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

Lecture 16: Overview

- Pointers to Functions
- Dynamic Data Structures
  - Sorted Double-linked List
  - Insertion Sort
  - Example: Student records
    - **`Student.h`**
    - **`StudentList.h`**
    - **`StudentSort.h`**
    - **`StudentSort.c`**
    - **`Makefile`**

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

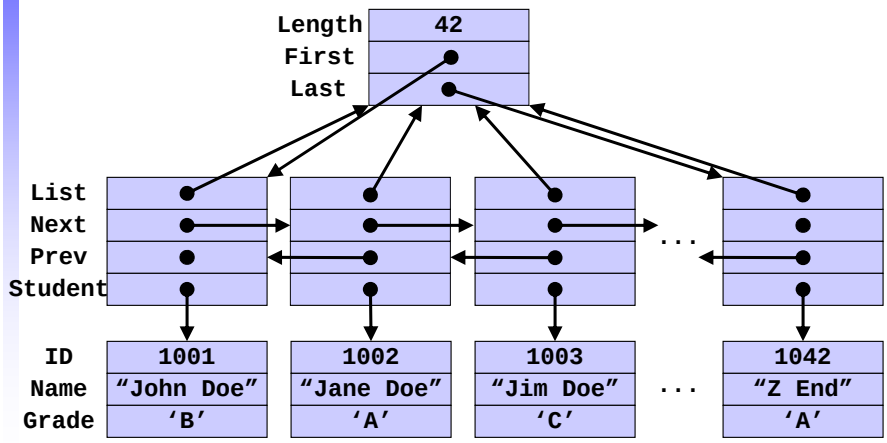
Pointers to Functions

- 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;
}
```

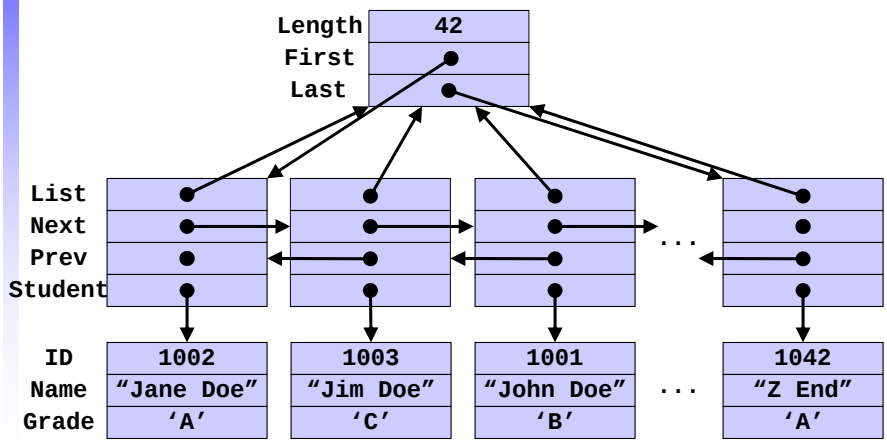
Dynamic Data Structures

- Sorted Double-Linked List
  - Example: List of Student Records sorted by ID



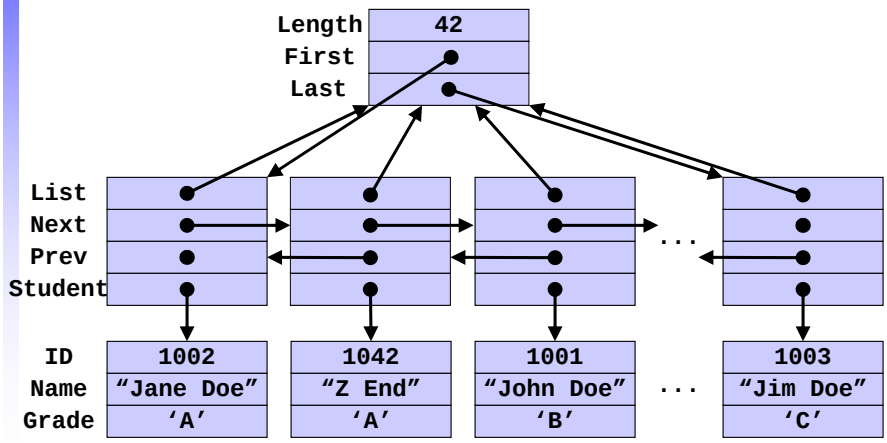
Dynamic Data Structures

- Sorted Double-Linked List
  - Example: List of Student Records sorted by Name



Dynamic Data Structures

- Sorted Double-Linked List
  - Example: List of Student Records sorted by Grade



Dynamic Data Structures

- Sorting
  - A very well-studied topic in computer science!
  - Many algorithms exist
    - Bubble Sort
    - Insertion Sort
    - Quick Sort
    - Heap Sort, …
  - For data stored in a linked list,
    Insertion Sort is a simple and effective solution
    - Start with an empty list
    - Insert items one by one while maintaining the sorted order
      - Traverse the list and compare the item with the list elements as long as the *sorting property* holds
      - Then insert the item as a new list element at this position

    ➢ Searching an item in a sorted list is twice as fast on average!

Dynamic Data Structures

- Insertion Sort, Algorithm:
  - Start with an empty list
  - Insert items one by one while maintaining the sorted order
    - Traverse the list and compare the item with the list elements as long as the *sorting property* holds
    - Then insert the item as a new list element at this position
- Insertion Sort, Pseudo Code:
  - ***InsertionSort*** *(List L)*
    - *Set List L2 = { }*
    - *For each unsorted item i ∈ L:*
      - » *InsertItem (L2, i)*
    - *Return L2*
  - ***InsertItem*** *(List L, Item i)*
    - *Repeat e = NextElement(L) until i < e;*
    - *Insert i before e into L*

Dynamic Data Structures

- Insertion Sort, Pseudo Code:
  - ***InsertionSort*** *(List L)*
    - *Set List L2 = { }*
    - *For each unsorted item i ∈ L:*
      - » *InsertItem (L2, i)*
    - *Return L2*
  - ***InsertItem*** *(List L, Item i)*
    - *Repeat e = NextElement(L) until i < e;*
    - *Insert i before e into L*

➢ To sort for a different criteria, only the comparison changes!

Dynamic Data Structures

- Insertion Sort, Pseudo Code:
  - ***InsertionSort*** *(List L, CompareFct)*
    - *Set List L2 = { }*
    - *For each unsorted item i ∈ L:*
      - » *InsertItem (L2, i, CompareFct)*
    - *Return L2*
  - ***InsertItem*** *(List L, Item i, CompareFct)*
    - *Repeat e = NextElement(L) until CompareFct(i, e);*
    - *Insert i before e into L*
  - ➢ To sort for a different criteria, only the comparison changes!
  - ➢ Instead of implementing many different sort functions, we can make the *comparison* operation a *parameter*!
  - ➢ Pass a *pointer to the comparison function* to the sort function!

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 **`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;
};

...
```


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 */
```

Dynamic Data Structures

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

```
/* StudentSort.h: header file for sorted student records       */

#ifndef STUDENT_SORT_H
#define STUDENT_SORT_H

#include "Student.h"
#include "StudentList.h"

typedef int CMP_F(STUDENT *s1, STUDENT *s2);

/* compare students by ID */
int CompareStudentID(STUDENT *s1, STUDENT *s2);

/* compare students by name */
int CompareStudentName(STUDENT *s1, STUDENT *s2);

/* compare students by grade */
int CompareStudentGrade(STUDENT *s1, STUDENT *s2);

...
```


Dynamic Data Structures

- Example **StudentSort.h** (part 2/2)

```
...

/* sort a student list */
SLIST *SortStudentList(SLIST *l, CMP_F *CompareFct);

/* find a student by ID in a sorted list */
STUDENT *FindStudentID(SLIST *l, int ID);

/* find a student by name in a sorted list */
STUDENT *FindStudentName(SLIST *l, char *Name);

/* insert a student into a sorted list */
void InsertStudent(SLIST *l, STUDENT *s, CMP_F *CompareFct);

#endif /* STUDENT_SORT_H */

/* EOF */
```

Dynamic Data Structures

- Example **StudentSort.c** (part 1/8)

```
/* StudentSort.c: sorting student records (by various criteria) */

#include "StudentSort.h"

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

/* compare students by ID */
int CompareStudentID(STUDENT *s1, STUDENT *s2)
{
    assert(s1);
    assert(s2);
    return(s1->ID - s2->ID);
} /* end of CompareStudentID */

...
```


Dynamic Data Structures

- Example **StudentSort.c** (part 2/8)

```
...

/* compare students by name */
int CompareStudentName(STUDENT *s1, STUDENT *s2)
{
    assert(s1);
    assert(s2);
    return(strcmp(s1->Name, s2->Name));
} /* end of CompareStudentName */

/* compare students by grade */
int CompareStudentGrade(STUDENT *s1, STUDENT *s2)
{
    assert(s1);
    assert(s2);
    return(s1->Grade - s2->Grade);
} /* end of CompareStudentGrade */

...
```

Dynamic Data Structures

- Example **StudentSort.c** (part 3/8)

```
...
/* sort a student list (insertion sort) */
SLIST *SortStudentList(SLIST *l, CMP_F *CompareFct)
{
    SLIST *l2 = NULL;
    STUDENT *s = NULL;

    assert(l);
    assert(CompareFct);
    l2 = NewStudentList();
    while((s = RemoveFirstStudent(l)))
      { InsertStudent(l2, s, CompareFct);
      }
    assert(l->Length == 0);
    DeleteStudentList(l);
    return(l2);
} /* end of SortStudentList */
...
```


Dynamic Data Structures

- Example **StudentSort.c** (part 4/8)

```
...
/* find a student by ID in a sorted list */
STUDENT *FindStudentID(SLIST *l, int ID)
{
    SENTRY *e;
    assert(l);
    e = l->First;
    while(e)
      { if (e->Student->ID == ID)
          { return(e->Student);
          }
        if (e->Student->ID > ID)
          { break;
          }
        e = e->Next;
      }
    return(NULL);
} /* end of FindStudentID */

...
```

Dynamic Data Structures

- Example **StudentSort.c** (part 5/8)

```
...
/* find a student by name in a sorted list */
STUDENT *FindStudentName(SLIST *l, char *Name)
{
    SENTRY *e;
    int CompVal;
    assert(l);
    assert(Name);
    e = l->First;
    while(e)
      { CompVal = strcmp(e->Student->Name, Name);
        if (CompVal == 0)
          { return(e->Student);
          }
        if (CompVal > 0)
          { break;
          }
        e = e->Next;
      }
    return(NULL);
} /* end of FindStudentName */
...
```


Dynamic Data Structures

- Example **StudentSort.c** (part 6/8)

```
...
/* insert a student into a sorted list */
void InsertStudent(SLIST *l, STUDENT *s, CMP_F *CompareFct)
{
    SENTRY *e = NULL;
    assert(l);
    assert(s);
    assert(CompareFct);
    e = l->First;
    while(e)
      { if (CompareFct(e->Student, s) > 0)
          { break;
          }
        e = e->Next;
      }
    if (e)
      { InsertStudentBefore(e, s); }
    else
      { AppendStudent(l, s); }
} /* end of InsertStudent */
...
```

Dynamic Data Structures

- Example **StudentSort.c** (part 7/8)

```
...
#ifdef MAIN

int main(void)
{
    SLIST *l = NULL;
    l = NewStudentList();
    AppendStudent(l, NewStudent(1001, "Jane Doe", 'A'));
    AppendStudent(l, NewStudent(1002, "John Doe", 'C'));
    AppendStudent(l, NewStudent(1000, "New Kid",  'F'));
    AppendStudent(l, NewStudent(1003, "Jane Doe", 'B'));
    AppendStudent(l, NewStudent(1009, "Z End",    'A'));
    AppendStudent(l, NewStudent(1008, "Alice A",  'A'));
    AppendStudent(l, NewStudent(1007, "Bob B",    'B'));
    AppendStudent(l, NewStudent(1006, "Carl C",   'C'));
    AppendStudent(l, NewStudent(1005, "Dave D",   'D'));
    AppendStudent(l, NewStudent(1004, "Eve E",    'F'));

    printf("1. Initial unsorted student list:\n");
    PrintStudentList(l);

...
```


Dynamic Data Structures

- Example **StudentSort.c** (part 8/8)

```
...
    printf("2. Student list sorted by name:\n");
    l = SortStudentList(l, CompareStudentName);
    PrintStudentList(l);
    assert(FindStudentName(l, "John Doe") != NULL);
    assert(FindStudentName(l, "No Body") == NULL);

    printf("3. Student list sorted by ID:\n");
    l = SortStudentList(l, CompareStudentID);
    PrintStudentList(l);
    assert(FindStudentID(l, 1005) != NULL);
    assert(FindStudentID(l, 1999) == NULL);

    printf("4. Student list sorted by grade:\n");
    l = SortStudentList(l, CompareStudentGrade);
    PrintStudentList(l);

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

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 StudentSort

test: all
        valgrind Student
        valgrind StudentList
        valgrind StudentSort
clean:
        rm -f *.o
        rm -f Student StudentList StudentSort
...
```


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) StudentList.c -o StudentList.o

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

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

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

# EOF
```

Dynamic Data Structures

- Example Session

```
% vi StudentSort.c
% vi Makefile
% make
gcc -DMAIN -Wall -ansi -g Student.c -o Student
gcc -Wall -ansi -g -c Student.c -o Student.o
gcc -DMAIN -Wall -ansi -g StudentList.c Student.o -o StudentList
gcc -Wall -ansi -g -c StudentList.c -o StudentList.o
gcc -DMAIN -Wall -ansi -g StudentSort.c StudentList.o Student.o \
                -o StudentSort
% StudentSort
1. Initial unsorted student list:
Student ID:    1001
Student Name:  Jane Doe
Student Grade: A
Student ID:    1002
Student Name:  John Doe
...
Student Grade: D
Student ID:    1004
Student Name:  Eve E
Student Grade: F
...
```


Dynamic Data Structures

- Example Session

```
...
2. Student list sorted by name:
Student ID:    1008
Student Name:  Alice A
Student Grade: A
Student ID:    1007
Student Name:  Bob B
Student Grade: B
Student ID:    1006
Student Name:  Carl C
Student Grade: C
Student ID:    1005
Student Name:  Dave D
Student Grade: D
...
Student ID:    1000
Student Name:  New Kid
Student Grade: F
Student ID:    1009
Student Name:  Z End
Student Grade: A
...
```

Dynamic Data Structures

- Example Session

```
...
3. Student list sorted by ID:
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
Student ID:    1003
Student Name:  Jane Doe
Student Grade: B
...
Student ID:    1008
Student Name:  Alice A
Student Grade: A
Student ID:    1009
Student Name:  Z End
Student Grade: A
...
```


Dynamic Data Structures

- Example Session

```
...
4. Student list sorted by grade:
Student ID:    1001
Student Name:  Jane Doe
Student Grade: A
Student ID:    1008
Student Name:  Alice A
Student Grade: A
Student ID:    1009
Student Name:  Z End
Student Grade: A
Student ID:    1003
Student Name:  Jane Doe
Student Grade: B
Student ID:    1007
Student Name:  Bob B
...
Student ID:    1004
Student Name:  Eve E
Student Grade: F
%
```