CS1010E Topic 3:
C Basic & Choices

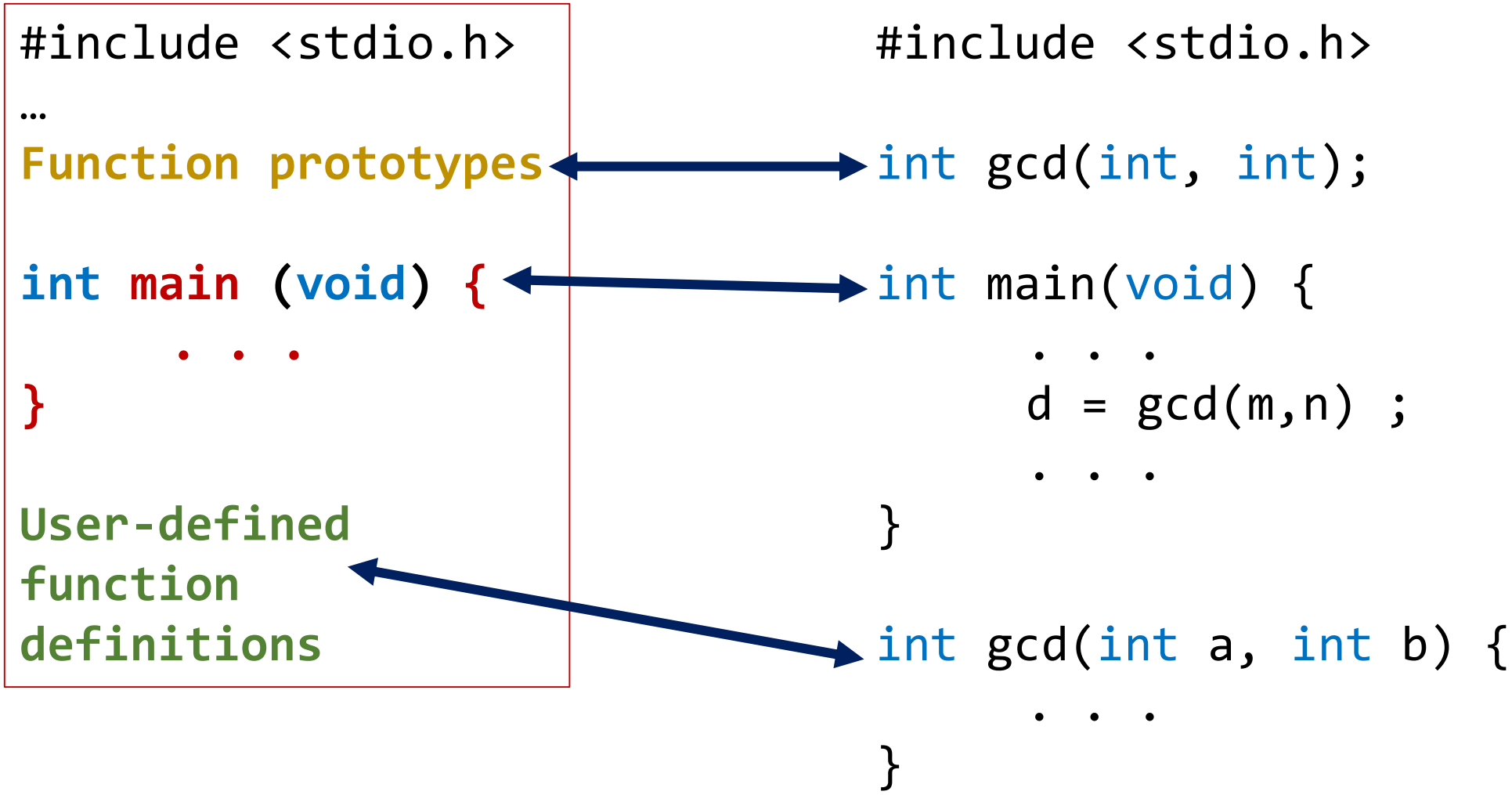
Siau-Cheng KHOO
Block COM2, Room 04-11, +65 6516 6730
www.comp.nus.edu.sg/~khoosc
khoosc@nus.edu.sg

Semester II, 2017/2018

Lecture Outline

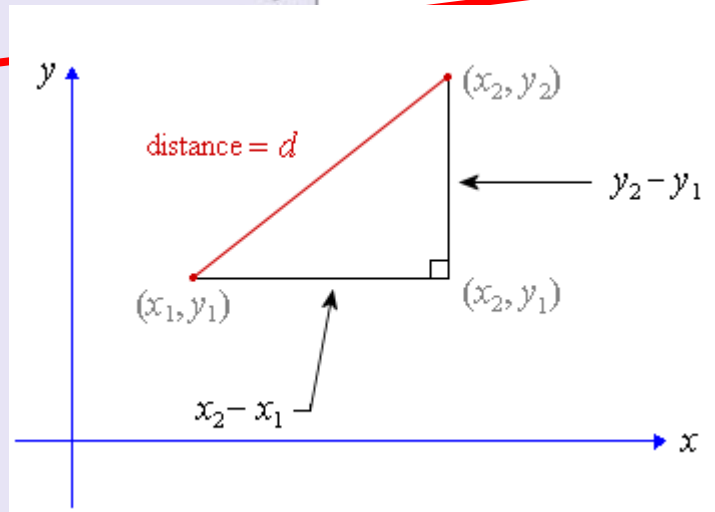
- Revision: Modular Design
- Better Understanding of Structure of C programs
  - Character Data
  - Standard Input and Output statements
- Making Choices
  - If statements and logical operations
  - If-else statements
  - Nested if statements
  - Switch statements

Recall: User-defined functions



Recall: Built-in math functions library

```
/*-------------------------------------------------------*/
/*  Program chapter1_1                                   */
/*                                                       */
/*  This program computes the                            */
/*  distance between two points.                         */

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

int main(void)
{
   /*  Declare and initialize variables.  */
   double x1=1, y1=5, x2=4, y2=7,
          side_1, side_2, distance;

   /*  Compute sides of a right triangle.  */
   side_1 = x2 - x1;
   side_2 = y2 - y1;
   distance = sqrt(side_1*side_1 + side_2*side_2);

   /*  Print distance.  */
   printf("The distance between the two points is "
          "%5.2f \n",distance);

   /*  Exit program.  */
   return 0;
}
/*-------------------------------------------------------*/
```

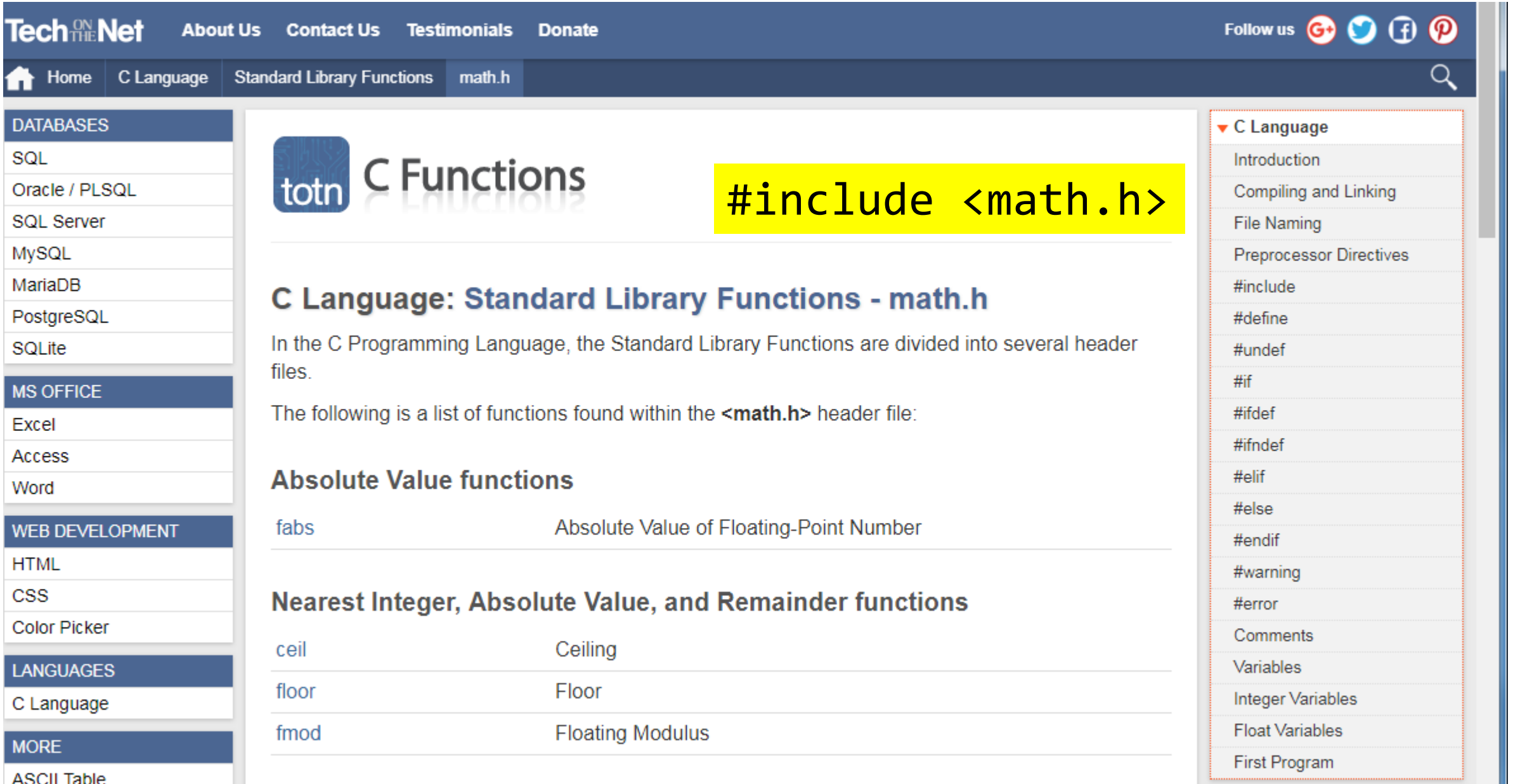


$$\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$$

<math.h> contains mathematics functions such as sqrt that can be used in this program.

```
gcc -lm chapter1_1.c
```

https://www.techonthenet.com/c_language/standard_library_functions/math_h/index.php



C Language: floor function (Floor)

In the C Programming Language, the **floor function** returns the largest integer that is smaller than or equal to *x* (ie: rounds downs the nearest integer).

```
#include <math.h>
```

Syntax

The syntax for the floor function in the C Language is:

```
double floor(double x);
```

Parameters or Arguments

```
gcc -lm -Wall yourProgram.c
```

x

The value to round down to the nearest integer.

Returns

The floor function returns the largest integer that is smaller than or equal to *x*.

<math.h> includes:

```
double      ceil(double);
double      cos(double);
double      exp(double);
double      fabs(double);
double      floor(double);
double      fmax(double, double);
double      fmin(double, double);
double      log(double);
double      log10(double);
double      pow(double, double);
double      round(double);
double      sqrt(double);
```

Lecture Outline

- Revision: Modular Design
- Better Understanding of Structure of C programs
  - Character Data
  - Standard Input and Output statements
- Making Choices
  - If statements and logical operations
  - If-else statements
  - Nested if statements
  - Switch statements

Character Data

- Recall – all info stored in a computer is represented internally as sequences of binary digits (0 and 1)
- Each **character** corresponds to a **binary code** value
  - `'a'` encoded as `01100001` equivalent to **97**
  - `'3'` encoded as `00110011` equivalent to **51**
- The most commonly used binary code is **ASCII** and **EBCDIC**
- We assume that ASCII code is used to represent characters

128 ASCII Character Codes

Etter's book: Appendix B

Character	Integer Equivalent	Binary Equivalent
NUL (Binary Zero)	0	00000000
HT (Horizontal Tab) `'\t'`	9	00001001
LF (Line Feed/New Line) `'\n'`	10	00001010
0	48	00110000
9	57	00111001
A	65	01000001
Z	90	01011010
a	97	01100001
z	122	01111010

Character Data

- Can be represented by constants or by variables
- A character is enclosed in single quotes, such as `'A', 'b', '3'`
- A variable that is going to contain a character is defined as an integer or a character data type

```
char c = 't' ;
```

- Once a character is stored in memory as a binary value, it can be interpreted as a character or as an integer.

Data type	Constant	Binary code
char	'3'	00110011
int	3	00000011

Standard Input and Output

- Need a statement to print values of variables to the screen
- Need a statement allowing us to enter values from the keyboard when the program is executing
- Must have preprocessor directive at the beginning of program:
  - `#include <stdio.h>`
- This directive gives the compiler the information it needs to check references to the input / output functions in the Standard C library

The `printf` function/statement

- Allows us to print values and explanatory text to the screen.

```
int a ;
. . .
printf("The result of gcd is %d.\n", a) ;
```

newline character:
Causes a skip to a new line on the screen

Control string

Enclosed in double quotation marks

Can contain text, and/or

Conversion specifiers

The `printf` function/statement

- Allows us to print values and explanatory text to the screen.

```
int a ;
. . .
printf("The result of gcd is %d.\n", a) ;
```

Conversion specifier

Matches the argument
In terms of type, and
Print its value in certain way

More arguments

Conversion Specifiers for Output statements

Variable Type	Specifier
int	%i, %d
long	%li, %ld
float	%f, %e, %E, …
long double	%LF, %Le, %LE, …

Specifier	Value Printed
%i	- 1 4 5
%4d	- 1 4 5
%6i	␣␣- 1 4 5
%-6i	- 1 4 5 ␣␣

Specifier	Value Printed
%f	157.89260
%6.2f	157.89
%+7.5f	+157.89260
%.3E	1.579E+02
%.1e	1.6e+02

Exercise on printf

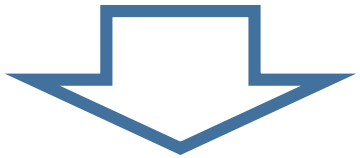
```
printf("Your results are: %d, %e, %10d.\n",. . .);
```

How many arguments should this `printf` statement have to be complete?

printf: Splitting lines

- If a printf statement is too long, you should split it over two lines

```
printf("The distance between the points is %5.2f
\n", distance) ;
```



```
printf("The distance between the "
       "points is %5.2f \n", distance) ;
```

Escape Character

- The backslash (\) is called an escape character when it is used in a control string.
- The compiler combines it with the character that follows it and gives a special meaning to the combination.
  - A skip to a new line: \n `"what!\n"`
  - A double quote in a control string: \“ `"you say, \" \""`
  - A single backslash in a control string: \\

Exercise on printf

How to print the following sentence on your screen?

```
The 'crow' said, "Gosh!"
```

The scanf function/statement

- Allows you to enter values to your program from the keyboard when it is executed.

```
scanf ("%i", &year) ;
```

Control string

The entered value will be held by variable `year`

Address operator (&): storing entered at the address obtained from `year`

A common error in scanf statements is to omit the address operator for the identifiers.

scanf statements: Multiple input

- If we wish to read more than one value from the keyboard, we can use statements such as:

```
scanf("%i %lf", &age, &weight_kg) ;
```

- The values entered, by the user, must be **separated** **by at least one blank**; they can be on the same line or on different lines.
- We always precede the scanf statement by a printf statement, in order to **prompt** the user to enter the values.

```
printf("Enter the age in years and the weight in kg: ") ;
scanf(“%i %lf“, &age, &weight_kg) ;
```

```
Enter the age in years and the weight in kg: 39  70.3
```

Lecture Outline

- Revision: Modular Design
- Better Understanding of Structure of C programs
  - Standard Input and Output statements
- Making Choices – Conditional Statements & Logical Operations
  - If-else statements
  - Nested if statements
  - Switch statements

Problem #4

Given a 4-digit number, check if there are more big digits or more small digits in this number.

A digit is considered big if it is one of the 5, 6, 7, 8, 9.
A digit is considered small if it is one of the 0, 1, 2, 3, 4.

```
Please enter a four digit number: 8925
There are more big digits.
```

```
Please enter a four digit number: 9042
There are more small digits.
```

```
Please enter a four digit number: 4075
There are equal number of big and small digits.
```

Problem #4 (Analysis & Design)

Given a 4-digit number, check if there are more big digits or more small digits in this number.

1. count = 0
2. Take out the rightmost digit d from the number;
3. Check if d is big or small
    1. If it is big, increment count by 1
    2. If it is small, decrement count by 1
4. number ← Drop the rightmost digit off the number
5. Do from Step 2 to Step 4 another 3 times, then
6. If (count > 0) the number has more big digits
7. Otherwise if (count < 0) the number has more small digits
8. Otherwise the number has equal number of big and small digits

```
8925

d = 5

increase count by 1, result: 1

892

count: 0 1 2
```

Problem #4 (Analysis & Design)

Given a 4-digit number, check if there are more big digits or more small digits in this number.

1. count = 0
2. **Take out the rightmost digit d from num;**
3. Check if d is big or small
   1. If it is big, increment count by 1
   2. If it is small, decrement count by 1
4. number ← Drop the rightmost digit off the number
5. Do from Step 2 to Step 4 another 3 times, then
6. If (count > 0) the number has more big digits
7. Otherwise if (count < 0) the number has more small digits
8. Otherwise the number has equal number of big and small digits

`d = 5` `8925`

```
d = num % 10 ;
```

Problem #4 (Analysis & Design)

Given a 4-digit number, check if there are more big digits or more small digits in this number.

1. count = 0
2. **Take out the rightmost digit d from num;** `d = num % 10 ;`
3. Check if d is big or small
   1. If it is big, increment count by 1
   2. If it is small, decrement count by 1 **8925** ➡ **892**
4. **num ← Drop the rightmost digit off num** `num = num / 10 ;`
5. Do from Step 2 to Step 4 another 3 times, then
6. If (count > 0) the number has more big digits
7. Otherwise if (count < 0) the number has more small digits
8. Otherwise the number has equal number of big and small digits

Problem #4 (Analysis & Design)

Given a 4-digit number, check if there are more big digits or more small digits in this number.

1. count = 0
2. Take out the rightmost digit d from num;
3. **Check if d is big or small**
   1. **If it is big, increment count by 1**
   2. **If it is small, decrement count by 1**
4. num ← Drop the rightmost digit off num
5. Do from Step 2 to Step 4 another 3 times, then
6. If (count > 0) the number has more big digits
7. Otherwise if (count < 0) the number has more small digits
8. Otherwise the number has equal number of big and small digits

```
d = num % 10 ;
if (d > 4) {
   count++ ;
} else {
   count-- ;
}
num = num / 10 ;
```

Problem #4 (Analysis & Design)

Given a 4-digit number, check if there are more big digits or more small digits in this number.

1. count = 0
2. Take out the rightmost digit d from num;
3. Check if d is big or small
   1. If it is big, increment count by 1
   2. If it is small, decrement count by 1
4. num ← Drop the rightmost digit off num
5. Do from Step 2 to Step 4 another 3 times, then
6. **If (count > 0) the number has more big digits**
7. **Otherwise if (count < 0) the number has more small digits**
8. **Otherwise the number has equal number of big and small digits**

```
d = num % 10 ;
if (d > 4) {
    count++ ;
} else {
    count-- ;
}
num = num / 10 ;
```

```
if (count > 0) {
   printf(…) ;
} else if (count < 0) {
   printf(…) ;
} else {
   printf(…) ;
}
```

Problem #4 Implementation

```
int checkLastDigit(int num, int counter) {
        int digit ;

        digit = num % 10 ;
        if (digit > 4) {  /* big number */
                counter++ ;
        }
        else if (digit < 5) { /* small number */
                counter-- ;
        }
        return counter;
}
```

```
d = num % 10 ;
if (d > 4) {
   count++ ;
} else {
   count-- ;
}
```

Problem #4 Implementation

```
int fourDigits ;
int count = 0 ;

printf("Please enter a four digit number: ") ;
scanf("%d", &fourDigits) ; /* input */

count = checkLastDigit(fourDigits, count) ;
fourDigits /= 10 ;
count = checkLastDigit(fourDigits, count) ;
fourDigits /= 10 ;
count = checkLastDigit(fourDigits, count) ;
fourDigits /= 10 ;
count = checkLastDigit(fourDigits, count) ;
```

```
int checkLastDigit(int num, int counter) {
        int digit ;

        digit = num % 10 ;
        if (digit > 4) {  /* big number */
                counter++ ;
        }
        else if (digit < 5) { /* small number */
                counter-- ;
        }
        return counter;
}
```

Problem #4 Implementation

```
if (count > 0) {
   printf("There are more big digits.\n") ;
} else if (count < 0) {
   printf("There are more small digits.\n") ;
} else {
   printf("There are equal number of"
          "big and small digits.\n") ;
}
```

Problem #4 Implementation

```
#include <stdio.h>

int checkLastDigit(int,int);

int main(void) {
    int fourDigits ;
    int count = 0 ;

    printf("Please enter a four digit number: ") ;
    scanf("%d", &fourDigits) ; /* input */

    count = checkLastDigit(fourDigits, count) ;
    fourDigits /= 10 ;
    count = checkLastDigit(fourDigits, count) ;
    fourDigits /= 10 ;
    count = checkLastDigit(fourDigits, count) ;
    fourDigits /= 10 ;
    count = checkLastDigit(fourDigits, count) ;

    if (count > 0) {
       printf("There are more big digits.\n") ;
    } else if (count < 0) {
       printf("There are more small digits.\n") ;
    } else {
       printf("There are equal number of"
              "big and small digits.\n") ;
    }

    return 0 ;
}

int checkLastDigit(int num, int counter) {
        int digit ;

        digit = num % 10 ;
        if (digit > 4) {  /* big number */
                counter++ ;
        }
        else if (digit < 5) { /* small number */
                counter-- ;
        }
        return counter;
}
```

Selection Structures

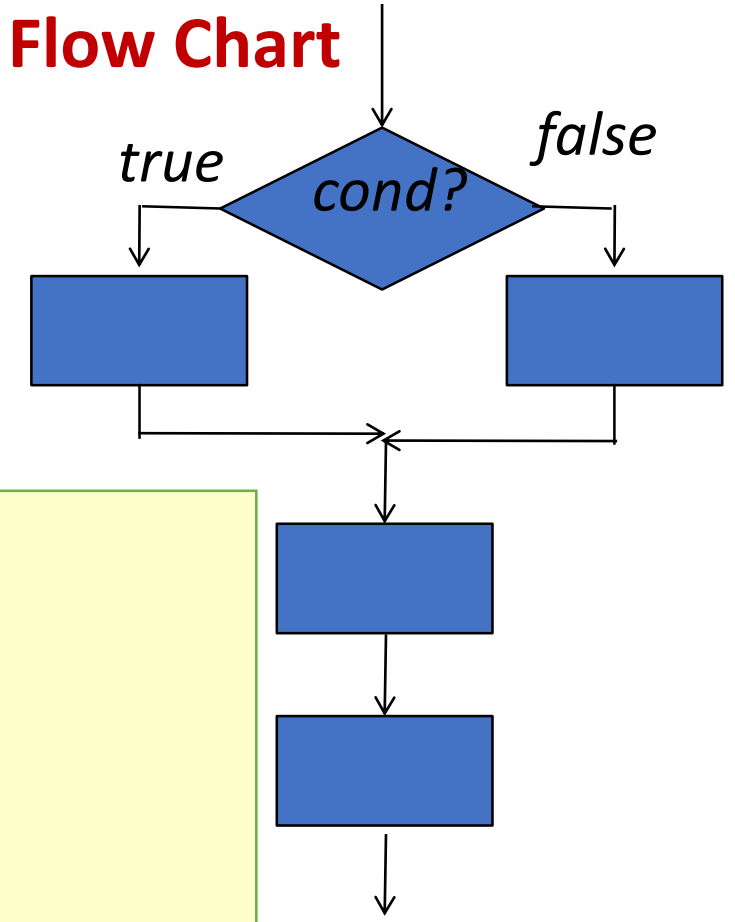
- C provides two control structures that allow you to select a group of statements to be executed or skipped when certain conditions are met.

if … else …

switch

if and if-else statements

- if-else statement



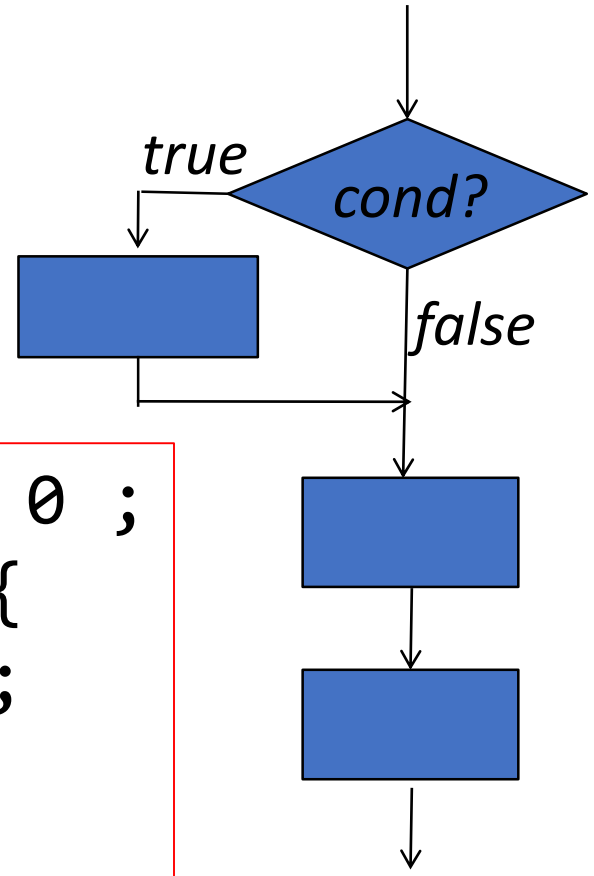

```
if ( condition ) {
     /* Execute these statements if
        condition evaluates to TRUE   */
}
else {
     /* Execute these statements if
        condition evaluates to FALSE */
}
```

`if` and `if-else` statements

- `if` statement

```
if ( condition ) {
     /* Execute these
        statements if
        condition evaluates
        to TRUE */
}
```

```
int count = 0 ;
if (1 < 0) {
   count++ ;
}
count++ ;
count++ ;
```



Flow Chart

What happen if the condition evaluates to false?

Condition

- An expression evaluated to either **true** or **false**
- Composed of expressions combined with **relational operators.**
- Examples: (a <= 10), (count > max), (value != -9)

Relational Operator	Interpretation
<	is less than
<=	is less than or equal to
>	is greater than
>=	is greater than or equal to
==	is equal to
!=	is not equal to

Truth Values

- Boolean values: true or false.
- There is **no** boolean type in ANSI C. Instead, we use integers:
  - 0 to represent false
  - Any other value to represent true (1 is used as the representative value for true in output)
- Example:

```
int a = (2 > 3);
int b = (3 > 2);

printf("a = %d; b = %d\n", a, b);
```

```
a = 0; b = 1
```

Logical Operators

- Complex condition: combining two or more boolean expressions.
  - Examples:
    - If temperature is greater than 40C **or** blood pressure is greater than 200, go to A&E immediately.
- Logical operators are needed: && (and), || (or), ! (not).

A	B	A && B	A || B	!A
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False

Operator Precedence Table

Precedence	Operation
1	( )
2	++ -- + - ! (type)
3	* / %
4	+ -
5	< <= > >=
6	== !=
7	&&
8	||
9	= += -= *= /= %=

Using integers in conditions

```
int  count = 100 ;
```

What are values of:

```
if (vote > 0) {
     /* protest */
Else {
     /* dissolve */
}
```

```
if (vote) {
     /* protest */
Else {
     /* dissolve */
}
```

`count`

`!count`

`!!count`

```
if (vote == 0) {
     /* dissolve */
Else {
     /* protest */
}
```

```
if (!vote) {
     /* dissolve */
Else {
     /* protest */
}
```

`!!!!count + 1`

Caution!

```c
int num;

printf("Enter an integer: ");
scanf("%d", &num);

if (num = 3) {
  printf("The value is 3.\n");
}
printf("num = %d\n", num);
```

- What is the result if user enters 7?

Short-Circuit Evaluation

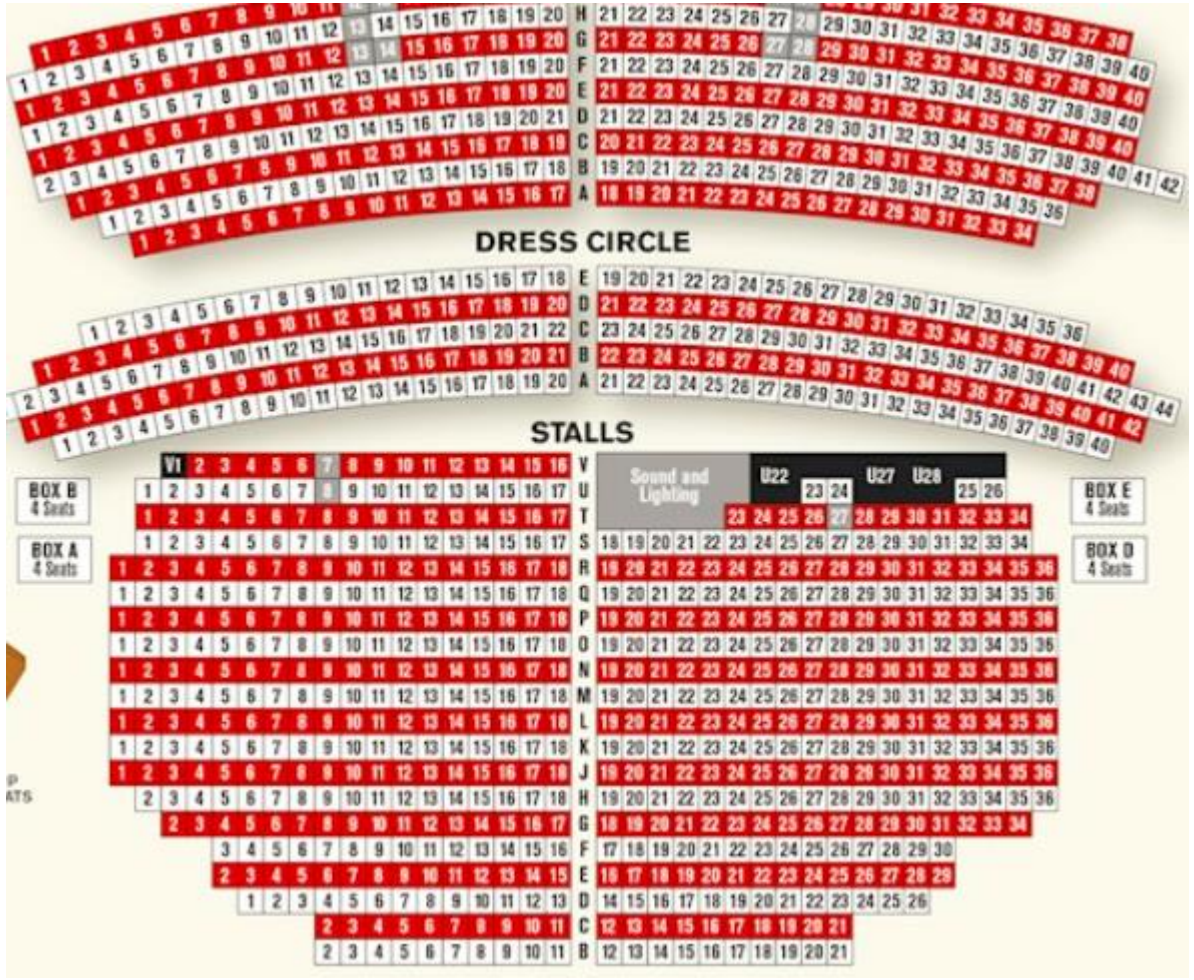
- Does the following code give an error if variable **a is zero**?

```
if ((a != 0) && (b/a > 3)) {
    printf(. . .);
}
```

A	B	A && B	A || B	!A
False	False	**False**	False	True
False	True	False	True	True
True	False	False	**True**	False
True	True	True	True	False

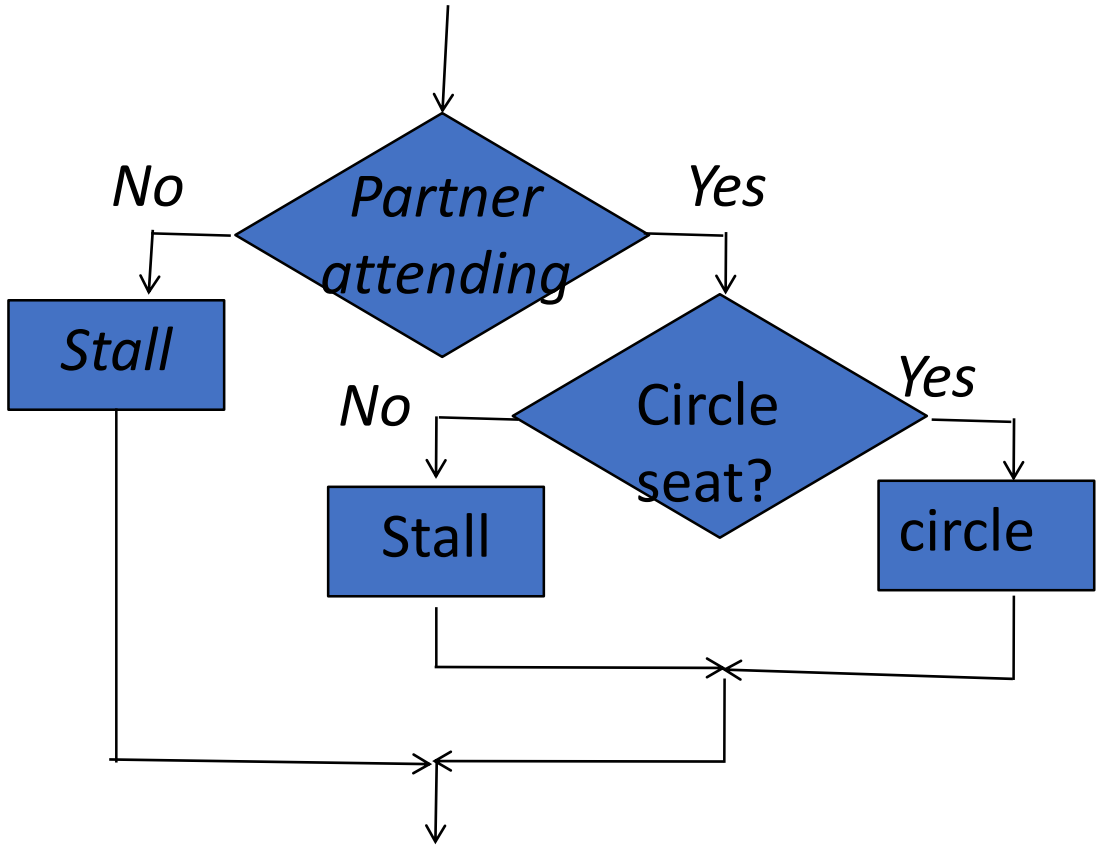
Nested if-else statements – Booking tickets

- Going to watch performance
- If your partner is attending, and circle seats are available, then book tickets for circle seat.
- If your partner is attending, and circle seats not available, then book tickets for stalls seat.
- If your partner is not attending, then book tickets for stalls seat.



Nested if-else statements – Booking tickets

- Going to watch performance
- If your partner is attending, and circle seats are available, then book tickets for circle seat.
- If your partner is attending, and circle seats not available, then book tickets for stalls seat.
- If your partner is not attending, then book tickets for stalls seat.



Nested if-else statements – Using { and }

- It's possible not to use "curly brackets" in writing conditional statements:

```
if (d <= 30)
   velocity = 0.425 + 0.00175*d*d ;
else
   velocity = 0.625 + 0.12*d – 0.0025*d*d ;
```

- But we **insist that you use "curly brackets"** in writing conditional statements for better clarity.

```
if (d <= 30) {
   velocity = 0.425 + 0.00175*d*d ;
} else {
   velocity = 0.625 + 0.12*d – 0.0025*d*d ;
}
```

Nested if-else statements

```
if (x > y)
    if (y < z)
        k++;
    else
        m++;
else
    j++;
```

When will k increase?

When will m increase?

When will j increase?

Nested if-else statements

```
if (x > y)
    if (y < z)
        k++;
else
    j++;
```

When will k increase?

When will j increase?

The C compiler will **associate an `else` statement with the closest `if` statement** within a block.

j will increase when (x > y) and (y >= z).

Nested if-else statements

```
if (x > y) {
    if (y < z) {
        k++;
    }
}
else {
    j++;
}
```

```
if (x > y) {
    if (y < z) {
        k++;
    }
    else {
    j++;
    }
}
```

Having curly brackets avoid confusion.

Caution! Without using Curly Brackets

```
int a = 3;
if (a > 10);
   printf("a is larger than 10\n");
printf("Next line.\n");
```

Let’s Run it!

Problem #5

Write a program that reads in a 6-digit zip code and uses its first digit to print the associated geographic area.

If zip code begins with	Print this message
0, 2 or 3	<zip code> is on the East Coast.
4 – 6	<zip code> is in the Central Plains.
7	<zip code> is in the South.
8 or 9	<zip code> is in the West.
others	<zip code> is invalid.

Problem #5

```
switch (zip/100000) {
  case 0: case 2: case 3:
    printf("%06d is on the East Coast.\n", zip);
    break;
  case 4: case 5: case 6:
    printf("%d is in the Central Plains.\n", zip);
    break;
  case 7:
    printf("%d is in the South.\n", zip);
    break;
  case 8: case 9:
    printf("%d is in the West.\n", zip);
    break;
  default:
    printf("%d is invalid.\n", zip);
} // end switch
```

The Switch Statement

```
switch ( <variable or expression> ) {
  case value1:
    Code to execute if <variable or expr> == value1
    break;
  case value2:
    Code to execute if <variable or expr> == value2
    break;
  ...
  default:
    Code to execute if <variable or expr> does not
    equal to the value of any of the cases above
    break;
}
```

The Switch Statement

```
switch ( <variable or expression> ) {
  case value1:
    Code to execute if <variable or expr> == value1
    break;
  case value2:
    Code to execute if <variable or expr> == value2
    break;
  ...
  default:
    Code to execute if <variable or expr> does not
    equal to the value of any of the cases above
    break;
}
```

- Restriction: Evaluated Value must be of **discrete type**
- Every case must end with a break statement
- Catch all situation

Recap:

```
switch (zip/100000) {
  case 0: case 2: case 3:
    printf("%06d is on the East Coast.\n", zip);
    break;
  case 4: case 5: case 6:
    printf("%d is in the Central Plains.\n", zip);
    break;
  case 7:
    printf("%d is in the South.\n", zip);
    break;
  case 8: case 9:
    printf("%d is in the West.\n", zip);
    break;
  default:
    printf("%d is invalid.\n", zip);
} // end switch
```

Summary

- Revision: Modular Design
- Better Understanding of Structure of C programs
  - Character Data
  - Standard Input and Output statements
- Making Choices
  - If statements and logical operations
  - If-else statements
  - Nested if statements
  - Switch statements