-Linked List (with header)
 - Example: Sorted List of Records

1. Empty list
2. Append a record
3. Insert after a record
4. Insert before a record

EECS22: Advanced C Programming, Lecture 21
(c) 2017 R. Doemer
6

Dynamic Data Structures

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



EECS22: Advanced C Programming, Lecture 21

(c) 2017 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 21

(c) 2017 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 21

(c) 2017 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 21

(c) 2017 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 21

(c) 2017 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 21

(c) 2017 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(1002, "Jim Doe", 'B');
AppendStudent(l, s);
e = l->First;
assert(e->Student == s);
s = NewStudent(1003, "John Doe", 'C');
InsertStudentAfter(e, s);
s = NewStudent(1001, "Jane Doe", 'A');
InsertStudentBefore(e, s);

PrintStudentList(l);

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

```

EECS22: Advanced C Programming, Lecture 21

(c) 2017 R. Doemer

13

Dynamic Data Structures

- Example Session

```

% vi StudentList.c
% vi Makefile
% make
gcc -DMAIN -Wall -ansi -std=c99 -g StudentList.c Student.o
-o StudentList
% valgrind ./StudentList
==5908== Memcheck, a memory error detector
Student ID: 1001
Student Name: Jane Doe
Student Grade: A
Student ID: 1002
Student Name: Jim Doe
Student Grade: B
Student ID: 1003
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
%

```

EECS22: Advanced C Programming, Lecture 21

(c) 2017 R. Doemer

14

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) browser
 - 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 -std=c99 -g -c Student.c -o Student.o
gcc -DMAIN -Wall -ansi -std=c99 -g StudentList.c Student.o
-o StudentList
% ddd ./StudentList
```

Graphical Data Structure Display

Part 2: Overview

- Pointers to Functions

Pointers to Functions

- In C, a Function is an Object...
 - ... with a name, a type, and a location (just like a variable)
 - ... but without a value or size (meaningless for functions)
- The Address of a Function...
 - ... is the address of its first instruction in memory
 - ... can be obtained with the *address-of* operator (`&fct`)
- Function Pointers...
 - ... can be declared, defined, and initialized
 - ... can be assigned to point to defined functions
 - ... can be assigned to other pointers (of the same type)
 - ... can be passed as arguments to and from functions
 - ... can be used to call the referenced function

Pointers to Functions

- Example:

```
/* function definitions */
int add(int a, int b)
{ return a + b;
}

int multiply(int a, int b)
{ return a * b;
}

/* type definition for a pointer to a function */
typedef int (*t_OP)(int a, int b);

int main(void)
{ int n = 0;
  t_OP op = NULL; /* function pointer */

  op = &add; /* set pointer to add */
  n = (*op)(4, 17); /* call function via pointer */

  op = multiply; /* set pointer to multiply */
  n = op(2, n); /* call function via pointer */

  return n;
}
```