Week 7 Lecture 3

Arrays of Structures

Objectives

- Understand arrays of structures
  - Add an array of structures
  - Print an array of structures
  - Find an element in an array of structures
  - Modify an element in an array of structures
  - Delete an element in an array of structures

Print and Add

First: implement output, then input

- Output: Write a function to print the structure
  - Test this function by building a struct by hand
- Input: Write a function to add a structure
  - Test this function by adding then printing a structure
- Output: Print an array of structures
  - Simply call the function to print the structure in a loop
  - Test this function by calling add a few times, then printing the array
- Input: Add an array of records
  - Call the function to add a record in a loop
  - Test this function by calling the function to print an array of structures

Print the struct

```c
typedef struct student {
  int s_no;
  char *name;
  float grade;
} Student;
```

File: struct.h

```c
#include <stdio.h>
#include "struct.h"

int main()
{
  Student s;

  s.s_no = 1;
  s.name = "Nat Martin";
  s.grade = 75.6;

  printf("S_no = %d; name = %s; grade = %f\n",
         s.s_no, s.name, s.grade);
  return 0;
}
```

File: struct.c

Extract print to function

```
void print_record(Student rec)
{
  printf("S_no = %d; name = %s; grade = %f\n",
         rec.s_no,
         rec.name,
         rec.grade);
}
```

File: struct.c (partial)

Test the add function

```c
#include <stdio.h>
#include "struct.h"

void print_record(Student rec)
{
  printf("S_no = %d; name = %s; grade = %f\n",
         rec.s_no,
         rec.name,
         rec.grade);
}

Student add_record(int s_no, char *name, float grade)
{
  Student s;

  s.s_no = s_no;
  s.name = name;
  s.grade = grade;
  return s;
}

int main()
{
  Student s;

  s = add_record(1, "Nat Martin", 75.5);
  print_record(s);
}
```

File: struct.c

```
examples> make
gcc -Wall -std=c11 -g -c struct.c
gcc -Wall -std=c11 -g -o struct struct.o
examples> ./struct
S_no = 1; name = Nat Martin; grade = 75.500000
```

Output

Extract add to function

```
Student add_record(int s_no, char *name, float grade)
{
  Student s;

  s.s_no = s_no;
  s.name = name;
  s.grade = grade;
  return s;
}
```

File: struct.c (partial)

Test the function

```c
#include <stdio.h>
#include "struct.h"

void print_record(Student rec)
{
  printf("S_no = %d; name = %s; grade = %f\n",
         rec.s_no,
         rec.name,
         rec.grade);
}

int main()
{
  Student s;

  s.s_no = 1;
  s.name = "Nat Martin";
  s.grade = 75.5;

  print_record(s);
}
```

File: struct.c

```
examples> make
gcc -Wall -std=c11 -g -c struct.c
gcc -Wall -std=c11 -g -o struct struct.o
examples> ./struct
S_no = 1; name = Nat Martin; grade = 75.500000
```

Output

Moving to arrays of structs

- We need to change the name to an array of char
  - A char * does not allocate space for the name, it only allocates space for the pointer.
  - With an array of structs we need to store multiple names
- We need to change the add_record function to copy the character into the new array

Add to Array of Structs

```c
typedef struct student {
  int s_no;
  char name[30];
  float grade;
} Student;
```

File: struct.h

```c
Student add_record(int s_no, char *name, float grade)
{
  Student s;

  s.s_no = s_no;
  // s.name = name;
  for (int i = 0; name[i] != '\0' && i < 30; i++) {
    s.name[i] = name[i];
    if (name[i] == '\0' || i == 29) {
      s.name[i + 1] = '\0';
    }
  }
  s.grade = grade;
  return s;
}

int main()
{
  Student s;

  s = add_record(1, "Nat Martin", 75.5);
  print_record(s);
}
```

File: struct.c (partial)

Test the function

```c
#include <stdio.h>
#include "struct.h"

void print_record(Student rec)
{
  printf("S_no = %d; name = %s; grade = %f\n",
         rec.s_no,
         rec.name,
         rec.grade);
}

Student add_record(int s_no, char *name, float grade)
{
  Student s;

  s.s_no = s_no;
  // s.name = name;
  for (int i = 0; name[i] != '\0' && i < 30; i++) {
    s.name[i] = name[i];
    if (name[i] == '\0' || i == 29) {
      s.name[i + 1] = '\0';
    }
  }
  s.grade = grade;
  return s;
}

int main()
{
  Student s;

  s = add_record(1, "Nat Martin", 75.5);
  print_record(s);
}
```

File: struct.c

```
examples> make
gcc -Wall -std=c11 -g -c struct.c
gcc -Wall -std=c11 -g -o struct struct.o
examples> ./struct
S_no = 1; name = Nat Martin; grade = 75.500000
```

Output

Moving to arrays of structs

- We need to add an array of structs
- We need to print out the array

Printing Array of Structs

```
Student class[30];
int class_size = 0;

void print_record(Student rec)
{
  printf("S_no = %d; name = %s; grade = %f\n",
         rec.s_no,
         rec.name,
         rec.grade);
}

void print_db()
{
  for (int i = 0; i < class_size; i++) {
    print_record(class[i]);
  }
}
```

File: struct.c (partial)

Test the function

```
Student class[30];
int class_size = 0;

void print_record(Student rec)
{
  printf("S_no = %d; name = %s; grade = %f\n",
         rec.s_no,
         rec.name,
         rec.grade);
}

void print_db()
{
  for (int i = 0; i < class_size; i++) {
    print_record(class[i]);
  }
}
```

```
int main()
{
  class[0] = add_record(1, "Nat Martin", 75.5);
  class[1] = add_record(2, "Good Student", 95.5);
  class[2] = add_record(3, "Poor Student", 55.5);
  class_size = 3;
  print_db(class, class_size);
}
```

File: struct.c (partial)

```
examples> make
gcc -Wall -std=c11 -g -c struct.c
gcc -Wall -std=c11 -g -o struct struct.o
examples> ./struct
S_no = 1; name = Nat Martin; grade = 75.500000
S_no = 2; name = Good Student; grade = 95.500000
S_no = 3; name = Poor Student; grade = 55.500000
```

Output

Initializing a Database

- Now that we can insert students into the database we want to add a class
- Create a new function that will add 10 students
  - We will need to create new names for the students.
  - We create names by manipulating the string that will be the name

Initializing the database

```
void fill_db(void)
{
  char name[80];
  char *stud_str = "Student ";

  for (int i = 0; i < 10; i++) {
    for (int str_i = 0; str_i < 8; str_i++) {
      name[str_i] = stud_str[str_i];
    }
    name[8] = (char)('0' + i);
    name[9] = '\0';
    add_rec_to_class(i, name, 99.5 - i);
  }
}
```

File: struct.c (partial)

Test the function

```
void fill_db(void)
{
  char name[80];
  char *stud_str = "Student ";

  for (int i = 0; i < 10; i++) {
    for (int str_i = 0; str_i < 8; str_i++) {
      name[str_i] = stud_str[str_i];
    }
    name[8] = (char)('0' + i);
    name[9] = '\0';
    add_rec_to_class(i, name, 99.5 - i);
  }
}

int main()
{
  fill_db();
  print_db(class, class_size);
}
```

File: struct.c (partial)

```
examples> make
gcc -Wall -std=c11 -g -c struct.c
gcc -Wall -std=c11 -g -o struct struct.o
examples> ./struct
S_no = 0; name = Student 0; grade = 99.500000
S_no = 1; name = Student 1; grade = 98.500000
S_no = 2; name = Student 2; grade = 97.500000
S_no = 3; name = Student 3; grade = 96.500000
S_no = 4; name = Student 4; grade = 95.500000
S_no = 5; name = Student 5; grade = 94.500000
S_no = 6; name = Student 6; grade = 93.500000
S_no = 7; name = Student 7; grade = 92.500000
S_no = 8; name = Student 8; grade = 91.500000
S_no = 9; name = Student 9; grade = 90.500000
```

Output

Find

Finding an Element in the Array

- Now that we have a class we can search in it.
- Create a find a student by serial number
  - Look at all of the elements
  - Return when you find one
  - Return the index of the element found
  - Return -1 (an illegal index) is not found

Searching the array

```
int find(int target)
{
  for (int i = 0; i < class_size; i++) {
    if (class[i].s_no == target) {
      return i;
    }
  }
  return -1;
}
```

File: struct.c (partial)

Test the function

```c
int find(int target)
{
  for (int i = 0; i < class_size; i++) {
    if (class[i].s_no == target) {
      return i;
    }
  }
  return -1;
}

int main()
{
  int found;

  fill_db();
  found = find(1);
  if (found == -1) {
    printf("Target 1 not found\n");
  } else {
    print_record(class[found]);
  }
}
```

File: struct.c (partial)

```
examples> make
gcc -Wall -std=c11 -g -c struct.c
gcc -Wall -std=c11 -g -o struct struct.o
examples> ./struct
S_no = 1; name = Student 1; grade = 98.500000
```

Output

A single test is not enough

- We need to test the function throughly
  - Search for first element
  - Search for element in middle
  - Search for last element
  - Search for an element not in array
- Extract the test function

- Good tests elicit failure
  - Boundary conditions
    - Succeed but the next fails
    - E.g., first and last elements of the array
  - Failure conditions:
    - Should fail
    - E.g., elements not in the array.
  - Normal condition
    - Should succeed
      - must be selected
    - E.g. element found in middle

Extract the test function

```
void print_if_found(int target)
{
  int found;

  found = find(target, class, class_size);
  if (found == -1) {
    printf("Target 1 not found\n");
  } else {
    print_record(class[found]);
  }
}
```

File: struct.c (partial)

Run tests

```c
void print_if_found(int target)
{
  int found;

  found = find(target, class, class_size);
  if (found == -1) {
    printf("Target 1 not found\n");
  } else {
    print_record(class[found]);
  }
}

int main()
{
  fill_db();
  print_if_found(0);
  print_if_found(5);
  print_if_found(9);
  print_if_found(10);
  // print_db(class, class_size);
}
```

File: struct.c (partial)

```
Target 1 not foundexamples> make
gcc -Wall -std=c11 -g -c struct.c
gcc -Wall -std=c11 -g -o struct struct.o
examples> ./struct
S_no = 0; name = Student 0; grade = 99.500000
S_no = 5; name = Student 5; grade = 94.500000
S_no = 9; name = Student 9; grade = 90.500000
Target 1 not found
```

Output

Modify

Modify

- Find the serial number
- Add a new record if not found.
- Change the other values if found.
- Return index of modified record (-1 if not found)

Modify implementation

```
int modify(int s_no, char *name, float grade)
{
  int found = -2;

  found = find(s_no);
  if (found < 0) {
    add_rec_to_class(s_no, name, grade);
  } else {
    class[found] = add_record(s_no, name, grade);
  }
  return found;
}
```

File: struct.c (partial)

Run tests

```c
int modify(int s_no, char *name, float grade)
{
  int found = -2;

  found = find(s_no);
  if (found < 0) {
    add_rec_to_class(s_no, name, grade);
  } else {
    class[found] = add_record(s_no, name, grade);
  }
  return found;
}

int main()
{
  fill_db();
  printf("Before modification:\n");
  print_db();
  modify(0, "New 0", 10.5);
  modify(5, "New 5", 15.5);
  modify(9, "New 9", 19.5);
  modify(10, "New 10", 20.5);
  printf("After modification:\n");
  print_db();
}
```

File: struct.c (partial)

```
examples> make
gcc -Wall -std=c11 -g -c struct.c
gcc -Wall -std=c11 -g -o struct struct.o
examples> ./struct
Before modification:
S_no = 0; name = Student 0; grade = 99.500000
S_no = 1; name = Student 1; grade = 98.500000
S_no = 2; name = Student 2; grade = 97.500000
S_no = 3; name = Student 3; grade = 96.500000
S_no = 4; name = Student 4; grade = 95.500000
S_no = 5; name = Student 5; grade = 94.500000
S_no = 6; name = Student 6; grade = 93.500000
S_no = 7; name = Student 7; grade = 92.500000
S_no = 8; name = Student 8; grade = 91.500000
S_no = 9; name = Student 9; grade = 90.500000
After modification:
S_no = 0; name = New 0; grade = 10.500000
S_no = 1; name = Student 1; grade = 98.500000
S_no = 2; name = Student 2; grade = 97.500000
S_no = 3; name = Student 3; grade = 96.500000
S_no = 4; name = Student 4; grade = 95.500000
S_no = 5; name = New 5; grade = 15.500000
S_no = 6; name = Student 6; grade = 93.500000
S_no = 7; name = Student 7; grade = 92.500000
S_no = 8; name = Student 8; grade = 91.500000
S_no = 9; name = New 9; grade = 19.500000
S_no = 10; name = New 10; grade = 20.500000
```

Output

Delete

Delete

- Find the serial number
- Return -1 if not found
- Move all of the records following the one found up one space
  - The delete one will be overwritten
  - The following ones will overwrite it
- Reduce the number of elements in the array by 1.

Delete implementation

```
int delete(int s_no)
{
  int found = -2;

  found = find(s_no);
  if (found == -1) {
    return found;
  }
  for (int i = found; i < class_size; i++) {
    class[i] = class[i+1];
  }
  return class_size -= 1;
}
```

File: struct.c (partial)

Run tests

```c
int delete(int s_no)
{
  int found = -2;

  found = find(s_no);
  if (found == -1) {
    return found;
  }
  for (int i = found; i < class_size; i++) {
    class[i] = class[i+1];
  }
  return class_size -= 1;
}

int main()
{
  fill_db();
  printf("Before modification:\n");
  print_db();
  delete(9);
  delete(5);
  delete(0);
  delete(10);
  printf("After modification:\n");
  print_db();
}
```

File: struct.c (partial)

```
examples> make
gcc -Wall -std=c11 -g -c struct.c
gcc -Wall -std=c11 -g -o struct struct.o
examples> ./struct
Before modification:
S_no = 0; name = Student 0; grade = 99.500000
S_no = 1; name = Student 1; grade = 98.500000
S_no = 2; name = Student 2; grade = 97.500000
S_no = 3; name = Student 3; grade = 96.500000
S_no = 4; name = Student 4; grade = 95.500000
S_no = 5; name = Student 5; grade = 94.500000
S_no = 6; name = Student 6; grade = 93.500000
S_no = 7; name = Student 7; grade = 92.500000
S_no = 8; name = Student 8; grade = 91.500000
S_no = 9; name = Student 9; grade = 90.500000
After modification:
S_no = 1; name = Student 1; grade = 98.500000
S_no = 2; name = Student 2; grade = 97.500000
S_no = 3; name = Student 3; grade = 96.500000
S_no = 4; name = Student 4; grade = 95.500000
S_no = 6; name = Student 6; grade = 93.500000
S_no = 7; name = Student 7; grade = 92.500000
S_no = 8; name = Student 8; grade = 91.500000
```

Output