

# EECS 10: Computational Methods in Electrical and Computer Engineering

## Review of Lectures 9 - 17

Rainer Dömer

doemer@uci.edu

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

## Review of Lectures 9 - 17

- Lecture 9: Formatted output
- Lecture 10: Structured programming, conditions
- Lecture 11: Structured programming, loops
- Lecture 12: Jump statements, debugging
- Lecture 13: Functions, terms and concepts
- Lecture 14: Functions, scope rules, stack frames
- Lecture 15: Standard library functions
- Lecture 16: Data structures, arrays
- Lecture 17: Passing arrays to functions, strings

## Formatted Output

- Formatted output using `printf()`
  - standard format specifiers for integral values
    - `unsigned long long`      `%llu`
    - `long long`                `%lld`
    - `unsigned long`            `%lu`
    - `long`                        `%ld`
    - `unsigned int`              `%u`
    - `int`                         `%d`
    - `short`                     `%hd`
  - standard format specifiers for floating point values
    - `long double`            `%Lf`
    - `double`                    `%f`
    - `float`                     `%f`

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

3

## Formatted Output

- Detailed formatting sequence for integral values
  - `% flags width length conversion`
  - **flags**
    - (none) standard formatting (right-justified)
    - `-` left-justified output
    - `+` leading plus-sign for positive values
    - `0` leading zeros
  - field **width**
    - (none) minimum number of characters needed
    - integer width of field to be filled with output
  - **length** modifier
    - (none) `int` type
    - `h` `short int` type
    - `l` `long int` type
    - `ll` `long long int` type
  - **conversion** specifier
    - `d` signed decimal value
    - `u` unsigned decimal value
    - `o` (unsigned) octal value
    - `x` (unsigned) hexadecimal value using characters `0-9, a-f`
    - `X` (unsigned) hexadecimal value using characters `0-9, A-F`

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

4

## Formatted Output

- Detailed formatting sequence for floating-point values
  - *% flags width precision length conversion*
  - **flags**
    - (none) standard formatting (right-justified)
    - - left-justified output
    - + leading plus-sign for positive values
    - 0 leading zeros
  - field **width**
    - (none) minimum number of characters needed
    - integer width of field to be filled with output
  - **precision**
    - (none) default precision (e.g. 6)
    - .int number of digits after decimal point (for **f**, **e**, or **E**), maximum number of significant digits (for **g**, or **G**)
  - **length** modifier
    - (none) **float** or **double** type
    - **L** long **double** type
  - **conversion** specifier
    - **f** standard floating-point notation (fixed-point)
    - **e** or **E** exponential notation using (**e** or **E**)
    - **g** or **G** standard or exponential notation (using **e** or **E**)

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

5

## Formatted Output

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

```

/* Formatting.c: formatted output demo          */
/* author: Rainer Doemer                       */
/* modifications:                             */
/* 10/19/04 RD initial version                 */

#include <stdio.h>

/* main function */

int main(void)
{
    /* output section */
    printf("42 formatted as %d: |%d|\n", 42);
    printf("42 formatted as %8d: |%8d|\n", 42);
    printf("42 formatted as %-8d: |%-8d|\n", 42);
    printf("42 formatted as %+8d: |%+8d|\n", 42);
    printf("42 formatted as %08d: |%08d|\n", 42);
    printf("42 formatted as %x: |%x|\n", 42);
    printf("42 formatted as %o: |%o|\n", 42);
    ...
}

```

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

6

## Formatted Output

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

```

...
printf("\n");
printf("123.456 formatted as |%f|:      |%f|\n", 123.456);
printf("123.456 formatted as |%e|:      |%e|\n", 123.456);
printf("123.456 formatted as |%g|:      |%g|\n", 123.456);
printf("123.456 formatted as |%12.4f|: |%12.4f|\n",
      123.456);
printf("123.456 formatted as |%12.4e|: |%12.4e|\n",
      123.456);
printf("123.456 formatted as |%12.4g|: |%12.4g|\n",
      123.456);

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

/* EOF */

```

## Formatted Output

- Example session: `Formatting.c`

```

% vi Formatting.c
% gcc Formatting.c -o Formatting -Wall -ansi
% Formatting
42 formatted as |%d|: |42|
42 formatted as |%8d|: |      42|
42 formatted as |%-8d|: |42      |
42 formatted as |%+8d|: |      +42|
42 formatted as |%08d|: |00000042|
42 formatted as |%x|: |2a|
42 formatted as |%o|: |52|

123.456 formatted as |%f|: |123.456000|
123.456 formatted as |%e|: |1.234560e+02|
123.456 formatted as |%g|: |123.456|
123.456 formatted as |%12.4f|: | 123.4560|
123.456 formatted as |%12.4e|: | 1.2346e+02|
123.456 formatted as |%12.4g|: | 123.5|
%

```

## Programming Principles

- Thorough *understanding* of the problem
- *Problem definition*
  - Input data
  - Output data
- *Algorithm*: Procedure to solve the problem
  - Detailed set of *actions* to perform
  - Specification of *order* in which to perform the actions
  - Termination after a *finite* number of steps
- *Pseudo code*: Planning a program
  - Informal (English) description of steps in an algorithm
  - Example: Cake baking recipe
- *Control flow*
  - Execution order of statements in the program
- *Program*: Instructions for the computer
  - Formal description in programming language
    - Statements (steps, actions)
    - Control structures (flow of control)

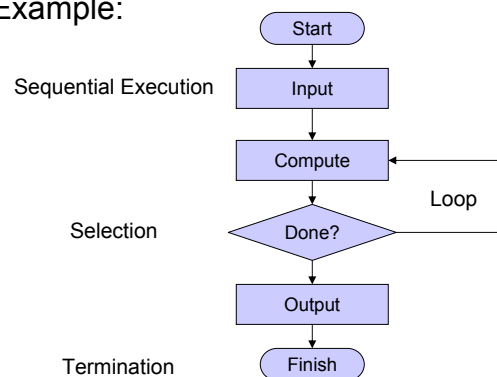
EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

9

## Control Flow

- Control flow charts
  - Graphical representation of program control flow
  - Example:



EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

10

## Structured Programming

- Sequential execution in C
  - Statement blocks: *Compound statements*
  - Sequence of statements grouped by braces: { }
- Example:

```

{
  /* statement 1 */

  /* statement 2 */

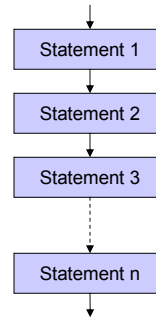
  /* statement 3 */

  /* ... */

  /* statement n */
}

```

Flow chart:



EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

11

## Structured Programming

- Sequential execution in C
  - Statement blocks: *Compound statements*
  - Sequence of statements grouped by braces: { }
- *Indentation* increases readability of the code
  - proper indentation is highly recommended!
- Example:

```

/* some statements... */
if (x < 0) {
  printf("%d is negative!", x);
  /* handle negative values of x... */
  if (x < 100) {
    printf("%d is too small!", x);
    /* handle the problem... */
  } /* fi */
} /* fi */
if (x > 0) {
  printf("%d is positive!", x);
  /* handle positive values of x... */
} /* fi */
/* more statements... */

```

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

12

## Structured Programming

- Sequential execution in C
  - Statement blocks: *Compound statements*
  - Sequence of statements grouped by braces: { }
- *Indentation* increases readability of the code
  - proper indentation is highly recommended!

• **Example:**

```

/* some statements... */
indentation level 0 if (x < 0) {
                    printf("%d is negative!", x);
indentation level 1 → /* handle negative values of x... */
                    if (x < 100) {
indentation level 2 → → printf("%d is too small!", x);
                        /* handle the problem... */
                        } /* fi */
indentation level 1 → } /* fi */
indentation level 0 if (x > 0) {
                    printf("%d is positive!", x);
indentation level 1 → /* handle positive values of x... */
                    } /* fi */
indentation level 0 /* more statements... */
  
```

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

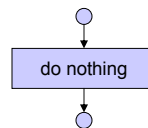
13

## Structured Programming

- Empty statement blocks
    - empty compound statement
    - does nothing (no operation, no-op)
    - Example:
- Flow chart:

```

{
  /* nothing */
}
  
```



EECS10: Computational Methods in ECE, Review 9-17

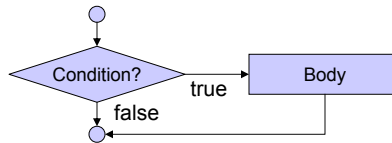
(c) 2009 R. Doemer

14

## Structured Programming

- Selection: **if** statement

– Flow chart:



– Example:

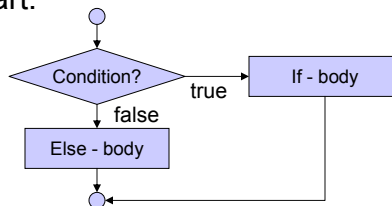
```

if (grade >= 60)
{ printf("You passed.");
} /* fi */
  
```

## Structured Programming

- Selection: **if-else** statement

– Flow chart:



– Example:

```

if (grade >= 60)
{ printf("You passed.");
} /* fi */
else
{ printf("You failed.");
} /* esle */
  
```



## Structured Programming

- Selection: **switch** statement
  - Flow chart:
  - Example:
 

```
switch (LetterGrade)
{ case 'A':
  { printf("Excellent!");
    break; }
  case 'B':
  case 'C':
  case 'D':
  { printf("Passed.");
    break; }
  case 'F':
  { printf("Failed!");
    break; }
  default:
  { printf("Invalid grade!");
    break; }
} /* hctiws */
```

EECS10: Computational Methods in ECE, Review 9-17 (c) 2009 R. Doemer 17

## Structured Program Composition

- Initial flow chart
  - Start
  - Program body
  - Finish
- Statement sequences
  - Statement blocks can be concatenated
  - Sequential execution
- Nested control structures
  - control structures can be placed wherever statement blocks can be placed in the code

EECS10: Computational Methods in ECE, Review 9-17 (c) 2009 R. Doemer 18

## Structured Program Composition

- Example:
  - Initial flow chart

```

    graph TD
      Start([Start]) --> Process[Process]
      Process --> End([End])
    
```

EECS10: Computational Methods in ECE, Review 9-17 (c) 2009 R. Doemer 19

## Structured Program Composition

- Example:
  - Sequential composition

```

    graph TD
      Start([Start]) --> P1[Process 1]
      P1 --> P2[Process 2]
      P2 --> End([End])
      subgraph SequentialComposition [ ]
        P1
        P2
      end
    
```

EECS10: Computational Methods in ECE, Review 9-17 (c) 2009 R. Doemer 20

## Structured Program Composition

- Example:
  - insertion of another sequential statement

The flowchart illustrates a sequential process. It starts with an oval terminal node at the top. An arrow points down to a rectangular process node. This process node is enclosed in a dashed rectangular box. Below it is another rectangular process node, also enclosed in the dashed box. An arrow points down from the second node in the dashed box to a third rectangular process node outside the box. Finally, an arrow points down to an oval terminal node at the bottom.

EECS10: Computational Methods in ECE, Review 9-17 (c) 2009 R. Doemer 21

## Structured Program Composition

- Example:
  - insertion of **if-else** statement

The flowchart illustrates a program structure with an if-else statement. It starts with an oval terminal node at the top. An arrow points down to a rectangular process node. Below this is a diamond-shaped decision node, which is enclosed in a dashed rectangular box. To the right of the diamond is a rectangular process node, also enclosed in the dashed box. An arrow points from the right side of the diamond to this process node. Below the diamond is another rectangular process node, also enclosed in the dashed box. An arrow points from the bottom of the diamond to this process node. An arrow points down from the process node below the diamond to a fourth rectangular process node outside the dashed box. Finally, an arrow points down to an oval terminal node at the bottom.

EECS10: Computational Methods in ECE, Review 9-17 (c) 2009 R. Doemer 22

## Structured Program Composition

- Example:
  - insertion of sequential statement

The flowchart shows a sequence of operations starting from a start node (oval). It proceeds through a process node (rectangle), a decision node (diamond), another process node, and a third process node before reaching an end node (oval). A dashed box highlights a modification: a new process node is added to the right of the first process node, and a second process node is added below it. Arrows indicate that the flow from the first process node now branches to the new top node, which then flows to the new bottom node, and finally back to the original flow path between the first and second process nodes.

EECS10: Computational Methods in ECE, Review 9-17 (c) 2009 R. Doemer 23

## Structured Program Composition

- Example:
  - insertion of **if-else** statement

The flowchart shows a sequence of operations starting from a start node (oval). It proceeds through a process node, a decision node, another process node, and a third process node before reaching an end node (oval). A dashed box highlights the insertion of an if-else statement: a new decision node is added to the right of the first process node. This new decision node has two outgoing paths: one to a process node above it and another to a process node below it. Arrows show the flow branching from the first process node to the new decision node, then to either the top or bottom process node, and finally merging back into the original flow path between the first and second process nodes.

EECS10: Computational Methods in ECE, Review 9-17 (c) 2009 R. Doemer 24

## Structured Program Composition

- Example:
  - insertion of sequential statement

The flowchart shows a sequence of operations: a start node, a process box, a decision diamond, a process box, a decision diamond, a process box, a loop body (two process boxes), a decision diamond, a process box, and an end node. A dashed box highlights the two process boxes in the loop body, indicating the insertion point for a new sequential statement.

EECS10: Computational Methods in ECE, Review 9-17 (c) 2009 R. Doemer 25

## Structured Program Composition

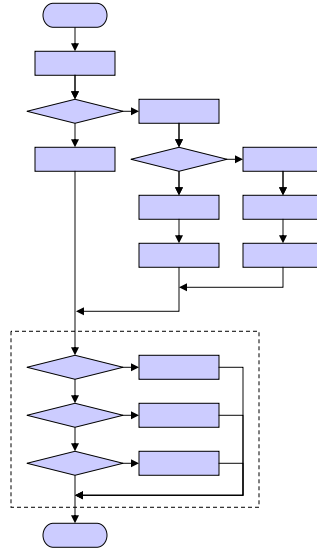
- Example:
  - insertion of sequential statement (twice)

The flowchart is identical to the one on slide 25, but the dashed box highlights the two process boxes in the loop body, indicating the insertion point for two sequential statements.

EECS10: Computational Methods in ECE, Review 9-17 (c) 2009 R. Doemer 26

## Structured Program Composition

- Example:
  - insertion of **switch** statement
  - etc. ...



EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

27

## Example Program

- Grade calculation: **Grade.c** (part 1/3)

```

/* Grade.c: convert score into letter grade */
/* author: Rainer Doemer */
/* modifications: */
/* 10/17/04 RD initial version */

#include <stdio.h>

/* main function */
int main(void)
{
    /* variable definitions */
    int score = 0;
    char grade;

    /* input section */
    while (score < 1 || score > 100)
    { printf("Please enter your score (1-100): ");
      scanf("%d", &score);
    } /* elihw */

    ...
  
```

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

28

## Example Program

- Grade calculation: `Grade.c` (part 2/3)

```
...
/* computation section */
if (score >= 90)
    { grade = 'A'; }
else
    { if (score >= 80)
      { grade = 'B'; }
      else
        { if (score >= 70)
          { grade = 'C'; }
          else
            { if (score >= 60)
              { grade = 'D'; }
              else
                { grade = 'F'; }
            } /* esle */
          } /* esle */
        } /* esle */
...

```

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

29

## Example Program

- Grade calculation: `Grade.c` (part 3/3)

```
...
/* output section */
printf("Your letter grade is %c.\n", grade);

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

/* EOF */

```

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

30

## Example Program

- Example session: `Grade.c`

```
% vi Grade.c
% gcc Grade.c -o Grade -Wall -ansi
% Grade
Please enter your score (1-100): 111
Please enter your score (1-100): 99
Your letter grade is A.
% Grade
Please enter your score (1-100): 85
Your letter grade is B.
% Grade
Please enter your score (1-100): 71
Your letter grade is C.
% Grade
Please enter your score (1-100): 69
Your letter grade is D.
% Grade
Please enter your score (1-100): 55
Your letter grade is F.
%
```

## Example Program

- Grade calculation: `Grade2.c` (part 1/3)

```
/* Grade2.c: convert score into letter grade */
/* author: Rainer Doemer */
/* modifications: */
/* 10/18/04 RD use 'switch' statement */
/* 10/17/04 RD initial version */

#include <stdio.h>

/* main function */

int main(void)
{
    /* variable definitions */
    int score = 0;
    char grade;

    /* input section */
    while (score < 1 || score > 100)
    { printf("Please enter your score (1-100): ");
      scanf("%d", &score);
    } /* elihw */
```



## Example Program

- Grade calculation: `Grade2.c` (part 2/3)

```
.../* computation section */
switch (score / 10)
{ case 10:
  case 9:
    { grade = 'A';
      break; }
  case 8:
    { grade = 'B';
      break; }
  case 7:
    { grade = 'C';
      break; }
  case 6:
    { grade = 'D';
      break; }
  default:
    { grade = 'F';
      break; }
} /* hctiws */
```

EECS ...

## Example Program

- Grade calculation: `Grade2.c` (part 3/3)

```
...

/* output section */
printf("Your letter grade is %c.\n", grade);

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

/* EOF */
```

## Example Program

- Example session: `Grade2.c`

```
% cp Grade.c Grade2.c
% vi Grade2.c
% gcc Grade2.c -o Grade2 -Wall -ansi
% Grade2
Please enter your score (1-100): 111
Please enter your score (1-100): 99
Your letter grade is A.
% Grade2
Please enter your score (1-100): 85
Your letter grade is B.
% Grade2
Please enter your score (1-100): 71
Your letter grade is C.
% Grade2
Please enter your score (1-100): 69
Your letter grade is D.
% Grade2
Please enter your score (1-100): 55
Your letter grade is F.
%
```

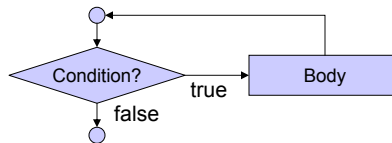
## Programming == Thinking

- Programming ...
  - ... is *not* a mechanic procedure!
  - ... requires *thinking*!
- Program ...
  - ... *writing* requires an *intelligent human being*!
  - ... *execution* can be done by a *dumb machine*.
- General programming steps:
  1. Understand the problem
  2. Define the input and output data
  3. Develop the algorithm (e.g. use pseudo code)
  4. Define the control flow (e.g. use control flow charts)
  5. Write the program in programming language
  6. Test and debug the program

## Structured Programming

- Repetition: **while** loop

- Flow chart:



- Example:

```

int product = 2;
while (product < 1000)
{ product *= 2;
  } /* elihw */
  
```

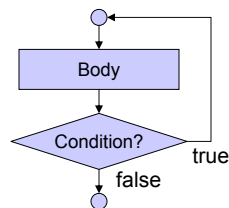
- Note:

- The condition is evaluated at the *beginning* of each loop!

## Structured Programming

- Repetition: **do-while** loop

- Flow chart:



- Example:

```

int product = 2;
do { product *= 2;
  } while (product < 1000);
  
```

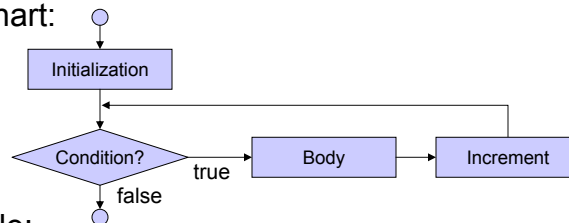
- Note:

- The condition is evaluated at the *end* of each loop!

## Structured Programming

- Repetition: **for** loop

- Flow chart:



- Example:

```

for(i = 0; i < 10; i++)
{ printf("i = %d\n", i);
} /* rof */
  
```

- Syntax:

- **for**(*initialization*; *condition*; *increment*)  
*body*

## Example Program

- Compound interest: **Interest.c**

- Assignment:

- Write a program that calculates the interest accumulated in a savings account. Given an initial deposit amount and an annual percentage rate (APR), compute the yearly interest earned and the resulting balance, for a period of ten years.

- The output should be listed in a table as follows:

```

Interest for year 1 is $ 45.00, total balance is $ 1045.00.
Interest for year 2 is $ 47.02, total balance is $ 1092.03.
Interest for year 3 is $ 49.14, total balance is $ 1141.17.
...
  
```

## Example Program

- Compound interest: **Interest.c**
- Assignment:
  - Write a program that calculates the interest accumulated in a savings account. Given an initial deposit amount and an annual percentage rate (APR), compute the yearly interest earned and the resulting balance, for a period of ten years.
- Step 1: Understand the problem
  - What is given?
    - deposit amount, annual percentage rate
  - What is asked for?
    - yearly interest, resulting balance
  - How do we compute what is asked for?
    - $interest = amount * APR/100$
    - $balance = amount + interest$

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

41

## Example Program

- Compound interest: **Interest.c**
- Assignment:
  - Write a program that calculates the interest accumulated in a savings account. Given an initial deposit amount and an annual percentage rate (APR), compute the yearly interest earned and the resulting balance, for a period of ten years.
- Step 2: Define the input and output data
  - Input:
    - Initial deposit amount: floating point value, **amount**
    - Annual percentage rate: floating point value, **rate**
  - Output:
    - Current year: integral value, **year**
    - Interest earned: floating point value, **interest**
    - Resulting balance: floating point value, **balance**

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

42

## Example Program

- Compound interest: **Interest.c**
- Assignment:
  - Write a program that calculates the interest accumulated in a savings account. Given an initial deposit amount and an annual percentage rate (APR), compute the yearly interest earned and the resulting balance, for a period of ten years.
- Step 3: Develop the algorithm
  - First, input **amount** and **rate**
  - Next, compute **interest** on the **amount** for the year
  - Next, compute new **balance** at the end of the year
  - Then, print **year**, **interest** and **balance** in tabular format
  - Finally, set the **amount** to the new **balance**
  - Repeat the previous 4 steps for 10 years
  - Done!

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

43

## Example Program

- Compound interest: **Interest.c**
- Assignment:
  - Write a program that calculates the interest accumulated in a savings account. Given an initial deposit amount and an annual percentage rate (APR), compute the yearly interest earned and the resulting balance, for a period of ten years.
- Step 4: Define the control flow
  - First, input **amount** and **rate**
  - Repeat for 10 years:
    - Compute **interest** on the **amount** for the year
    - Compute new **balance** at the end of the year
    - Print **year**, **interest** and **balance** in tabular format
    - Set the **amount** to the new **balance**
  - Done!

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

44

## Example Program

- Compound interest: `Interest.c`
- Assignment:
  - Write a program that calculates the interest accumulated in a savings account. Given an initial deposit amount and an annual percentage rate (APR), compute the yearly interest earned and the resulting balance, for a period of ten years.
- Step 5: Write the program in programming language

```
double amount;      double rate;      int year;
double interest;    double balance;

printf("Please enter the initial amount in $: ");
scanf("%lf", &amount);

printf("Please enter the interest rate in %% : ");
scanf("%lf", &rate);
```

etc.

## Example Program

- Compound interest: `Interest.c` (part 1/2)

```
/* Interest.c: compound interest on savings account */
/* author: Rainer Doemer */
/* modifications: */
/* 10/18/06 RD distinguish amount and balance */
/* 10/19/04 RD initial version */

#include <stdio.h>

/* main function */
int main(void)
{
    /* variable definitions */
    double amount, balance, rate, interest;
    int year;

    /* input section */
    printf("Please enter the initial amount in $: ");
    scanf("%lf", &amount);
    printf("Please enter the interest rate in %% : ");
    scanf("%lf", &rate);
    ...
}
```

## Example Program

- Compound interest: `Interest.c` (part 2/2)

```
...  
  
/* computation and output section */  
for(year = 1; year <= 10; year++)  
{ interest = amount * (rate/100.0);  
  balance = amount + interest;  
  printf("Interest for year %2d is $%8.2f,"  
        " total balance is $%8.2f.\n",  
        year, interest, balance);  
  amount = balance;  
} /* rof */  
  
/* exit */  
return 0;  
} /* end of main */  
  
/* EOF */
```

## Example Program

- Compound interest: `Interest.c`
- Assignment:
  - Write a program that calculates the interest accumulated in a savings account. Given an initial deposit amount and an annual percentage rate (APR), compute the yearly interest earned and the resulting balance, for a period of ten years.
- Step 6: Test (and debug) the program
  - see next slide!



## Example Program

- Example session: `Interest.c`

```
% vi Interest.c
% gcc Interest.c -o Interest -Wall -ansi
% Interest
Please enter the initial amount in $: 1500
Please enter the interest rate in % : 1.5
Interest for year 1 is $ 22.50, total balance is $ 1522.50.
Interest for year 2 is $ 22.84, total balance is $ 1545.34.
Interest for year 3 is $ 23.18, total balance is $ 1568.52.
Interest for year 4 is $ 23.53, total balance is $ 1592.05.
Interest for year 5 is $ 23.88, total balance is $ 1615.93.
Interest for year 6 is $ 24.24, total balance is $ 1640.16.
Interest for year 7 is $ 24.60, total balance is $ 1664.77.
Interest for year 8 is $ 24.97, total balance is $ 1689.74.
Interest for year 9 is $ 25.35, total balance is $ 1715.08.
Interest for year 10 is $ 25.73, total balance is $ 1740.81.
%
```

EECS10: Computational Methods in ECE, Review 9-17

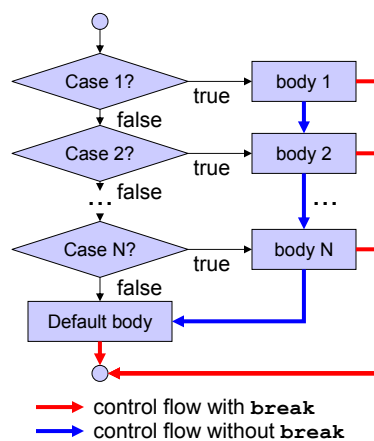
(c) 2009 R. Doemer

49

## Structured Programming

- Selection: `break` in `switch` statement

– Flow chart:



EECS10: Computational Methods in ECE, Review 9-17

Example:

```
switch(LetterGrade)
{ case 'A':
  { printf("Excellent!");
    break; }
  case 'B':
  case 'C':
  case 'D':
  { printf("Passed.");
    break; }
  case 'F':
  { printf("Failed!");
    break; }
  default:
  { printf("Invalid grade!");
    break; }
} /* hctiws */
```

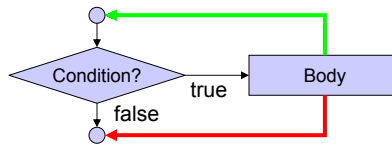
(c) 2009 R. Doemer

50

## Structured Programming

- Repetition: **break/continue** in **while** loop

– Flow chart:



– Control flow:

- control flow with **break**
- control flow with **continue**

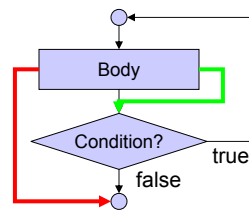
– Note:

- The condition is evaluated at the *beginning* of each loop!

## Structured Programming

- Repetition: **break/continue** in **do-while** loop

– Flow chart:



– Control flow:

- control flow with **break**
- control flow with **continue**

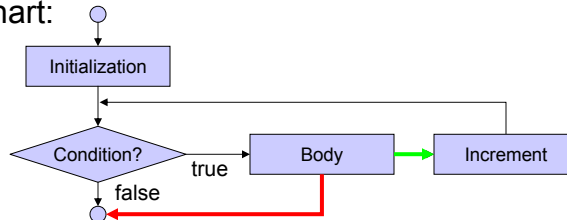
– Note:

- The condition is evaluated at the *end* of each loop!

## Structured Programming

- Repetition: **break/continue** in **for** loop

– Flow chart:



– Control flow:

- control flow with **break**
- control flow with **continue**

– Syntax:

- `for(initialization; condition; increment)`  
`body`

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

53

## Arbitrary Jump Statements

- Arbitrary jumps: **goto** statement

– **goto** statement jumps to the specified *labeled* statement (within the same function)

– Example:

```

begin: count = 0;
      goto next;
repeat: if (count > 100)
        { goto end; }
next:   count++;
        if (count == 77)
        { goto next; }
        goto repeat;
end:    printf("%d", count);
  
```

– Warning:

- **goto** statement allows *un-structured* programming!
- **goto** statement should be avoided whenever possible!

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

54

## Debugging

- Source-level Debugger **gdb**
  - Debugging features
    - run the program under debugger control
    - follow the control flow of the program during execution
    - set breakpoints to stop execution at specific points
    - inspect (and adjust) the values of variables
    - find the point in the program where the “crash” happens
  - Preparation:
    - compile your program with debugging support on
    - Option **-g** tells compiler to add debugging information (symbol tables) to the generated executable file
    - **gcc -g Program.c -o Program -Wall -ansi**
    - **gdb Program**

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

55

## Debugging

- Source-level Debugger **gdb**
  - Basic **gdb** commands
    - **run**
      - starts the execution of the program in the debugger
    - **break *function\_name***
      - inserts a breakpoint at *function\_name*
      - program execution will stop at the breakpoint
    - **list *line\_numbers***
      - lists the current or specified *line\_numbers*
    - **print *variable\_name***
      - prints the current value of the variable *variable\_name*
    - **next**
      - executes the next statement (one statement at a time)
    - **quit**
      - exits the debugger (and terminates the program)
    - **help**
      - provides helpful details on debugger commands

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

56

## Example Program

- Compound interest: `Interest2.c` (part 1/2)

```

/* Interest2.c: compound interest on savings account */
/* author: Rainer Doemer */
/* modifications: */
/* 10/23/05 RD version to demonstrate debugging */
/* 10/19/04 RD initial version */

#include <stdio.h>

/* main function */

int main(void)
{
    /* variable definitions */
    double amount, balance, rate, interest;
    int year;

    /* input section */
    printf("Please enter the initial amount in $:\n");
    scanf("%lf", &amount);
    printf("Please enter the interest rate in %:\n");
    scanf("%lf", &rate);

```

EECS ...

## Example Program

- Compound interest: `Interest2.c` (part 2/2)

```

...

/* computation and output section */
for(year = 1; year <= 10; year++)
{ interest = amount * (rate/100.0);
  balance = amount + interest;
  printf("Interest for year%3d is $%8.2f.\n", year,
        interest);
  printf("The new balance is $%8.2f.\n", balance);
  amount = balance;
} /* rof */

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

/* EOF */

```

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

58

## Example Program

- Example session: `Interest2.c` (part 1/6)

```
% vi Interest2.c
% gcc Interest2.c -o Interest2 -g -Wall -ansi
% Interest2
Please enter the initial amount in $:
1000
Please enter the interest rate in %:
1.5
Interest for year 1 is $ 15.00.
The new balance is $ 1015.00.
Interest for year 2 is $ 15.22.
The new balance is $ 1030.22.
...
Interest for year 10 is $ 17.15.
The new balance is $ 1160.54.
% gdb Interest2
GNU gdb 6.3
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, ...
There is absolutely no warranty for GDB.
This GDB was configured as "sparc-sun-solaris2.7"...
(gdb)
...
```

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

59

## Example Program

- Example session: `Interest2.c` (part 2/6)

```
...
(gdb) break main
Breakpoint 1 at 0x106ac: file Interest2.c, line 20.
(gdb) run
Starting program: /users/faculty/doemer/eecs10/Interest/Interest2
Breakpoint 1, main () at Interest2.c:20
20     printf("Please enter the initial amount in $:\n");
(gdb) next
Please enter the initial amount in $:
21     scanf("%lf", &amount);
(gdb) next
1000
22     printf("Please enter the interest rate in %:\n");
(gdb) next
Please enter the interest rate in %:
23     scanf("%lf", &rate);
(gdb) next
1.5
26     for(year = 1; year <= 10; year++)
(gdb) next
...
```

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

60

## Example Program

- Example session: `Interest2.c` (part 3/6)

```

...
27 { interest = amount * (rate/100.0);
(gdb) next
28     balance = amount + interest;
(gdb) print interest
$1 = 15
(gdb) print amount
$2 = 1000
(gdb) print balance
$3 = -7.3987334479772013e+304
(gdb) next
29     printf("Interest for year%d is $%.2f.\n", year, interest);
(gdb) print balance
$4 = 1015
(gdb) next
Interest for year 1 is $ 15.00.
30     printf("The new balance is      $%.2f.\n", balance);
(gdb) next
The new balance is      $ 1015.00.
31     amount = balance;
(gdb) next
...

```

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

61

## Example Program

- Example session: `Interest2.c` (part 4/6)

```

...
26 for(year = 1; year <= 10; year++)
(gdb) next
27 { interest = amount * (rate/100.0);
(gdb) print year
$5 = 2
(gdb) list
22 printf("Please enter the interest rate in %:\n");
23 scanf("%lf", &rate);
24
25 /* computation and output section */
26 for(year = 1; year <= 10; year++)
27 { interest = amount * (rate/100.0);
28     balance = amount + interest;
29     printf("Interest for year%d is $%.2f.\n", year, interest);
30     printf("The new balance is      $%.2f.\n", balance);
31     amount = balance;
(gdb) list 35
30     printf("The new balance is      $%.2f.\n", balance);
31     amount = balance;
32 } /* eof */
...

```

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

62

## Example Program

- Example session: `Interest2.c` (part 5/6)

```

...
33
34 /* exit */
35 return 0;
36 } /* end of main */
37
38 /* EOF */
(gdb) break 35
Breakpoint 2 at 0x1079c: file Interest2.c, line 35.
(gdb) cont
Continuing.
Interest for year 2 is $ 15.22.
The new balance is $ 1030.22.
Interest for year 3 is $ 15.45.
The new balance is $ 1045.68.
...
Interest for year 10 is $ 17.15.
The new balance is $ 1160.54.

Breakpoint 2, main () at Interest2.c:35
35 return 0;
...

```

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

63

## Example Program

- Example session: `Interest2.c` (part 6/6)

```

...
(gdb) print balance
$6 = 1160.5408250251503
(gdb) cont
Continuing.

Program exited normally.
(gdb) quit
%

```

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

64



## Functions

- Introduction to Functions
  - Important programming concepts
    - Hierarchy
    - Encapsulation
    - Information hiding
    - Divide and conquer
  - Software reuse
    - Don't re-invent the wheel!
  - Program composition
    - C program = Set of functions
      - starting point: function named `main`
    - Libraries = Set of functions
      - predefined functions (often written by somebody else)

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

65

## Functions

- C programming language distinguishes 3 constructs around functions
  - Function declaration
    - declaration of function name, parameters, and return type
  - Function definition
    - extension of a function declaration with a function body
    - definition of the function behavior
  - Function call
    - invocation of a function

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

66

## Functions

- Function declaration
  - aka. function prototype or function signature
  - declares
    - function name
    - function parameters
    - type of return value
- Example:

```
double Square(double p);
```

  - function is named **Square**
  - function takes one parameter **p** of type **double**
  - function returns a value of type **double**

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

67

## Functions

- Function definition
  - extends a function declaration with a function body
  - defines the statements executed by the function
  - may use local variables for the computation
  - returns result value via **return** statement (if any)
- Example:

```
double Square(double p)
{
    double r;
    r = p * p;
    return r;
}
```

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

68

## Functions

- Function call
  - expression invoking a function
  - supplies arguments for formal parameters
  - invokes the function
  - result is the value returned by the function
- Example:
 

```
double a, b;
b = Square(a);
```

  - function **Square** is called
  - argument **a** is passed for parameter **p** (by value)
  - value returned by the function is assigned to **b**

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

69

## Functions

- C programming language distinguishes 3 constructs
  - Function declaration
    - declaration of function name, parameters, and return type
  - Function definition
    - extension of a function declaration with a function body
    - definition of the function behavior
  - Function call
    - invocation of a function
- C program rules
  - A function must be declared before it can be called.
  - Multiple function declarations are allowed (if they match).
  - A function definition is an implicit function declaration.
  - A function must be defined exactly once in a program.
  - A function may be called any number of times.

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

70

## Functions

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

```

/* Square.c: example demonstrating functions */
/* author: Rainer Doemer */
/* modifications: */
/* 10/27/08 RD renamed parameters and arguments */
/* 10/27/04 RD initial version */

#include <stdio.h>

/* function declaration */
double square(double p);

/* function definition */
double square(double p)
{
    double r;
    r = p * p;
    return r;
} /* end of square */

...

```

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

71

## Functions

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

```

...
/* main function */
int main(void)
{
    /* variable definitions */
    double a, b;

    /* input section */
    printf("Please enter a value for the argument: ");
    scanf("%lf", &a);

    /* computation section */
    b = square(a);

    /* output section */
    printf("The square of %g is %g.\n", a, b);

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

/* EOF */

```

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

72

## Functions

- Example session: `Square.c`

```
% vi Square.c
% gcc Square.c -o Square -Wall -ansi
% Square
Please enter a value for the argument: 3
The square of 3 is 9.
% Square
Please enter a value for the argument: 5.5
The square of 5.5 is 30.25.
%
```

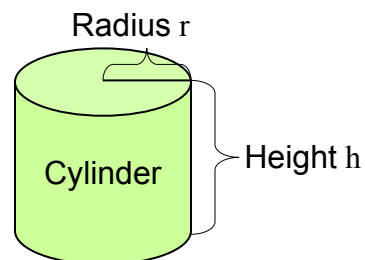
## Functions

- Hierarchy of Functions
  - functions call other functions

- Example:

### Cylinder calculations

- given radius and height
- calculate surface and volume



- Circle constant  $\pi = 3.14159265\dots$
- Circle perimeter  $f_p(r) = 2 \times \pi \times r$
- Circle area  $f_a(r) = \pi \times r^2$
- Cylinder surface  $f_s(r, h) = f_p(r) \times h + 2 \times f_a(r)$
- Cylinder volume  $f_v(r, h) = f_a(r) \times h$

## Functions

- Program example: `Cylinder.c` (part 1/3)

```

/* Cylinder.c: cylinder functions */
/* author: Rainer Doemer */
/* modifications: */
/* 10/25/05 RD initial version */

#include <stdio.h>

/* cylinder functions */

double pi(void)
{
    return(3.1415927);
}

double CircleArea(double r)
{
    return(pi() * r * r);
}
...

```

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

75

## Functions

- Program example: `Cylinder.c` (part 2/3)

```

...
double CirclePerimeter(double r)
{
    return(2 * pi() * r);
}

double Surface(double r, double h)
{
    double side, lid;

    side = CirclePerimeter(r) * h;
    lid = CircleArea(r);

    return(side + 2*lid);
}

double Volume(double r, double h)
{
    return(CircleArea(r) * h);
}
...

```

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

76

## Functions

- Program example: `Cylinder.c` (part 3/3)

```

...
/* main function */
int main(void)
{   double r, h, s, v;

    /* input section */
    printf("Please enter the radius: ");
    scanf("%lf", &r);
    printf("Please enter the height: ");
    scanf("%lf", &h);

    /* computation section */
    s = Surface(r, h);
    v = Volume(r, h);

    /* output section */
    printf("The surface area is %f.\n", s);
    printf("The volume is %f.\n", v);

    return 0;
} /* end of main */

```

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

77

## Functions

- Example session: `Cylinder.c`

```

% vi Cylinder.c
% gcc Cylinder.c -o Cylinder -Wall -ansi
% Cylinder
Please enter the radius: 5.0
Please enter the height: 8.0
The surface area is 408.407051.
The volume is 628.318540.
%

```

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

78

## Functions

- Review: Terms and Concepts
  - Function declaration
    - function prototype with name, parameters, and return type
  - Function definition
    - extended declaration, defines the behavior in function body
  - Function call
    - expression invoking a function with supplied arguments
  - Function arguments
    - arguments passed to a function call (initial values for parameters)
  - Function parameters
    - formal parameters holding the data supplied to a function
  - Local variables
    - variables defined locally in a function body
  - Return value
    - result computed by a function call

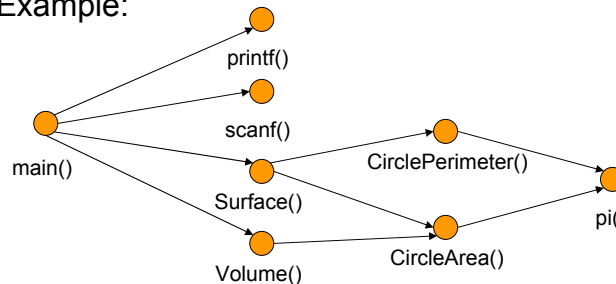
EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

79

## Function Call Graph

- Graphical representation of function calls
  - Directed Graph
    - Nodes: Functions
    - Edges: Function calls
  - Shows dependencies among functions
  - Example:



EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

80



## Function Call Trace

- Sequence of function calls
  - Shows execution order of functions at run-time
- Example:
  - main()
    - printf()
    - scanf()
    - printf()
    - scanf()
    - Surface()
      - CirclePerimeter()
        - » pi()
      - CircleArea()
        - » pi()
    - Volume()
      - CircleArea()
        - » pi()
  - printf()
  - printf()

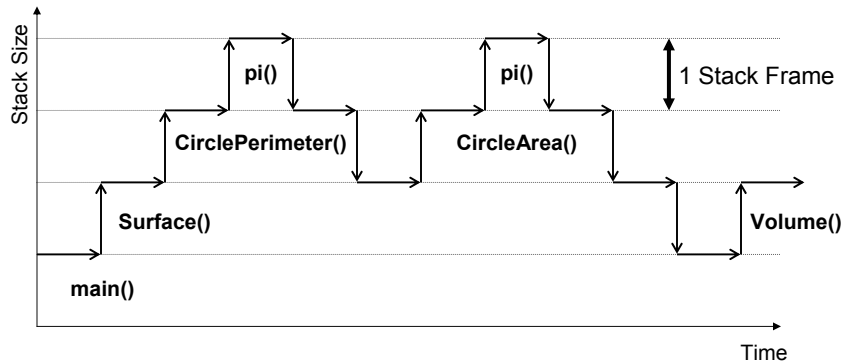
EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

81

## Function Call Stack

- Stack Frames
  - Keep track of active function calls
    - Stack grows by one frame with each function call
    - Stack shrinks by one frame with each completed function



EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

82

## Function Call Stack

- Stack Frames
  - Keep track of active function calls
    - Stack grows by one frame with each function call
    - Stack shrinks by one frame with each completed function

Time

EECS10: Computational Methods in ECE, Review 9-17 (c) 2009 R. Doemer 83

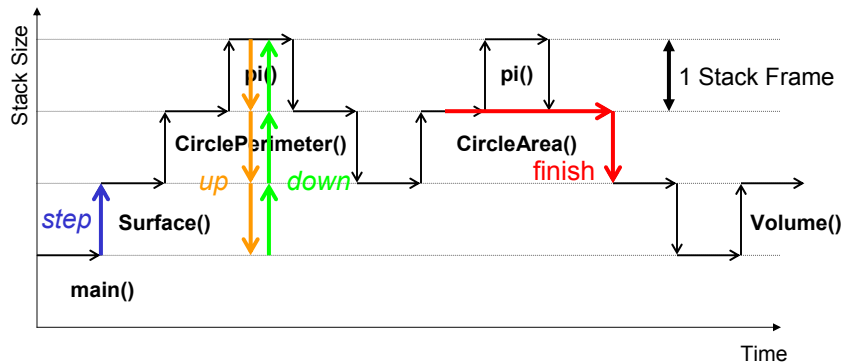
## Debugging

- Source-level Debugger **gdb** (continued)
  - Additional **gdb** commands
    - **step**
      - steps into a function call
    - **finish**
      - continues execution until the current function is finished
    - **where**
      - shows where in the function call hierarchy you are
      - prints a *back trace* of current *stack frames*
    - **up**
      - steps up one stack frame (up into the caller)
    - **down**
      - steps down one stack frame (down into the callee)

EECS10: Computational Methods in ECE, Review 9-17 (c) 2009 R. Doemer 84

## Function Call Stack

- Navigating Stack Frames in the Debugger
  - *step*: execute and step into a function call
  - *up*, *down*: navigate stack frames
  - *finish*: resume execution until the end of the current function



EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

85

## Functions

- Example session: `Cylinder.c`

```

% vi Cylinder.c
% gcc Cylinder.c -o Cylinder -Wall -ansi -g
% gdb Cylinder
GNU gdb 6.3
(gdb) break 55
Breakpoint 1 at 0x108d0: file Cylinder.c, line 55.
(gdb) run
Starting program: /users/faculty/doemer/eecs10/Cylinder/Cylinder
Please enter the radius: 10
Please enter the height: 10
Breakpoint 1, main () at Cylinder.c:56
56      s = Surface(r, h);
(gdb) step
Surface (r=10, h=10) at Cylinder.c:31
31      side = CirclePerimeter(r) * h;
(gdb) step
CirclePerimeter (r=10) at Cylinder.c:24
24      return(2 * pi() * r);
...

```

EE

## Functions

- Example session: `Cylinder.c`

```
(gdb) step
pi () at Cylinder.c:14
14      return(3.1415927);
(gdb) where
#0  pi () at Cylinder.c:14
#1  0x000107bc in CirclePerimeter (r=10) at Cylinder.c:24
#2  0x000107f8 in Surface (r=10, h=10) at Cylinder.c:31
#3  0x000108e0 in main () at Cylinder.c:56
(gdb) up
#1  0x000107bc in CirclePerimeter (r=10) at Cylinder.c:24
24      return(2 * pi() * r);
(gdb) up
#2  0x000107f8 in Surface (r=10, h=10) at Cylinder.c:31
31      side = CirclePerimeter(r) * h;
(gdb) up
#3  0x000108e0 in main () at Cylinder.c:56
56      s = Surface(r, h);
...
```

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

87

## Functions

- Example session: `Cylinder.c`

```
(gdb) down
#2  0x000107f8 in Surface (r=10, h=10) at Cylinder.c:31
31      side = CirclePerimeter(r) * h;
(gdb) down
#1  0x000107bc in CirclePerimeter (r=10) at Cylinder.c:24
24      return(2 * pi() * r);
(gdb) down
#0  pi () at Cylinder.c:14
14      return(3.1415927);
(gdb) finish
Run till exit from #0  pi () at Cylinder.c:14
0x000107bc in CirclePerimeter (r=10) at Cylinder.c:24
24      return(2 * pi() * r);
Value returned is $1 = 3.1415926999999999
(gdb) finish
Run till exit from #0  CirclePerimeter (r=10) at Cylinder.c:24
0x000107f8 in Surface (r=10, h=10) at Cylinder.c:31
31      side = CirclePerimeter(r) * h;
...
```

EE

## Functions

- Example session: `Cylinder.c`

```
Value returned is $2 = 62.831854
(gdb) next
32     lid = CircleArea(r);
(gdb) step
CircleArea (r=10) at Cylinder.c:19
19     return(pi() * r * r);
(gdb) finish
Run till exit from #0  CircleArea (r=10) at Cylinder.c:19
0x00010818 in Surface (r=10, h=10) at Cylinder.c:32
32     lid = CircleArea(r);
Value returned is $3 = 314.15926999999999
(gdb) cont
Continuing.
The surface area is 1256.637080.
The volume is 3141.592700.
Program exited normally.
(gdb) quit
%
```

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

89

## Functions

- *Scope of an identifier*
  - Portion of the program where the identifier can be referenced
  - aka. accessibility, visibility
- *Scope rules*
  - Global variables: *file scope*
    - Declaration outside any function (at global level)
    - Scope in entire source file after declaration
  - Function parameters: *function scope*
    - Declaration in function parameter list
    - Scope limited to this function body (entirely)
  - Local variables: *block scope*
    - Declaration inside a compound statement (i.e. function body)
    - Scope limited to this compound statement block (entirely)

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

90

## Scope Rules: Example

<code>#include &lt;stdio.h&gt;</code>	Header file inclusion
<code>int square(int a);</code> <code>int add_y(int x);</code>	Function declarations
<code>int x = 5;</code> <code>    y = 7;</code>	Global variables
<code>int square(int a)</code> { <code>int s;</code> <code>s = a * a;</code> <code>return s;</code> }	Function definition Local variable
<code>int add_y(int x)</code> { <code>int s;</code> <code>s = x + y;</code> <code>return s;</code> }	Function definition Local variable
<code>int main(void)</code> { <code>int z;</code> <code>z = square(x);</code> <code>z = add_y(z);</code> <code>printf("%d\n", z);</code> <code>return 0;</code> }	Function definition Local variable

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

91

## Scope Rules: Example

<code>#include &lt;stdio.h&gt;</code>	
<code>int square(int a);</code> <code>int add_y(int x);</code>	
<code>int x = 5;</code> <code>    y = 7;</code>	
<code>int square(int a)</code> { <code>int s;</code> <code>s = a * a;</code> <code>return s;</code> }	} Scope of global functions <b>printf()</b> , <b>scanf()</b> , etc.
<code>int add_y(int x)</code> { <code>int s;</code> <code>s = x + y;</code> <code>return s;</code> }	
<code>int main(void)</code> { <code>int z;</code> <code>z = square(x);</code> <code>z = add_y(z);</code> <code>printf("%d\n", z);</code> <code>return 0;</code> }	

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

92

## Scope Rules: Example

```
#include <stdio.h>
int square(int a);
int add_y(int x);
int x = 5,
    y = 7;
int square(int a)
{ int s;
  s = a * a;
  return s;
}
int add_y(int x)
{ int s;
  s = x + y;
  return s;
}
int main(void)
{ int z;
  z = square(x);
  z = add_y(z);
  printf("%d\n", z);
  return 0;
}
```

Scope of global function  
**square ()**

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

93

## Scope Rules: Example

```
#include <stdio.h>
int square(int a);
int add_y(int x);
int x = 5,
    y = 7;
int square(int a)
{ int s;
  s = a * a;
  return s;
}
int add_y(int x)
{ int s;
  s = x + y;
  return s;
}
int main(void)
{ int z;
  z = square(x);
  z = add_y(z);
  printf("%d\n", z);
  return 0;
}
```

Scope of global function  
**add\_y ()**

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

94

## Scope Rules: Example

```
#include <stdio.h>
int square(int a);
int add_y(int x);
int x = 5;
    y = 7;
int square(int a)
{ int s;
  s = a * a;
  return s;
}
int add_y(int x)
{ int s;
  s = x + y;
  return s;
}
int main(void)
{ int z;
  z = square(x);
  z = add_y(z);
  printf("%d\n", z);
  return 0;
}
```

Scope of global variable  
**x**

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

95

## Scope Rules: Example

```
#include <stdio.h>
int square(int a);
int add_y(int x);
int x = 5;
    y = 7;
int square(int a)
{ int s;
  s = a * a;
  return s;
}
int add_y(int x)
{ int s;
  s = x + y;
  return s;
}
int main(void)
{ int z;
  z = square(x);
  z = add_y(z);
  printf("%d\n", z);
  return 0;
}
```

Scope of global variable  
**y**

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

96



## Scope Rules: Example

```
#include <stdio.h>
int square(int a);
int add_y(int x);
int x = 5,
    y = 7;
int square(int a)
{ int s;
  s = a * a;
  return s;
}
int add_y(int x)
{ int s;
  s = x + y;
  return s;
}
int main(void)
{ int z;
  z = square(x);
  z = add_y(z);
  printf("%d\n", z);
  return 0;
}
```

Scope of parameter  
a

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

97

## Scope Rules: Example

```
#include <stdio.h>
int square(int a);
int add_y(int x);
int x = 5,
    y = 7;
int square(int a)
{ int s;
  s = a * a;
  return s;
}
int add_y(int x)
{ int s;
  s = x + y;
  return s;
}
int main(void)
{ int z;
  z = square(x);
  z = add_y(z);
  printf("%d\n", z);
  return 0;
}
```

Scope of local variable  
s

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

98

## Scope Rules: Example

```
#include <stdio.h>
int square(int a);
int add_y(int x);
int x = 5,
    y = 7;
int square(int a)
{ int s;
  s = a * a;
  return s;
}
int add_y(int x)
{ int s;
  s = x + y;
  return s;
}
int main(void)
{ int z;
  z = square(x);
  z = add_y(z);
  printf("%d\n", z);
  return 0;
}
```

*Local variables  
are independent!*  
(unless their scopes are nested)

Scope of local variable

**s**

Scope of local variable

**s**

Scope of local variable

**z**

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

99

## Scope Rules: Example

```
#include <stdio.h>
int square(int a);
int add_y(int x);
int x = 5,
    y = 7;
int square(int a)
{ int s;
  s = a * a;
  return s;
}
int add_y(int x)
{ int s;
  s = x + y;
  return s;
}
int main(void)
{ int z;
  z = square(x);
  z = add_y(z);
  printf("%d\n", z);
  return 0;
}
```

*Local variables  
are independent!*  
(unless their scopes are nested)

Scope of local variable

**s**

Scope of local variable

**s**

Scope of local variable

**z**

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

100

## Scope Rules: Example

```
#include <stdio.h>
int square(int a);
int add_y(int x);

int x = 5,
    y = 7;

int square(int a)
{ int s;
  s = a * a;
  return s;
}

int add_y(int x)
{ int s;
  s = x + y;
  return s;
}

int main(void)
{ int z;
  z = square(x);
  z = add_y(z);
  printf("%d\n", z);
  return 0;
}
```

Scope of parameter  
**x**

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

101

## Scope Rules: Example

```
#include <stdio.h>
int square(int a);
int add_y(int x);

int x = 5,
    y = 7;

int square(int a)
{ int s;
  s = a * a;
  return s;
}

int add_y(int x)
{ int s;
  s = x + y;
  return s;
}

int main(void)
{ int z;
  z = square(x);
  z = add_y(z);
  printf("%d\n", z);
  return 0;
}
```

**Shadowing!**  
In nested scopes,  
inner scope takes precedence!

Scope of global variable  
**x**

Scope of parameter  
**x**

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

102

## Debugging

- Source-level Debugger `gdb` (continued)
  - Additional `gdb` commands
    - `step`
      - steps into a function call
    - `finish`
      - continues execution until the current function is finished
    - `where`
      - shows where in the function call hierarchy you are
      - prints a *back trace* of current *stack frames*
    - `up`
      - steps up one stack frame (up into the caller)
    - `down`
      - steps down one stack frame (down into the callee)
    - `info locals`
      - lists the local variables in the current function (current stack frame)
    - `info scope function_name`
      - lists the variables in scope of the *function\_name*

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

103

## Scope Rules: Example

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

```

/* Scope.c: example demonstrating scope rules */
/* author: Rainer Doemer */
/* modifications: */
/* 10/30/04 RD initial version */

#include <stdio.h>

int square(int a); /* global function declarations */
int add_y(int x);

int x = 5, /* global variables */
    y = 7;

int square(int a) /* global function definition */
{
    int s; /* local variable */

    s = a * a;
    return s;
}
...

```

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

104

## Scope Rules: Example

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

```

...
int add_y(int x)          /* global function definition */
{
    int s;                /* local variable */

    s = x + y;
    return s;
}

int main(void)           /* main function definition */
{
    int z;                /* local variable */

    z = square(x);
    z = add_y(z);

    printf("%d, %d, %d\n", x, y, z);
    return 0;
}
/* EOF */

```

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

105

## Scope Rules: Example

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

```

% vi Scope.c
% gcc Scope.c -o Scope -Wall -ansi -g
% Scope
5, 7, 32
% gdb Scope
GNU gdb 5.0
[...]
(gdb) break main
Breakpoint 1 at 0x1079c: file Scope.c, line 36.
(gdb) run
Starting program: /users/faculty/doemer/eecs10/Scope/Scope

Breakpoint 1, main () at Scope.c:36
36      z = square(x);
(gdb) step
square (a=5) at Scope.c:20
20      s = a * a;
(gdb) next
21      return s;
...

```

EE

## Scope Rules: Example

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

```

...
(gdb) next
22     }
(gdb) next
main () at Scope.c:37
37     z = add_y(z);
(gdb) step
add_y (x=25) at Scope.c:28
28     s = x + y;
(gdb) where
#0  add_y (x=25) at Scope.c:28
#1  0x107c4 in main () at Scope.c:37
(gdb) up
#1  0x107c4 in main () at Scope.c:37
37     z = add_y(z);
(gdb) down
#0  add_y (x=25) at Scope.c:28
28     s = x + y;
...

```

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

107

## Scope Rules: Example

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

```

...
(gdb) finish
Run till exit from #0  add_y (x=25) at Scope.c:28
0x107c4 in main () at Scope.c:37
37     z = add_y(z);
Value returned is $1 = 32
(gdb) info locals
z = 25
(gdb) info scope square
Scope for square:
Symbol a is an argument at stack/frame offset 68, length 4.
Symbol s is a local variable at frame offset -20, length 4.
(gdb) info scope add_y
Scope for add_y:
Symbol x is an argument at stack/frame offset 68, length 4.
Symbol s is a local variable at frame offset -20, length 4.
(gdb) quit
%

```

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

108

## Math Library Functions

- C standard math library
  - standard library supplied with every C compiler
  - predefined mathematical functions
    - e.g.  $\cos(x)$ ,  $\sqrt{x}$ , etc.
- Math library header file
  - contains math function declarations
  - `#include <math.h>`
- Math library linker file
  - contains math function definitions (pre-compiled)
    - library file `libm.a`
  - compiler needs to *link* against the math library
  - use option `-l $libraryname$`
  - Example: `gcc MathProgram.c -o MathProgram -lm`

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

109

## Math Library Functions

- Functions declared in `math.h` (part 1/2)
  - `double sqrt(double x);`                     $\sqrt{x}$
  - `double pow(double x, double y);`         $x^y$
  - `double exp(double x);`                     $e^x$
  - `double log(double x);`                     $\log(x)$
  - `double log10(double x);`                 $\log_{10}(x)$
  - `double ceil(double x);`                 $\lceil x \rceil$
  - `double floor(double x);`                 $\lfloor x \rfloor$
  - `double fabs(double x);`                 $|x|$
  - `double fmod(double x, double y);`     $x \bmod y$

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

110

## Math Library Functions

- Functions declared in `math.h` (part 2/2)

- <code>double cos(double x);</code>	<code>cos(x)</code>
- <code>double sin(double x);</code>	<code>sin(x)</code>
- <code>double tan(double x);</code>	<code>tan(x)</code>
- <code>double acos(double x);</code>	<code>acos(x)</code>
- <code>double asin(double x);</code>	<code>asin(x)</code>
- <code>double atan(double x);</code>	<code>atan(x)</code>
- <code>double cosh(double x);</code>	<code>cosh(x)</code>
- <code>double sinh(double x);</code>	<code>sinh(x)</code>
- <code>double tanh(double x);</code>	<code>tanh(x)</code>

## Math Library Functions

- Program example: `Function.c` (part 1/3)

```

/* Function.c: compute a math function table */
/* */
/* author: Rainer Doemer */
/* */
/* modifications: */
/* 10/28/04 RD initial version */

#include <stdio.h>
#include <math.h>

/* function definition */

double f(double x)
{
    return cos(x);
} /* end of f */

...

```



## Math Library Functions

- Program example: `Function.c` (part 2/3)

```

...
/* main function */

int main(void)
{
    /* variable definitions */
    double hi, lo, step;
    double x, y;

    /* input section */
    printf("Please enter the lower bound: ");
    scanf("%lf", &lo);
    printf("Please enter the upper bound: ");
    scanf("%lf", &hi);
    printf("Please enter the step size:  ");
    scanf("%lf", &step);

    ...

```

## Math Library Functions

- Program example: `Function.c` (part 3/3)

```

...

    /* computation and output section */
    for(x = lo; x <= hi; x += step)
    {
        y = f(x);
        printf("f(%10g) = %10g\n", x, y);
    } /* rof */

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

/* EOF */

```

## Math Library Functions

- Example session: `Function.c`

```

% vi Function.c
% gcc Function.c -o Function -Wall -ansi -lm
% Function
Please enter the lower bound: -0.5
Please enter the upper bound: 1.0
Please enter the step size: .1
f(   -0.5) =  0.877583
f(   -0.4) =  0.921061
f(   -0.3) =  0.955336
f(   -0.2) =  0.980067
f(   -0.1) =  0.995004
f(-2.77556e-17) =  1
f(    0.1) =  0.995004
f(    0.2) =  0.980067
f(    0.3) =  0.955336
f(    0.4) =  0.921061
f(    0.5) =  0.877583
f(    0.6) =  0.825336
f(    0.7) =  0.764842
f(    0.8) =  0.696707
f(    0.9) =  0.62161
f(    1) =  0.540302
%

```

## Standard Library Functions

- Standard C library
  - standard library supplied with every C compiler
  - predefined standard functions
    - e.g. `printf()`, `scanf()`, etc.
- C library header files
  - input/output function declarations `#include <stdio.h>`
  - standard function declarations `#include <stdlib.h>`
  - time function declarations `#include <time.h>`
  - etc.
- C library linker file
  - contains standard function definitions (pre-compiled)
    - library file `libc.a`
  - compiler *automatically links* against the standard library (no need to supply extra options)

## Standard Library Functions

- Functions declared in `stdlib.h` (partial list)
  - `int abs(int x);`
  - `long int labs(long int x);`
    - return the absolute value of a (long) integer `x`
  - `int rand(void);`
    - return a random value in the range 0 - `RAND_MAX`
    - `RAND_MAX` is a constant integer (e.g. 32767)
  - `void srand(unsigned int seed);`
    - initialize the random number generator with value `seed`
  - `void exit(int result);`
    - exit the program with return value `result`
  - `void abort(void);`
    - abort the program (with an error result)

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

117

## Standard Library Functions

- Random number generation
  - Standard library provides *pseudo* random number generator
    - `int rand(void);`
  - Pseudo random numbers are a sequence of values seemingly random in the range 0 - `RAND_MAX`
    - Computer is a *deterministic* machine
    - Sequence will always be the same
  - Start of sequence is determined by *seed* value
    - `void srand(unsigned int seed);`
  - Trick: Initialize random sequence with current time
    - header file `time.h` declares function `unsigned int time()`
    - `time(0)` returns number of seconds since Jan 1, 1970
    - at beginning of program, use:  
`srand(time(0));`

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

118

## Standard Library Functions

- Program example: `Dice.c` (part 1/4)

```

/* Dice.c: roll the dice                                     */
/* author: Rainer Doemer                                    */
/* modifications:                                          */
/* 10/28/04 RD initial version                             */

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

/* function definition */

int roll(void)
{
    int r;

    r = rand() % 6 + 1;
    /* printf("Rolled a %d.\n", r); */
    return r;
} /* end of roll */

...

```

## Standard Library Functions

- Program example: `Dice.c` (part 2/4)

```

...
/* main function */

int main(void)
{
    /* variable definitions */
    int i, n;
    int count1 = 0, count2 = 0, count3 = 0,
        count4 = 0, count5 = 0, count6 = 0;

    /* random number generator initialization */
    srand(time(0));

    /* input section */
    printf("Roll the dice: How many times? ");
    scanf("%d", &n);

    ...

```

## Standard Library Functions

- Program example: `Dice.c` (part 3/4)

```

... /* computation section */
for(i = 0; i < n; i++)
{ switch(roll())
  { case 1:
    { count1++; break; }
  case 2:
    { count2++; break; }
  case 3:
    { count3++; break; }
  case 4:
    { count4++; break; }
  case 5:
    { count5++; break; }
  case 6:
    { count6++; break; }
  default:
    { printf("INVALID ROLL!");
      exit(10); }
  } /* hctiws */
} /* rof */
...

```

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

121

## Standard Library Functions

- Program example: `Dice.c` (part 4/4)

```

...

/* output section */
printf("Rolled a 1 %5d times.\n", count1);
printf("Rolled a 2 %5d times.\n", count2);
printf("Rolled a 3 %5d times.\n", count3);
printf("Rolled a 4 %5d times.\n", count4);
printf("Rolled a 5 %5d times.\n", count5);
printf("Rolled a 6 %5d times.\n", count6);

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

/* EOF */

```

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

122

## Standard Library Functions

- Example session: `Dice.c`

```
% vi Dice.c
% gcc Dice.c -o Dice -Wall -ansi
% Dice
Roll the dice: How many times? 6000
Rolled a 1   963 times.
Rolled a 2   995 times.
Rolled a 3  1038 times.
Rolled a 4  1024 times.
Rolled a 5   984 times.
Rolled a 6   996 times.
% Dice
Roll the dice: How many times? 6000
Rolled a 1   977 times.
Rolled a 2  1043 times.
Rolled a 3  1012 times.
Rolled a 4  1001 times.
Rolled a 5   963 times.
Rolled a 6  1004 times.
%
```

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

123

## Data Structures

- Introduction

- Until now, we have used (mostly) single data elements of basic (non-composite) type
  - integral types
  - floating point types
- Most programs, however, require complex *data structures* using composite types
  - arrays, lists, queues, stacks
  - trees, graphs
  - dictionaries
- ANSI C provides built-in support for
  - arrays
  - structures, unions, enumerators
  - pointers

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

124

## Arrays

- Array data type in C
  - Composite data type
    - Type is an array of a sub-type (e.g. array of `int`)
  - Fixed number of elements
    - Array size is fixed at time of definition (e.g. 100 elements)
  - Element access by index (aka. subscript)
    - Element-access operator: `array[index]` (e.g. `A[42]`)
- Example:

```
int A[10]; /* array of ten integers */

A[0] = 42; /* access to elements */
A[1] = 100;
A[2] = A[0] + 5 * A[1];
```

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

125

## Arrays

- Array Indexing
  - Start counting from 0
    - First element has index 0
    - Last element has index *Size-1*
- Example:

```
int A[10];

A[0] = 42;
A[1] = 100;
A[2] = A[0] + 5 * A[1];
A[3] = -1;
A[4] = 44;
A[5] = 55;
/* ... */
A[9] = 99;
```

	<b>A</b>
0	42
1	100
2	542
3	-1
4	44
5	55
6	0
7	0
8	0
9	99

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

126

## Arrays

- Array Indexing
  - for loops are often very helpful
    - `for(i=0; i<N; i++)`  
`{...A[i]...}`

- Example:

```
int A[10];
int i;

for(i=0; i<10; i++)
  { A[i] = i*10 + i;
  }
for(i=0; i<10; i++)
  { printf("%d, ", A[i]);
  }
```

```
0, 11, 22, 33, 44, 55, 66, 77, 88, 99,
```

	A
0	0
1	11
2	22
3	33
4	44
5	55
6	66
7	77
8	88
9	99

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

127

## Arrays



- Array Indexing
  - Array indices are *not* checked by the compiler, nor at runtime!
  - Accessing an array with an *index out of range* results in undefined behavior!

- Example:

```
int A[10];
int i;

A[-1] = 42; /* INVALID ACCESS! */

for(i=0; i<=10; i++)
  /* INVALID LOOP RANGE! */
  { printf("%d, ", A[i]);
  }
```

	
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0
	

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

128



## Arrays

- Array Initialization
  - Static initialization at time of array definition
  - Initial elements listed in { }
- Example:

```
int A[10] = { 42, 100,
             310, 44,
             55, 0,
             3, 4,
             0, 99};
```

	A
0	42
1	100
2	310
3	44
4	55
5	0
6	3
7	4
8	0
9	99

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

129

## Arrays

- Array Initialization
  - Static initialization at time of array definition
  - Initial elements listed in { }
- Example:

```
int A[ ] = { 42, 100,
            310, 44,
            55, 0,
            3, 4,
            0, 99};
```

- With given initializer list, array size may be omitted
  - automatically determined

	A
0	42
1	100
2	310
3	44
4	55
5	0
6	3
7	4
8	0
9	99

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

130

## Arrays

- Array Initialization
  - Static initialization at time of array definition
  - Initial elements listed in { }
- Example:
 

```
int A[10] = { 1, 2, 3};
```
- With given initializer list *and* array size, unlisted elements are zero-initialized
  - array is filled up with zeros

	A
0	1
1	2
2	3
3	0
4	0
5	0
6	0
7	0
8	0
9	0

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

131

## Arrays

- Multi-dimensional Arrays
  - Array of an array...
- Example:

```
int M[3][2] = {{1, 2},
               {3, 4},
               {5, 6}};

int i, j;

for(i=0; i<3; i++)
  { for(j=0; j<2; j++)
    { printf("%d ",
             M[i][j]);
      }
    printf("\n");
  }
```

M	0	1
0	1	2
1	3	4
2	5	6

```
1 2
3 4
5 6
```

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

132

## Arrays

- Operator associativity and precedence
  - parentheses, array access      ( ), []      left to right
  - unary operators                    +, -, !, ++, --      right to left
  - type casting                        ( *typename* )      right to left
  - multiplication, division, modulo    \*, /, %      left to right
  - addition, subtraction                +, -      left to right
  - shift left, shift right                <<, >>      left to right
  - relational operators                 <, <=, >=, >      left to right
  - equality                                ==, !=      left to right
  - logical and                            &&      left to right
  - logical or                             ||      left to right
  - conditional operator                 ?:      left to right
  - assignment operators                =, +=, \*=, etc.      right to left
  - comma operator                       ,      left to right

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer 133

## Arrays

- Program example: **Histogram.c**
  - Display a simple bar chart for 10 integer values
- Desired output:

```
% Histogram
Please enter data value 1: 111
Please enter data value 2: 222
Please enter data value 3: 33
Please enter data value 4: 333
[...]
Please enter data value 10: 111
1: 111 *****
2: 222 *****
3: 33 ****
4: 333 *****
[...]
10: 111 *****
%
```

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer 134

## Arrays

- Program example: `Histogram.c` (part 1/3)

```

/* Histogram.c: print a histogram of data values */
/* author: Rainer Doemer */
/* modifications: */
/* 11/02/04 RD initial version */

#include <stdio.h>

/* constants */
#define NUM_ROWS 10

/* main function */
int main(void)
{
    /* variable definitions */
    int Data[NUM_ROWS];
    int i, j, max;
    double scale;

    ...

```

## Arrays

- Program example: `Histogram.c` (part 2/3)

```

...
/* input section */
for(i = 0; i < NUM_ROWS; i++)
{ printf("Please enter data value %2d: ", i+1);
  scanf("%d", &Data[i]);
} /* rof */

/* computation section */
max = 0;
for(i = 0; i < NUM_ROWS; i++)
{ if (Data[i] > max)
  { max = Data[i];
  } /* fi */
} /* rof */
scale = 70.0 / max;

...

```

## Arrays

- Program example: `Histogram.c` (part 3/3)

```

...
/* output section */
for(i = 0; i < NUM_ROWS; i++)
{ printf("%2d: %5d ", i+1, Data[i]);
  for(j = 0; j < Data[i]*scale; j++)
  { printf("*");
    } /* rof */
  printf("\n");
} /* rof */

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

/* EOF */

```

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

137

## Arrays

- Example session: `Histogram.c`

```

% vi Histogram.c
% gcc Histogram.c -o Histogram -Wall -ansi
% Histogram
Please enter data value 1: 11
Please enter data value 2: 22
Please enter data value 3: 3
Please enter data value 4: 33
Please enter data value 5: 44
Please enter data value 6: 55
Please enter data value 7: 66
Please enter data value 8: 33
Please enter data value 9: 22
Please enter data value 10: 22
1: 11 *****
2: 22 *****
3: 3 ****
4: 33 *****
5: 44 *****
6: 55 *****
7: 66 *****
8: 33 *****
9: 22 *****
10: 22 *****
%

```

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

138

## Arrays

- Earlier program example: `Dice.c` (part 1/4)

```

/* Dice.c: roll the dice                                */
/* author: Rainer Doemer                                */
/* modifications:                                       */
/* 10/28/04 RD initial version                          */

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

/* function definition */

int roll(void)
{
    int r;

    r = rand() % 6 + 1;
    /* printf("Rolled a %d.\n", r); */
    return r;
} /* end of roll */
...

```

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

139

## Arrays

- Earlier program example: `Dice.c` (part 2/4)

```

...
/* main function */

int main(void)
{
    /* variable definitions */
    int i, n;
    int count1 = 0, count2 = 0, count3 = 0,
        count4 = 0, count5 = 0, count6 = 0;

    /* random number generator initialization */
    srand(time(0));

    /* input section */
    printf("Roll the dice: How many times? ");
    scanf("%d", &n);

    ...

```

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

140

## Arrays

- Earlier program example: `Dice.c` (part 3/4)

```

... /* computation section */
for(i = 0; i < n; i++)
{ switch(roll())
  { case 1:
    { count1++; break; }
  case 2:
    { count2++; break; }
  case 3:
    { count3++; break; }
  case 4:
    { count4++; break; }
  case 5:
    { count5++; break; }
  case 6:
    { count6++; break; }
  default:
    { printf("INVALID ROLL!");
      exit(10); }
  } /* hctiws */
} /* rof */
...

```

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

141

## Arrays

- Earlier program example: `Dice.c` (part 4/4)

```

...

/* output section */
printf("Rolled a 1 %5d times.\n", count1);
printf("Rolled a 2 %5d times.\n", count2);
printf("Rolled a 3 %5d times.\n", count3);
printf("Rolled a 4 %5d times.\n", count4);
printf("Rolled a 5 %5d times.\n", count5);
printf("Rolled a 6 %5d times.\n", count6);

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

/* EOF */

```

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

142

## Arrays

- Improved program example: `Dice2.c` (part 1/3)

```

/* Dice2.c: roll the dice */
/* author: Rainer Doemer */
/* modifications: */
/* 11/04/04 RD version using arrays */
/* 10/28/04 RD initial version */

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

/* function definition */

int roll(void)
{
    int r;

    r = rand() % 6 + 1;
    /* printf("Rolled a %d.\n", r); */
    return r;
} /* end of roll */

...

```

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

143

## Arrays

- Improved program example: `Dice2.c` (part 2/3)

```

...
/* main function */

int main(void)
{
    /* variable definitions */
    int i, n;
    int count[6] = { 0, 0, 0, 0, 0, 0 };

    /* random number generator initialization */
    srand(time(0));

    /* input section */
    printf("Roll the dice: How many times? ");
    scanf("%d", &n);

    ...

```

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

144



## Arrays

- Improved program example: `Dice2.c` (part 3/3)

```

...
/* computation section */
for(i = 0; i < n; i++)
  { count[roll()-1]++;
    } /* rof */

/* output section */
for(i = 0; i < 6; i++)
  { printf("Rolled a %d %5d times.\n",
          i+1, count[i]);
    } /* rof */

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

/* EOF */

```

## Arrays

- Example session: `Dice2.c`

```

% vi Dice2.c
% gcc Dice2.c -o Dice2 -Wall -ansi
% Dice2
Roll the dice: How many times? 6000
Rolled a 1 1009 times.
Rolled a 2 1005 times.
Rolled a 3 962 times.
Rolled a 4 998 times.
Rolled a 5 996 times.
Rolled a 6 1030 times.
% Dice2
Roll the dice: How many times? 6000
Rolled a 1 1042 times.
Rolled a 2 983 times.
Rolled a 3 972 times.
Rolled a 4 979 times.
Rolled a 5 1022 times.
Rolled a 6 1002 times.
%

```

## Passing Arguments to Functions

- Pass by Value
  - only the *current value* is passed as argument
  - the parameter is a *copy* of the argument
  - changes to the parameter *do not* affect the argument
- Pass by Reference
  - a *reference* to the object is passed as argument
  - the parameter is a *reference* to the argument
  - changes to the parameter *do* affect the argument
- In ANSI C, ...
  - ... basic types are passed by value
  - ... arrays are passed by reference

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

147

## Passing Arguments to Functions

- Example: Pass by Value

```
void f(int p)
{
    printf("p before modification is %d\n", p);
    p = 42;
    printf("p after modification is %d\n", p);
}

int main(void)
{
    int a = 0;
    printf("a before function call is %d\n", a);
    f(a);
    printf("a after function call is %d\n", a);
}
```

```
a before function call is 0
p before modification is 0
p after modification is 42
a after function call is 0
```

**Changes to the parameter *do not* affect the argument!**

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

148

## Passing Arguments to Functions

- Example: Pass by Reference

```
void f(int p[2])
{
    printf("p[1] before modification is %d\n", p[1]);
    p[1] = 42;
    printf("p[1] after modification is %d\n", p[1]);
}

int main(void)
{
    int a[2] = {0, 0};
    printf("a[1] before function call is %d\n", a[1]);
    f(a);
    printf("a[1] after function call is %d\n", a[1]);
}
```

```
a[1] before function call is 0
p[1] before modification is 0
p[1] after modification is 42
a[1] after function call is 42
```

Changes to the parameter *do* affect the argument!

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

149

## Character Arrays: Strings

- Text is represented by character arrays (aka. *strings*)
  - Strings are null-terminated arrays of characters
  - String output
    - `printf()` format specifier: `"%s"`
- Example:

```
char s1[] = {'H', 'e', 'l', 'l', 'o', 0};

printf("s1 is %s.\n", s1);
```

```
s1 is Hello.
```

	s1
0	'H'
1	'e'
2	'l'
3	'l'
4	'o'
5	0

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

150

## Character Arrays: Strings

- Text is represented by character arrays (aka. *strings*)
  - Strings are null-terminated arrays of characters
  - String output
    - `printf()` format specifier: `"%s"`
- Example:

```
char s1[] = {'H', 'e', 'l', 'l', 'o', 0};
char s2[] = "Hello";

printf("s1 is %s.\n", s1);
printf("s2 is %s.\n", s2);
```

```
s1 is Hello.
s2 is Hello.
```

s2	
0	'H'
1	'e'
2	'l'
3	'l'
4	'o'
5	0

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

151

## Character Arrays: Strings

- Text is represented by character arrays (aka. *strings*)
  - Strings are null-terminated arrays of characters
  - String output
    - `printf()` format specifier: `"%s"`
- Example:

```
char s1[] = {'H', 'e', 'l', 'l', 'o', 0};
char s2[] = "Hello";

printf("s1 is %s.\n", s1);
printf("s2 is %s.\n", s2);
s1[1] = 'i';
s1[2] = 0;
printf("Modified s1 is %s.\n", s1);
```

```
s1 is Hello.
s2 is Hello.
Modified s1 is Hi.
```

s1	
0	'H'
1	'i'
2	0
3	'l'
4	'o'
5	0

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

152

## Character Arrays: Strings

- Text is represented by character arrays (aka. *strings*)
  - Strings are null-terminated arrays of characters
  - String input
    - `scanf()` format specifier: `"%Ns"`, where `N` specifies maximum field width = array size - 1
    - address argument can be `&string[0]`

- Example:

```
char s1[6];
printf("Enter a string: ");
scanf("%5s", &s1[0]);
printf("s1 is %s.\n", s1);
```

```
Enter a string: Test
s1 is Test.
```

	s1
0	'T'
1	'e'
2	's'
3	't'
4	0
5	0

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

153

## Character Arrays: Strings

- Text is represented by character arrays (aka. *strings*)
  - Strings are null-terminated arrays of characters
  - String input
    - `scanf()` format specifier: `"%Ns"`, where `N` specifies maximum field width = array size - 1
    - address argument can be `&string[0]` or simply `string`

- Example:

```
char s1[6];
printf("Enter a string: ");
scanf("%5s", s1);
printf("s1 is %s.\n", s1);
```

```
Enter a string: Test
s1 is Test.
```

	s1
0	'T'
1	'e'
2	's'
3	't'
4	0
5	0

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

154

## Character Arrays: Strings

- Text is represented by character arrays (aka. *strings*)
  - Strings are null-terminated arrays of characters
  - Characters are represented by numeric values
  - ASCII table defines character values 0-127
- Example:

```
char s1[] = "ABC12";
int i = 0;

while(s1[i])
  { printf("%c = %d\n", s1[i], s1[i]);
    i++; }
```

```
A = 65
B = 66
C = 67
1 = 49
2 = 50
```

	s1
0	'A'
1	'B'
2	'C'
3	'1'
4	'2'
5	0

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

155

## Character Arrays: Strings

- ASCII Table
  - American Standard Code for Information Interchange

0 NUL	1 SOH	2 STX	3 ETX	4 EOT	5 ENQ	6 ACK	7 BEL
8 BS	9 HT	10 NL	11 VT	12 NP	13 CR	14 SO	15 SI
16 DLE	17 DC1	18 DC2	19 DC3	20 DC4	21 NAK	22 SYN	23 ETB
24 CAN	25 EM	26 SUB	27 ESC	28 FS	29 GS	30 RS	31 US
32	33 !	34 "	35 #	36 \$	37 %	38 &	39 '
40 (	41 )	42 *	43 +	44 ,	45 -	46 .	47 /
48 0	49 1	50 2	51 3	52 4	53 5	54 6	55 7
56 8	57 9	58 :	59 ;	60 <	61 =	62 >	63 ?
64 @	65 A	66 B	67 C	68 D	69 E	70 F	71 G
72 H	73 I	74 J	75 K	76 L	77 M	78 N	79 O
80 P	81 Q	82 R	83 S	84 T	85 U	86 V	87 W
88 X	89 Y	90 Z	91 [	92 \	93 ]	94 ^	95 _
96 `	97 a	98 b	99 c	100 d	101 e	102 f	103 g
104 h	105 i	106 j	107 k	108 l	109 m	110 n	111 o
112 p	113 q	114 r	115 s	116 t	117 u	118 v	119 w
120 x	121 y	122 z	123 {	124	125 }	126 ~	127 DEL

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

156

## Character Arrays: Strings

- Case Study: *Bubble Sort*
  - Task: Sort an array of strings alphabetically
  - Input: Array of 10 strings entered by the user
  - Output: Array of 10 strings in alphabetical order
- Approach: Divide and Conquer
  - Step 1: Let user enter 10 strings
  - Step 2: Sort the array of strings
  - Step 3: Output the strings in order

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

157

## Character Arrays: Strings

- Case Study: *Bubble Sort*
  - Task: Sort an array of strings alphabetically
  - Input: Array of 10 strings entered by the user
  - Output: Array of 10 strings in alphabetical order
- Approach: Divide and Conquer
  - Step 1: Let user enter 10 strings
  - Step 2: Sort the array of strings
    - Algorithm
      - in 9 passes, compare adjacent pairs of strings and swap the pair if they are not in alphabetical order
    - String comparison
      - compare character pairs alphabetically: use ASCII table!
    - String swap (exchange two strings in place)
      - swap each character pair in the two strings
  - Step 3: Output the strings in order

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

158

## Character Arrays: Strings

- Program example: `BubbleSort.c` (part 1/7)

```

/* BubbleSort.c: sort strings alphabetically */
/* author: Rainer Doemer */
/* modifications: */
/* 11/01/06 RD swap only adjacent elements */
/* 11/06/04 RD initial version */

#include <stdio.h>

/* constant definitions */

#define NUM 10 /* ten strings */
#define LEN 20 /* of length 20 */

/* function declarations */

void EnterText(char Text[NUM][LEN]);
void PrintText(char Text[NUM][LEN]);
int CompareStrings(char s1[LEN], char s2[LEN]);
void SwapStrings(char s1[LEN], char s2[LEN]);
void BubbleSort(char Text[NUM][LEN]);
...

```

## Character Arrays: Strings

- Program example: `BubbleSort.c` (part 2/7)

```

...

/* function definitions */

/* let the user enter the text array */

void EnterText(char Text[NUM][LEN])
{
    int i;

    for(i = 0; i < NUM; i++)
    { printf("Enter text string %2d: ", i+1);
      scanf("%19s", Text[i]);
    } /* rof */
} /* end of EnterText */

...

```



## Character Arrays: Strings

- Program example: `BubbleSort.c` (part 3/7)

```

...

/* print the text array on the screen      */

void PrintText(char Text[NUM][LEN])
{
    int i;

    for(i = 0; i < NUM; i++)
        { printf("String %2d: %s\n", i+1, Text[i]);
          } /* rof */
} /* end of PrintText */

...

```

## Character Arrays: Strings

- Program example: `BubbleSort.c` (part 4/7)

```

...

/* alphabetically compare strings s1 and s2: */
/* return -1, if string s1 < string s2      */
/* return 0, if string s1 = string s2      */
/* return 1, if string s1 > string s2      */

int CompareStrings(char s1[LEN], char s2[LEN])
{
    int i;

    for(i = 0; i < LEN; i++)
        { if (s1[i] > s2[i])
          { return(1); }
          if (s1[i] < s2[i])
          { return(-1); }
          if (s1[i] == 0 || s2[i] == 0)
          { break; }
          } /* rof */
    return 0;
} /* end of CompareStrings */

...

```

## Character Arrays: Strings

- Program example: `BubbleSort.c` (part 5/7)

```

...

/* swap/exchange the strings s1 and s2 in place */

void SwapStrings(char s1[LEN], char s2[LEN])
{
    int i;
    char c;

    for(i = 0; i < LEN; i++)
    { c = s1[i];
      s1[i] = s2[i];
      s2[i] = c;
    } /* rof */
} /* end of SwapStrings */

...

```

## Character Arrays: Strings

- Program example: `BubbleSort.c` (part 6/7)

```

...

/* sort the text array by comparing every pair */
/* of strings; if the pair of strings is not in */
/* alphabetical order, swap it */

void BubbleSort(char Text[NUM][LEN])
{
    int p, i;

    for(p = 1; p < NUM; p++)
    { for(i = 0; i < NUM-1; i++)
      { if (CompareStrings(Text[i], Text[i+1]) > 0)
        { SwapStrings(Text[i], Text[i+1]);
          } /* fi */
        } /* rof */
      } /* rof */
    } /* end of BubbleSort */

...

```

## Character Arrays: Strings

- Program example: `BubbleSort.c` (part 7/7)

```

...
/* main function: enter, sort, print the text */
int main(void)
{
    /* local variables */
    char Text[NUM][LEN]; /* NUM strings, length LEN */

    /* input section */
    EnterText(Text);

    /* computation section */
    BubbleSort(Text);

    /* output section */
    PrintText(Text);

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

/* EOF */

```

EECS10: Computational Methods in ECE, Review 9-17

(c) 2009 R. Doemer

165

## Character Arrays: Strings

- Example session: `BubbleSort.c`

```

% vi BubbleSort.c
% gcc BubbleSort.c -o BubbleSort -Wall -ansi
% BubbleSort
Enter text string 1: Charlie
Enter text string 2: William
Enter text string 3: Donald
Enter text string 4: John
Enter text string 5: Jane
Enter text string 6: Jessie
Enter text string 7: Donald
Enter text string 8: Henry
Enter text string 9: George
Enter text string 10: Emily
String 1: Charlie
String 2: Donald
String 3: Donald
String 4: Emily
String 5: George
String 6: Henry
String 7: Jane
String 8: Jessie
String 9: John
String 10: William
%

```

EE