EECS 10: Computational Methods in Electrical and Computer Engineering

Lecture 14

Rainer Dömer

doemer@uci.edu

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

Lecture 14: Overview

- Recursion
  - Concept of recursion
  - Recursion vs. iteration
  - Examples
    - Factorial function: **`Factorial.c`**
    - Fibonacci series: **`Fibonacci.c`**
- Data Structures
  - Structures, unions, enumerators
  - Type definitions
  - Example
    - Student records: **`Students.c`**

Recursion

- Introduction
  - *Recursion* is often an alternative to *Iteration*
  - Recursion is a very simple concept, yet very powerful
  - Recursion is present in nature
    - Trees have branches, which have branches, which have branches, ... which have leaves.
  - Recursion is traversal of hierarchy
    - *Traverse* (climb) a tree to the top:
      - start at the root
      - at a leaf, stop
      - at a branch, *traverse* one branch
    - *Traverse* a file system on a computer
      - start at the current directory
      - at a file, process the file
      - at a directory, *traverse* the directory

Recursion

- Recursive Function
  - Function that calls itself
    - ... directly, or
    - ... indirectly

```
int f(...)
  { ...
    f(...);
    ...
  }
```

```
int a(...)
  { ...
    b(...);
    ...
  }
int b(...)
  { ...
    a(...);
    ...
  }
```

- Concept of Recursion
  - Trivial *base case*
    - Return value defined for simple case
    - Example: `if (param==0) {return 1;}`
  - *Recursion step*
    - Reduce the problem towards the base case
    - Make a recursive function call
    - Example: `if (param>0) {return ...fct(param-1);}`
- Termination of Recursion
  - Converging of recursive calls to the base case
  - Recursive call must be “simpler” than current call

Recursion

- Example: Factorial function *n!*
  - The factorial of a non-negative integer is
    - *n! = n * (n-1) * (n-2) * (n-3) * ... * 1*
  - This can be written as
    - *n! = n * ((n-1) * ((n-2) * ((n-3) * (... * 1))))*
  - *Recursive* definition:
    - *n=1: 1! = 1* (base case)
    - *n>1: n! = n * (n-1)!* (recursion step)
  - Example computation:

```
5! = 5 * 4!
   = 5 * (4 * 3!)
   = 5 * (4 * (3 * 2!))
   = 5 * (4 * (3 * (2 * 1!)))
   = 5 * (4 * (3 * (2 * 1)))
   = 5 * (4 * (3 * 2))
   = 5 * (4 * 6)
   = 5 * 24
   = 120
```


Recursion

- Program example: **`Factorial.c`** (part 1/2)

```
/* Factorial.c: example demonstrating recursion */
/* author: Rainer Doemer                        */
/* modifications:                               */
/* 11/14/04 RD  initial version                 */

#include <stdio.h>

/* function definition */

long factorial(long n)
{
   if (n == 1)          /* base case */
      { return 1;
       } /* fi */
   else                 /* recursion step */
      { return n * factorial(n-1);
       } /* esle */
} /* end of factorial */

...
```

Recursion

- Program example: **`Factorial.c`** (part 2/2)

```
...
int main(void)
{
   /* variable definitions */
   long int n, f;

   /* input section */
   printf("Please enter value n: ");
   scanf("%ld", &n);

   /* computation section */
   f = factorial(n);

   /* output section */
   printf("The factorial of %ld is %ld.\n", n, f);

   /* exit */
   return 0;
} /* end of main */

/* EOF */
```


Recursion

- Example session: **`Factorial.c`**

```
% vi Factorial.c
% gcc Factorial.c -o Factorial -Wall -ansi
% Factorial
Please enter value n: 1
The factorial of 1 is 1.
% Factorial
Please enter value n: 2
The factorial of 2 is 2.
% Factorial
Please enter value n: 3
The factorial of 3 is 6.
% Factorial
Please enter value n: 5
The factorial of 5 is 120.
% Factorial
Please enter value n: 10
The factorial of 10 is 3628800.
%
```

Recursion vs. Iteration

- Example: Factorial function *n!*
  - The factorial of a non-negative integer is
    - *n! = n * (n-1) * (n-2) * (n-3) * ... * 1*
  - This can be written as
    - *n! = n * ((n-1) * ((n-2) * ((n-3) * (... * 1))))*
  - *Recursive* definition:
    - *n=1: 1! = 1* (base case)
    - *n>1: n! = n * (n-1)!* (recursion step)
  - *Iterative* implementation:
    - Compute *n* products in a loop
    - *n! = n * (n-1) * (n-2) * (n-3) * ... * 1*
    -
```
p = n;
for (f=n-1; f>=1; f--)
   { p = p * f; }
```


Recursion vs. Iteration

- Program example: **`Factorial2.c`** (part 1/2)

```
/* Factorial2.c: example demonstrating iteration       */
/* author: Rainer Doemer                               */
/* modifications:                                      */
/* 11/14/04 RD  initial version (based on Factorial.c) */

#include <stdio.h>

/* function definition */

long factorial(long n)
{
   long product, factor;

   product = n;
   for(factor = n-1; factor >= 1; factor--)
      { product *= factor;
       } /* rof */
   return product;
} /* end of factorial */

...
```

Recursion vs. Iteration

- Program example: **`Factorial2.c`** (part 2/2)

```
...
int main(void)
{
   /* variable definitions */
   long int n, f;

   /* input section */
   printf("Please enter value n: ");
   scanf("%ld", &n);

   /* computation section */
   f = factorial(n);

   /* output section */
   printf("The factorial of %ld is %ld.\n", n, f);

   /* exit */
   return 0;
} /* end of main */

/* EOF */
```


Recursion vs. Iteration

- Example session: **`Factorial2.c`**

```
% cp Factorial.c Factorial2.c
% vi Factorial2.c
% gcc Factorial2.c -o Factorial2 -Wall -ansi
% Factorial2
Please enter value n: 1
The factorial of 1 is 1.
% Factorial2
Please enter value n: 2
The factorial of 2 is 2.
% Factorial2
Please enter value n: 3
The factorial of 3 is 6.
% Factorial2
Please enter value n: 5
The factorial of 5 is 120.
% Factorial2
Please enter value n: 10
The factorial of 10 is 3628800.
%
```

Recursion

- Example 2: Fibonacci series
  - Sequence of integers
    - 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, ...
  - Mathematical properties
    - The first two numbers are 0 and 1
    - Every subsequent Fibonacci number is the sum of the previous two Fibonacci numbers
  - Ratio of successive Fibonacci numbers is ...
    - ... converging to constant value 1.618...
    - ... called *Golden Ratio* or *Golden Mean*
  - Recursive definition:
    - Base case: $fibonacci(0) = 0$, $fibonacci(1) = 1$
    - Recursion step: $fibonacci(n) = fibonacci(n-1) + fibonacci(n-2)$

Recursion

- Program example: **`Fibonacci.c`** (part 1/2)

```
/* Fibonacci.c: example demonstrating recursion */
/* author: Rainer Doemer                        */
/* modifications:                               */
/* 11/14/04 RD  initial version                 */

#include <stdio.h>

/* function definition */

long fibonacci(long n)
{
   if (n <= 1)  /* base case */
      { return n;
       } /* fi */
   else         /* recursion step */
      { return fibonacci(n-1) + fibonacci(n-2);
       } /* esle */
} /* end of fibonacci */

/* main function */
...
```

Recursion

- Program example: **`Fibonacci.c`** (part 2/2)

```
...
int main(void)
{
   /* variable definitions */
   long int n, f;

   /* input section */
   printf("Please enter value n: ");
   scanf("%ld", &n);

   /* computation section */
   f = fibonacci(n);

   /* output section */
   printf("The %ld-th Fibonacci number is %ld.\n", n, f);

   /* exit */
   return 0;
} /* end of main */

/* EOF */
```


Recursion

- Example session: **`Fibonacci.c`**

```
% cp Factorial.c Fibonacci.c
% vi Fibonacci.c
% gcc Fibonacci.c -o Fibonacci -Wall -ansi
% Fibonacci
Please enter value n: 1
The 1-th Fibonacci number is 1.
% Fibonacci
Please enter value n: 10
The 10-th Fibonacci number is 55.
% Fibonacci
Please enter value n: 20
The 20-th Fibonacci number is 6765.
% Fibonacci
Please enter value n: 30
The 30-th Fibonacci number is 832040.
% Fibonacci
Please enter value n: 40
The 40-th Fibonacci number is 102334155.
%
```

Data Structures

- Structures (aka. records): **`struct`**
  - User-defined, composite data type
    - Type is a composition of (different) sub-types
  - Fixed set of members
    - Names and types of members are fixed at structure definition
  - Member access by name
    - Member-access operator: ***`structure_name.member_name`***
- Example:

```
struct S { int i; float f;} s1, s2;

s1.i = 42;      /* access to members */
s1.f = 3.1415;
s2 = s1;        /* assignment */
s1.i = s1.i + 2*s2.i;
```


Data Structures

- Structure Declaration
  - Declaration of a user-defined data type
- Structure Definition
  - Definition of structure members and their type
- Structure Instantiation and Initialization
  - Definition of a variable of structure type
  - Initializer list defines initial values of members
- Example:

```
struct Student;          /* declaration */

struct Student           /* definition */
{ int   ID;              /* members */
  char  Name[40];
  char  Grade;
};

struct Student Jane =    /* instantiation */
{1001, "Jane Doe", 'A'}; /* initialization */
```

Data Structures

- Structure Access
  - Members are accessed by their name
  - Member-access operator .
- Example:

```
struct Student
{  int  ID;
   char Name[40];
   char Grade;
};

struct Student Jane =
{1001, "Jane Doe", 'A'};

void PrintStudent(struct Student s)
{
   printf("ID:    %d\n", s.ID);
   printf("Name:  %s\n", s.Name);
   printf("Grade: %c\n", s.Grade);
}
```

Jane	
ID	1001
Name	"Jane Doe"
Grade	'A'

```
ID:    1001
Name:  Jane Doe
Grade: A
```

EECS10: Computational Methods in ECE, Lecture 14 (c) 2018 R. Doemer 19

Data Structures

- Unions: **`union`**
  - User-defined, composite data type
    - Type is a composition of (different) sub-types
  - Fixed set of *mutually exclusive* members
    - Names and types of members are fixed at union definition
  - Member access by name
    - Member-access operator: ***`union_name.member_name`***
  - *Only one member may be used at a time!*
    - *All members share the same location in memory!*
- Example:

```
union U { int i; float f;} u1, u2;

u1.i = 42;      /* access to members */
u2.f = 3.1415;
u1.f = u2.f;    /* destroys u1.i! */
```

EECS10: Computational Methods in ECE, Lecture 14 (c) 2018 R. Doemer 20

Data Structures

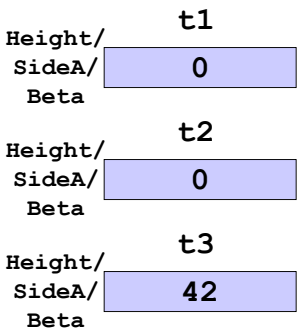
- Union Declaration
  - Declaration of a user-defined data type
- Union Definition
  - Definition of union members and their type
- Union Instantiation and Initialization
  - Definition of a variable of union type
  - *Single* initializer defines value of *first* member
- Example:

```
union HeightOfTriangle;  /* declaration */

union HeightOfTriangle   /* definition */
{ int   Height;          /* members */
  int   LengthOfSideA;
  float AngleBeta;
};

union HeightOfTriangle H /* instantiation */
= { 42 };                /* initialization */
```


Data Structures

- Union Access
  - Members are accessed by their name
  - Member-access operator .
- Example:

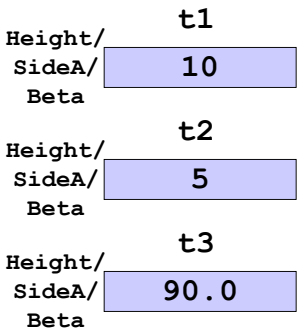
```
union HeightOfTriangle
{ int   Height;
  int   SideA;
  float Beta;
};

union HeightOfTriangle t1, t2, t3
= { 42 };
```



Data Structures

- Union Access
  - Members are accessed by their name
  - Member-access operator .
- Example:

```
union HeightOfTriangle
{ int   Height;
  int   SideA;
  float Beta;
};

union HeightOfTriangle t1, t2, t3
= { 42 };

void SetHeight(void)
{
   t1.Height = 10;
   t2.SideA = t1.Height / 2;
   t3.Beta = 90.0;
}
```



Data Structures

- Enumerators: **enum**
  - User-defined data type
    - Members are an enumeration of integral constants
  - Fixed set of members
    - Names and values of members are fixed at enumerator definition
  - Members are constants
    - Member values cannot be changed after definition
- Example:

```
enum E { red, yellow, green };
enum E LightNS, LightEW;

LightEW = green;        /* assignment */
if (LightNS == green)   /* comparison */
   { LightEW = red; }
```

Data Structures

- Enumerator Declaration
  - Declaration of a user-defined data type
- Enumerator Definition
  - Definition of enumerator members and their value
- Enumerator Instantiation and Initialization
  - Definition of a variable of enumerator type
  - Initializer should be one member of the enumerator
- Example:

```
enum Weekday;              /* declaration */

enum Weekday               /* definition */
{ Monday, Tuesday,         /* members */
  Wednesday, Thursday,
  Friday, Saturday, Sunday
};

enum Weekday Today         /* instantiation */
= Tuesday;                 /* initialization */
```


Data Structures

- Enumerator Values
  - Enumerators are represented as integer constants
  - By default, enumerator values start at 0 and are incremented by 1 for each following member
- Example:

```
enum Weekday
{ Monday,
  Tuesday,
  Wednesday,
  Thursday,
  Friday,
  Saturday,
  Sunday
};

enum Weekday Today
= Tuesday;

void PrintWeekday(
     enum Weekday d)
{
  printf("Day: %d\n", d);
}
```

```
Today
Tuesday

Day: 1
```

Data Structures

- Enumerator Values
  - Enumerators are represented as integer constants
  - By default, enumerator values start at 0 and are incremented by 1 for each following member
  - Specific enumerator values may be defined by the user
- Example:

```
Today
```

```
Tuesday
```

```
Day: 2
```

```
enum Weekday
{ Monday = 1,
  Tuesday,
  Wednesday,
  Thursday,
  Friday,
  Saturday,
  Sunday
};

enum Weekday Today
= Tuesday;

void PrintWeekday(
     enum Weekday d)
{
  printf("Day: %d\n", d);
}
```


Data Structures

- Enumerator Values
  - Enumerators are represented as integer constants
  - By default, enumerator values start at 0 and are incremented by 1 for each following member
  - Specific enumerator values may be defined by the user
- Example:

```
Today
```

```
Tuesday
```

```
Day: 3
```

```
enum Weekday
{ Monday = 2,
  Tuesday,
  Wednesday,
  Thursday,
  Friday,
  Saturday,
  Sunday = 1
};

enum Weekday Today
= Tuesday;

void PrintWeekday(
     enum Weekday d)
{
  printf("Day: %d\n", d);
}
```

Data Structures

- Type definitions: **`typedef`**
  - A *typedef* can be defined as an alias type for another type
  - A *typedef* definition follows the same rules as a variable definition
  - Type definitions are usually used to abbreviate access to user-defined types
- Examples:

```
typedef signed long int MyInteger;
MyInteger i1 = 42;

typedef enum Weekday Day;
Day Today = Tuesday;

typedef struct Student Scholar;
Scholar Jane, John;
```


Data Structures

- Program example: **`Students.c`** (part 1/6)

```
/* Students.c: simple array of student records  */
/* author: Rainer Doemer                        */
/* modifications:                               */
/* 08/28/13 RD initial version                  */

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

/* constants */

#define SLEN 40
#define MAX 100

/* data structures */

struct Student
{  int  ID;
   char Name[SLEN];
   int  Score;
};
typedef struct Student STUDENT;

STUDENT Record[MAX];
int N = 0;
...
```

Data Structures

- Program example: **`Students.c`** (part 2/6)

```
...
/* function declarations */

STUDENT EnterStudent(void);
void PrintStudent(STUDENT s);
char LetterGrade(int Score);
void InsertStudent(STUDENT s);

/* function definitions */

STUDENT EnterStudent(void)
{
    STUDENT s;

    printf("Enter student ID:    ");
    scanf("%d", &s.ID);
    printf("Enter student name:  ");
    scanf("%39s", &s.Name[0]);
    printf("Enter student score: ");
    scanf("%d", &s.Score);
    return s;
}

...
```


Data Structures

- Program example: **`Students.c`** (part 3/6)

```
...
void PrintStudent(STUDENT s)
{
    printf("ID %3d: %-39s, Score %3d%% = %c\n",
                s.ID, s.Name, s.Score, LetterGrade(s.Score));
}

char LetterGrade(int Score)
{   switch(Score/10)
    {   case 10:
        case  9: return 'A';
        case  8: return 'B';
        case  7: return 'C';
        case  6: return 'D';
        case  5: case 4: case 3: case 2: case 1:
        case  0: return 'F';
        default: break;
    } /* hctiws */
    return '-';
}
...
```

Data Structures

- Program example: **`Students.c`** (part 4/6)

```
...

void InsertStudent(STUDENT s)
{
    int i, j;

    for(i=0; i<N; i++)
    {
        if (s.ID < Record[i].ID)
        { break;
        }
    }
    for(j=N; j>i; j--)
    {
        Record[j] = Record[j-1];
    }
    Record[i] = s;
    N++;
}

...
```


Data Structures

- Program example: **`Students.c`** (part 5/6)

```
...

int main(void)
{
    int Choice, i;

    while(1)
    {   printf("Student records: %d\n", N);
        printf("1. Enter new student\n");
        printf("2. Print student table\n");
        printf("3. Quit\n");
        printf("Choice: ");
        scanf("%d", &Choice);
        switch(Choice)
        { case 1:
            {   if (N < MAX)
                {   InsertStudent(EnterStudent());
                }
                break;
            }
...
```

Data Structures

- Program example: **`Students.c`** (part 6/6)

```
...
          case 2:
            {   for(i=0; i<N; i++)
                { PrintStudent(Record[i]);
                }
                break;
            }
          case 3:
            {   exit(0);
            }
          default:
            {   break;
            }
        } /* hctiws */
    } /* elihw */
    return 0;
} /* end of main */

/* EOF */
```


Data Structures

- Example session: **`Students.c`** (part 1/3)

```
% vi Students.c
% gcc Students.c -o Students -Wall -ansi
% Students
Student records: 0
1. Enter new student
2. Print student table
3. Quit
Choice: 1
Enter student ID:    1879
Enter student name:  Albert_Einstein
Enter student score: 100
Student records: 1
1. Enter new student
2. Print student table
3. Quit
Choice: 2
ID 1879: Albert_Einstein                , Score 100% = A
...
```

Data Structures

- Example session: **`Students.c`** (part 2/3)

```
...
Student records: 1
1. Enter new student
2. Print student table
3. Quit
Choice: 1
Enter student ID:    1642
Enter student name:  Isaac_Newton
Enter student score: 100
Student records: 2
1. Enter new student
2. Print student table
3. Quit
Choice: 2
ID 1642: Isaac_Newton                  , Score 100% = A
ID 1879: Albert_Einstein               , Score 100% = A
...
```


Data Structures

- Example session: **`Students.c`** (part 3/3)

```
[...]
Choice: 1
Enter student ID:    1623
Enter student name:  Blaise_Pascal
Enter student score: 100
Student records: 3
1. Enter new student
2. Print student table
3. Quit
Choice: 2
ID 1623: Blaise_Pascal                 , Score 100% = A
ID 1642: Isaac_Newton                  , Score 100% = A
ID 1879: Albert_Einstein               , Score 100% = A
Student records: 3
1. Enter new student
2. Print student table
3. Quit
Choice: 3
%
```