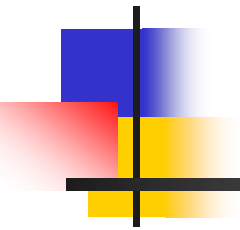
CS 1713
Introduction to Computer Programming II

Ch 1 – Overview – C programming Language

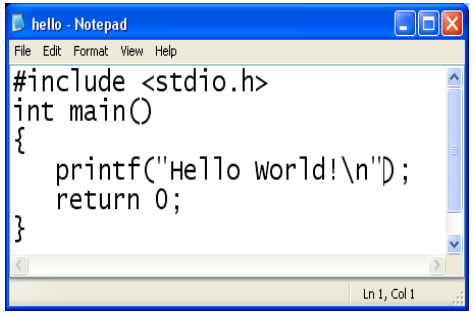


Turgay Korkmaz

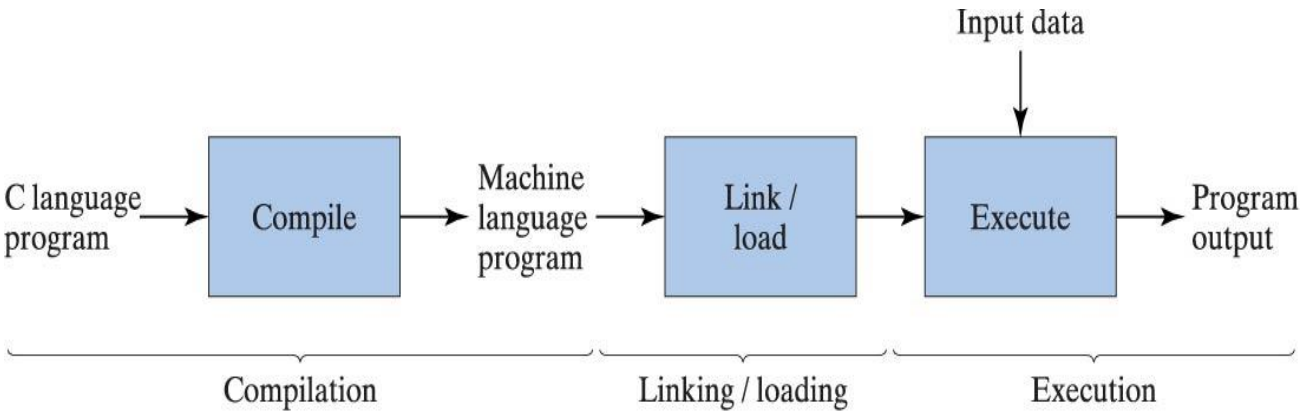
Office: SB 4.01.13
Phone: (210) 458-7346
Fax: (210) 458-4437
e-mail: korkmaz@cs.utsa.edu
web: www.cs.utsa.edu/~korkmaz

What is C?

- General purpose, machine-independent, high-level programming language
- Developed at Bell Labs in 1972 by Dennis Ritchie
- American National Standards Institute (ANSI) approved ANSI C standard in 1989



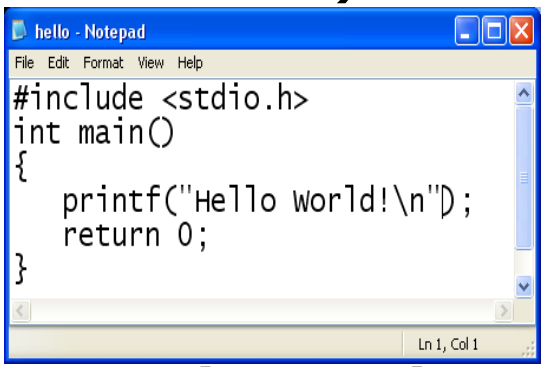
Source file



Hello World! in Linux

- Login to a linux machine
  - SSH Secure Shell (e.g., elk03.cs.utsa.edu)

```
elk03:> mkdir myprog
elk03:> cd myprog
elk03:> pico hello.c
```



- Type your program … and save it (ctrl-o)
- Compile and execute your program

```
elk03:> gcc hello.c –o hello
elk03:> hello
```

C Programming Language

- What is the best way to learn a language?
  - Look at sample programs
  - Read different books
  - Practice, practice, practice…
    - From web: "The absolute best way to immerse yourself quickly is to find a boyfriend or girlfriend who speaks the native language you are trying to learn"

Here is the first sample

1. Problem: generate a table showing the values of N^2 and 2^N for various values of N from 0 to 12
2. I/O: → program → N, N^2, 2^N
3. Hand example

N	N^2	2^N
0	0	1
1	1	2
2	4	4
3	9	8
4	16	16
5	25	32
6	36	64
…		
12	144	4096

4. Develop solution and **Coding**
5. Testing

```c
/*
 * File: powertab.c
 * ----------------
 * This program generates a table comparing values
 * of the functions n^2 and 2^n.
 */
#include <stdio.h>
#include "genlib.h"
/*
 * Constants
 * ---------
 * LowerLimit -- Starting value for the table
 * UpperLimit -- Final value for the table
 */
#define LowerLimit  0
#define UpperLimit 12

/* Private function prototypes */
static int RaiseIntToPower(int n, int k);
/* Main program */
main()
{
    int n;

    printf("    |   2 |    N\n");
    printf("  N |  N  |   2\n");
    printf("----+-----+------\n");
    for (n = LowerLimit; n <= UpperLimit; n++) {
        printf(" %2d | %3d | %4d\n", n,
               RaiseIntToPower(n, 2),
               RaiseIntToPower(2, n));
    }
}
/*
 * Function: RaiseIntToPower
 * Usage: p = RaiseIntToPower(n, k);
 * ---------------------------------
 * This function returns n to the kth power.
 */
static int RaiseIntToPower(int n, int k)
{
    int i, result;
    result = 1;
    for (i = 0; i < k; i++) {
        result *= n;
    }
    return (result);
}
```

```java
/*
 * File: powertab.java
 * -------------------
 * This program generates a table comparing values
 * of the functions n^2 and 2^n.
 */
import java.io.*;
public class powertab {
       /*
        * Constants
        * ---------
        * LowerLimit -- Starting value for the table
        * UpperLimit -- Final value for the table
        */
       public static final init LowerLimit = 0;
       public static final init UpperLimit = 12;



       /* Main program */
       public static main()
       {
           int n;

           System.out.println("    |   2 |    N");
           System.out.println("  N |  N  |   2");
           System.out.println("----+-----+------");
           for (n = LowerLimit; n <= UpperLimit; n++) {
             System.out.format(" %2d | %3d | %4d\n", n,
                      RaiseIntToPower(n, 2),
                      RaiseIntToPower(2, n));
           }
       }
       /*
        * Function: RaiseIntToPower
        * Usage: p = RaiseIntToPower(n, k);
        * ---------------------------------
        * This function returns n to the kth power.
        */
       private static int RaiseIntToPower(int n, int k)
       {
           int i, result;
           result = 1;
           for (i = 0; i < k; i++) {
               result *= n;
           }
           return (result);
       }
}
```

Structure of a C Program

Comments,
Preprocessor (library inclusion, program level symbolic definitions)
function prototypes
main function
{
  Variable declarations,
  Statements (must end with ;)
}
Other user-defined functions

```
/*
 * File: powertab.c
 * ----------------
 * This program generates a table comparing
 * values of the functions n^2 and 2^n.
 */

#include <stdio.h>
#include "genlib.h"

/*
 * Constants
 * ---------
 * LowerLimit -- Starting value for the table
 * UpperLimit -- Final value for the table
 */
#define LowerLimit  0
#define UpperLimit 12

/* Private function prototypes */
static int RaiseIntToPower(int n, int k);
```

```
/* Main program */
main()
{
    int n;

    printf("    |   2 |    N\n");
    printf("  N |  N  |   2\n");
    printf("----+-----+------\n");
    for (n = LowerLimit; n <= UpperLimit; n++) {
        printf(" %2d | %3d | %4d\n", n,
               RaiseIntToPower(n, 2),
               RaiseIntToPower(2, n));
    }
}
/*
 * Function: RaiseIntToPower
 * Usage: p = RaiseIntToPower(n, k);
 * ---------------------------------
 * This function returns n to the kth power.
 */
static int RaiseIntToPower(int n, int k)
{
    int i, result;
    result = 1;
    for (i = 0; i < k; i++) {
        result *= n;
    }
    return (result);
}
```

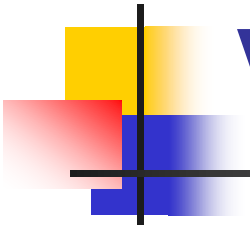
General Form

```
/* comments */

preprocessing
directives

int main(void)
{

      declarations

      statements

}
```

- The main function contains two types of commands:
  - ✓ declarations
  - ✓ statements
- Declarations and statements must end with a semicolon ;
- Preprocessor directives do not end with a semicolon ;
- To exit the program, use a **return 0**; statement

Another Simple Program

```
/*****************************************************/
/*  This program computes the sum of two numbers   */
/*****************************************************/
#include <stdio.h>
int main(void)
{
    /*  Declare and initialize variables.  */
    double number1 = 473.91, number2 = 45.7, sum;
    /*  Calculate sum.  */
    sum = number1 + number2;
    /*  Print the sum.  */
    printf("The sum is %5.2f \n", sum);

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



Variables

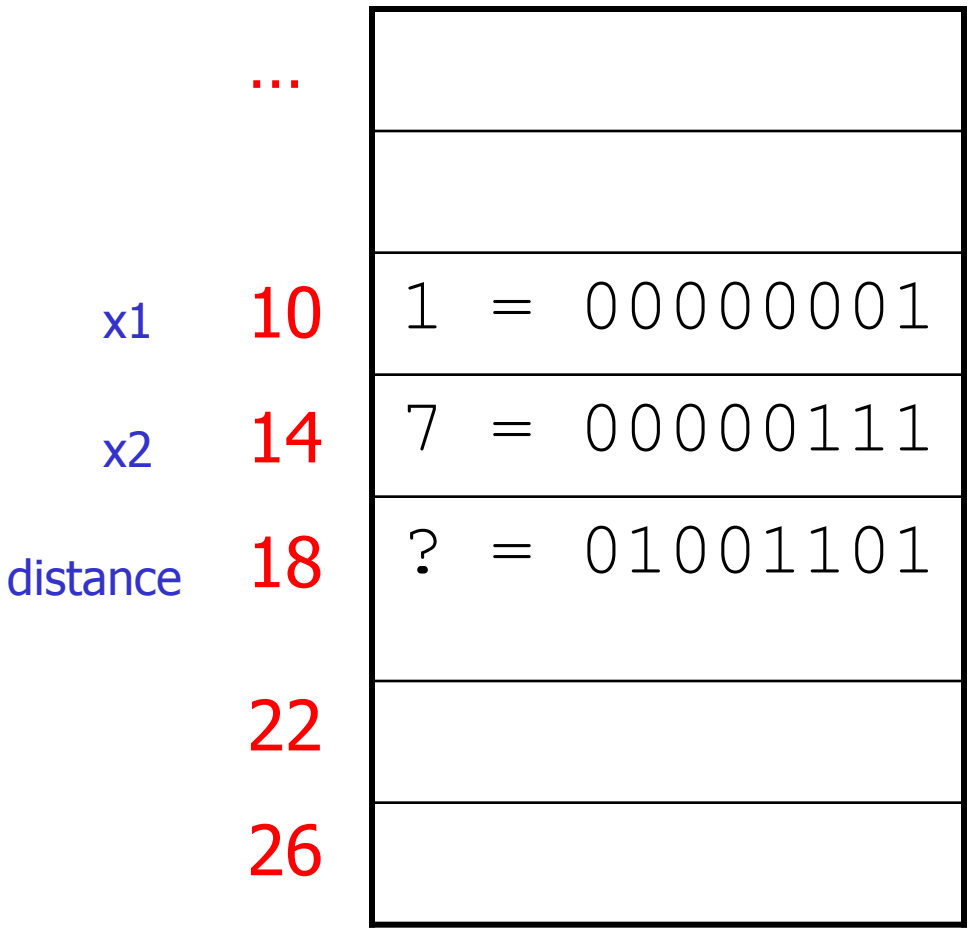
- What is a variable in math?

  $f(x) = x^2+x+4$

- In C,
  - An identifier or variable name is used to reference a memory location that holds a data value
  - A variable must be declared before it is used. `type name_list;` (lifetime - scope)
  - `int a, b;`

Memory

```
int x1=1,x2=7,distance;
```



How many memory cells does your computer have?

Say it says 2Gbyte memory?

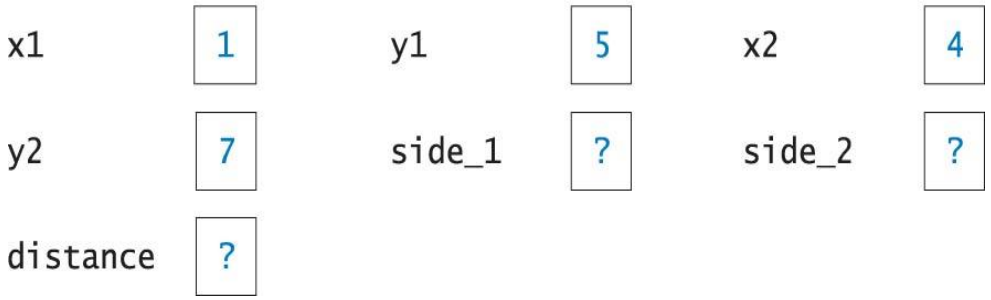
$1K=10^3$ or $2^{10} = 1024$

$1M=10^6$ or $2^{20} = 1024^2$

$1G=10^9$ or $2^{30} = 1024^3$

Memory Snapshot

```
double x1=1, y1=5, x2=4, y2=7,
       side_1, side_2, distance;
```



Name	Addr	Content
x1		1
y1		5
x2		4
y2		7
side_1		?
side_2		?
distance		?

Rules for selecting a valid identifier (variable name)

- Begin with an alphabetic character or underscore (e.g., abcABC_)
- Use only letters, digits and underscore (no special characters ^%@)
- Case sensitive (AbC, aBc are different)
- Cannot use C keywords

auto	double	ints	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Are the following valid identifiers?

distance

1x

x_1

initial_time

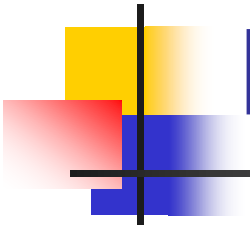
DisTaNce

X&Y

rate%

x_sum

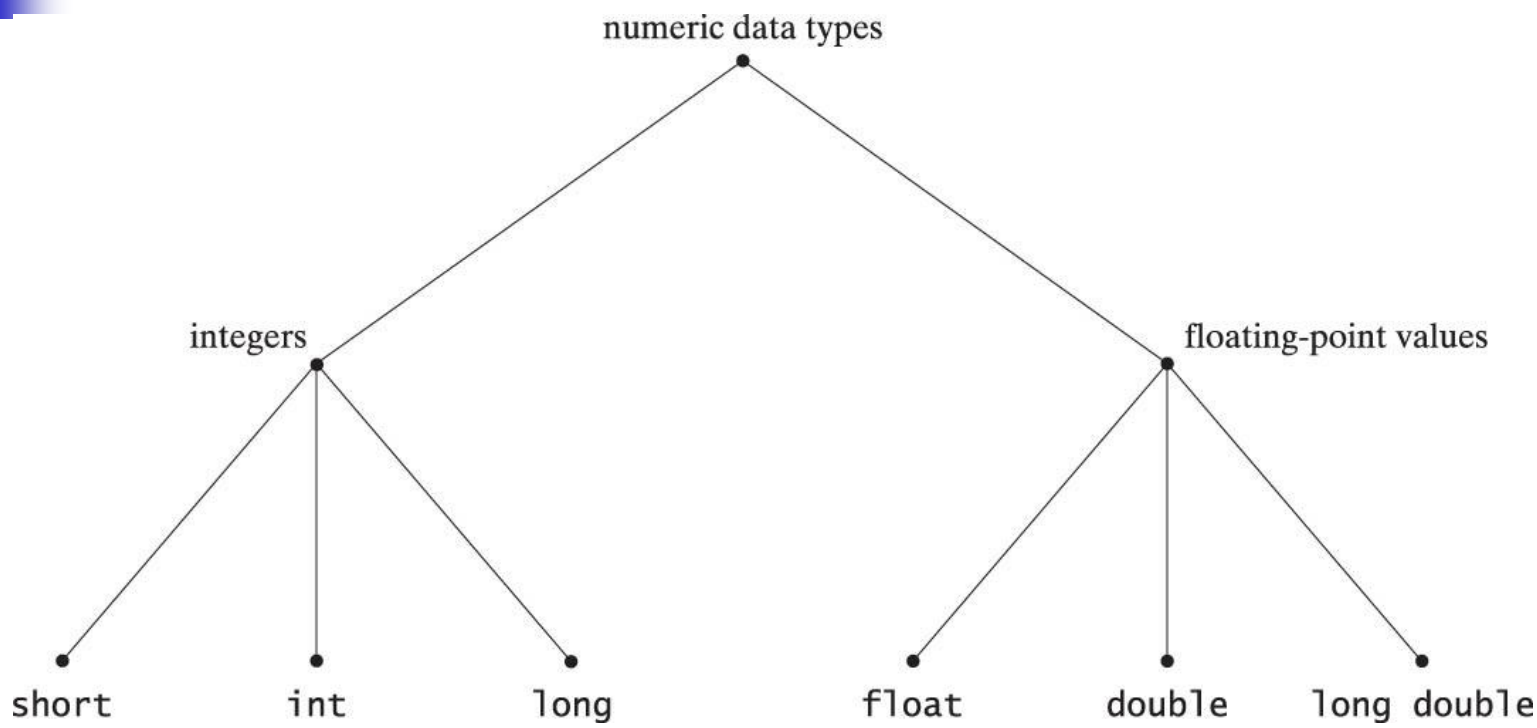
switch



Local and Global Variables

- *Local* scope
  - a local variable is defined within a function or a block and can be accessed only within the function or block that defines it
- *Global* scope
  - a global variable is defined outside the **main** function and can be accessed by any function within the program file.

C Numeric Data Types



Example Data-Type Limits

Integers	
`short`	Maximum = 32,767
`int`	Maximum = 2,147,483,647
`long`	Maximum = 2,147,483,647
Floating Point	
`float`	6 digits of precision Maximum exponent 38 Maximum value 3.402823e+38
`double`	15 digits of precision Maximum exponent 308 Maximum value 1.797693e+308
`long double`	15 digits of precision Maximum exponent 308 Maximum value 1.797693e+308

*Microsoft Visual C++ 6.0 compiler.

C Character Data Type: `char`

```
char result ='Y';
```

In memory, everything is stored as binary value, which can be interpreted as char or integer. Examples of ASCII Codes

Character	ASCII Code	Integer Equivalent
newline, \n	0001010	10
%	0100101	37
3	0110011	51
A	1000001	65
a	1100001	97
b	1100010	98
c	1100011	99

Memory

How to represent ‘a’ ?

char My_letter=‘a’;

int My_number = 97

Always we have 1’s and 0’s in the memory. It depends on how you look at it?

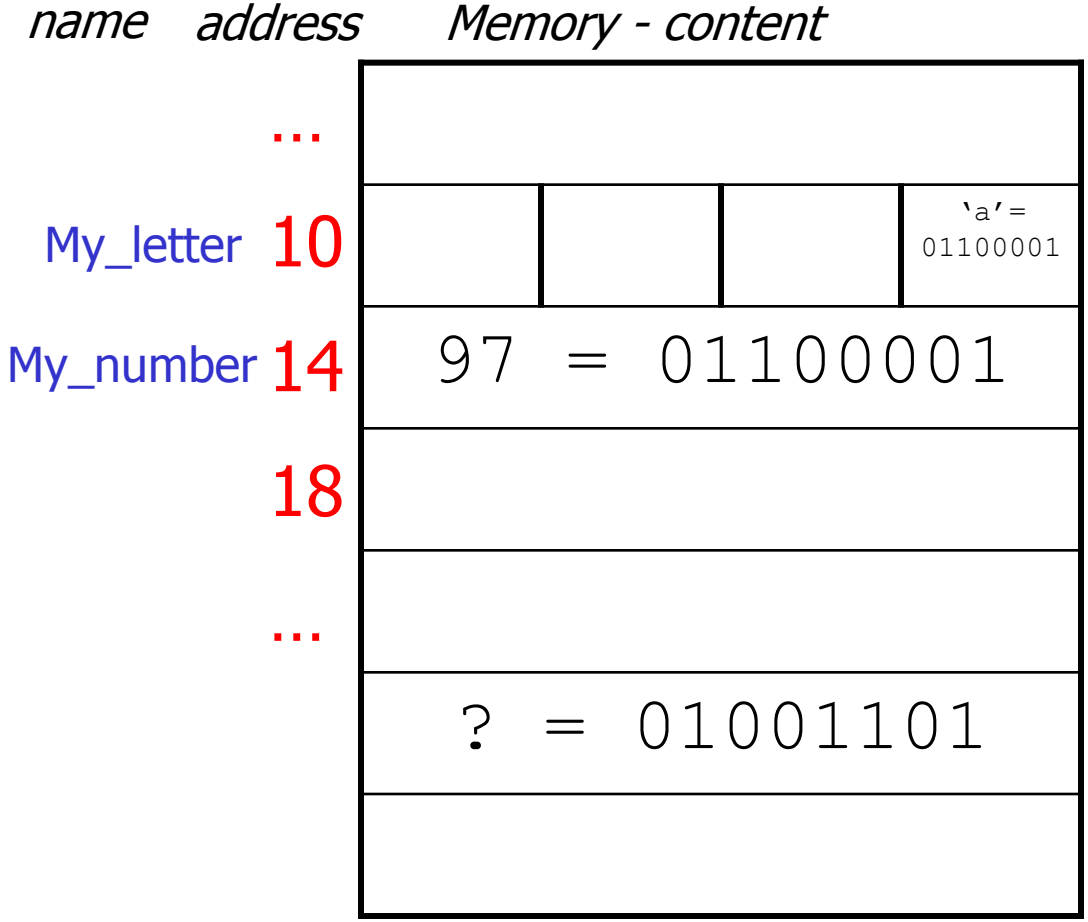
For example, 01100001 is

97 if you look at it as int, or

‘a’ if you look at it as char

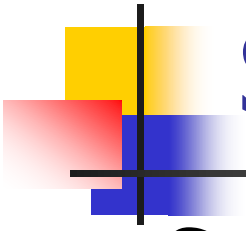
‘3’ is not the same as 3

How to represent 2.5?



Program to Print Values as Characters and Integers

```
/*-------------------------------------------------------------*/
/*  Program chapter2_1                                         */
/*                                                             */
/*  This program prints two values                             */
/*  as characters and integers.                                */

#include <stdio.h>

int main(void)
{
   /*  Declare and initialize variables.  */
   char ch='a';
   int i=97;

   /*  Print both values as characters.  */
   printf("value of ch: %c; value of i: %c \n",ch,i);

   /*  Print both values as integers.  */
   printf("value of ch: %i; value of i: %i \n",ch,i);

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



String

- Sequence of characters or array of characters

  `"this is a string"`

- There is no explicit string type in C, but
  - But some books define `string` type using `typedef char *string;`

Boolean type

- TRUE or FALSE
- There is no explicit Boolean type in C
  - Zero means FALSE
  - Other than zero means TRUE
- But, some books define `bool` type using

```
typedef int bool; /* OR */
typedef enum {FALSE,TRUE} bool;
```

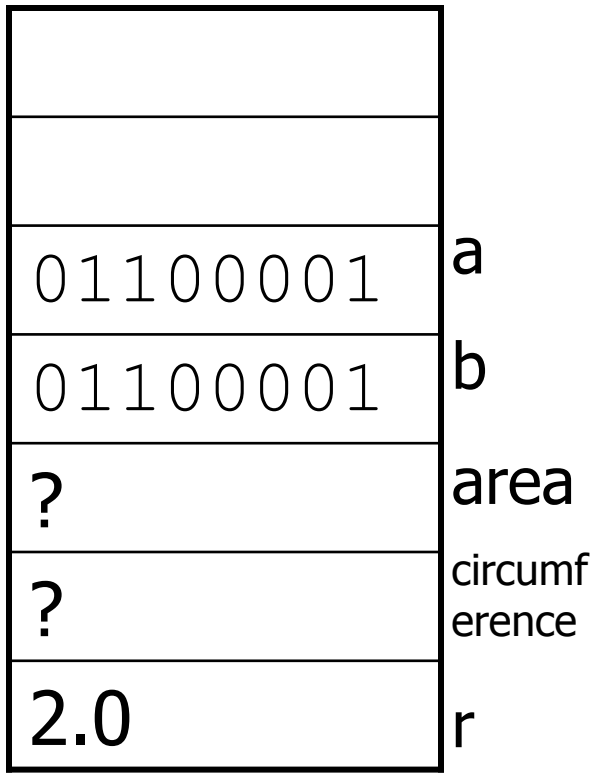
Constants

- A constant is a specific value that we use in our programs. For example

```
3.14,  97, 'a', or "hello"
```

- In your program,

```
int  a = 97;
char b ='a';
double area, r=2.0;
double circumference;
area = 3.14 * r*r;
circumference = 2 * 3.14 * r;
```

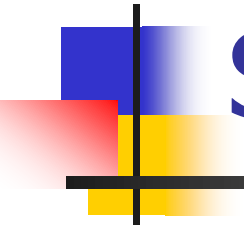


Symbolic Constants

- What if you want to use a better estimate of π?
  For example, you want 3.141593 instead of 3.14.
- You need to replace all by hand ☹
- Better solution, define π as a symbolic constant, e.g.

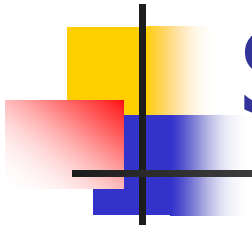
```
#define PI 3.141593
…
area = PI * r * r;
circumference = 2 * PI * r;
```

- Defined with a preprocessor directive
- Compiler replaces each occurrence of the directive identifier with the constant value in all statements that *follow* the directive



Simple I/O

- Recall the program computing distance between two points.
- How can we compute the distance between different points?



Simple Input and Output

- There is a standard I/O library for accepting input from user and displaying results on the screen:
  - `printf()`,
  - `scanf()`

Standard Input and Output

- Output: `printf` System.out.format(…)
- Input: `scanf`
  - Remember the program computing the distance between two points!

```
/*  Declare and initialize variables.  */
  double x1=1, y1=5, x2=4, y2=7,
         side_1, side_2, distance;
```

  - How can we compute distance for different points?
  - It would be better to get new points from user, right? For this we will use `scanf`
- To use these functions, we need to use

```
#include <stdio.h>
```

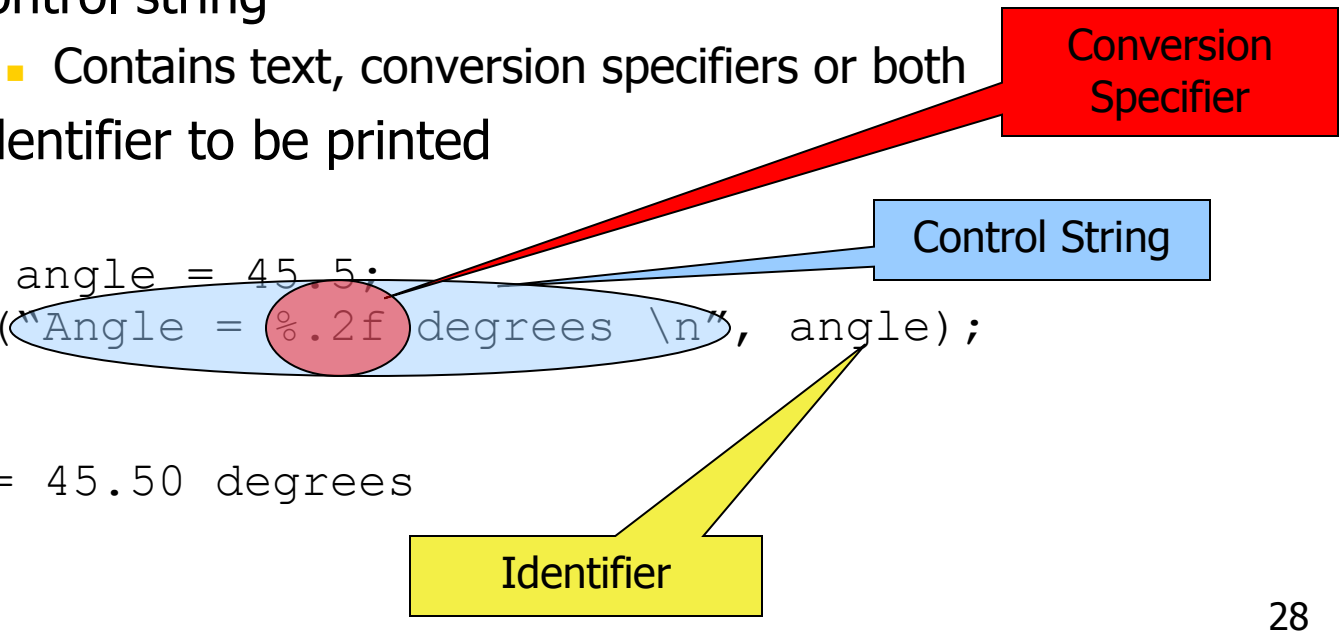
Standard Output

- **printf** Function
  - prints information to the screen
  - requires two arguments
    - control string
      - Contains text, conversion specifiers or both
    - Identifier to be printed
- Example

```
double angle = 45.5;
printf("Angle = %.2f degrees \n", angle);
```

Output:

```
Angle = 45.50 degrees
```



Conversion Specifiers for Output Statements

Variable Type	Output Type	Specifier
	Integer Values	
short, int	int	%i, %d
int	short	%hi, %hd
long	long	%li, %ld
int	unsigned int	%u
int	unsigned short	%hu
long	unsigned long	%lu
	Floating-Point Values	
float, double	double	%f, %e, %E, %g, %G
long double	long double	%LF, %Le, %LE, %Lg, %LG
	Character Values	
char	char	%c

Frequently Used

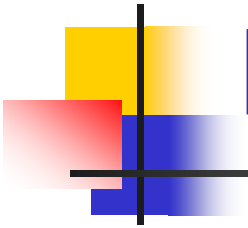
Standard Output

Output of -145

Specifier	Value Printed
%i	-145
%4d	-145
%3i	-145
%6i	__-145
%-6i	-145__
%8i	_____-145
%-8i	-145____

Output of 157.8926

Specifier	Value Printed
%f	157.892600
%6.2f	157.89
%7.3f	**157.893**
%7.4f	157.8926
%7.5f	157.89260
%e	1.578926e+02
%.3E	1.579E+02



Exercise

```
int sum = 65;
double average = 12.368;
char ch = 'b';
```

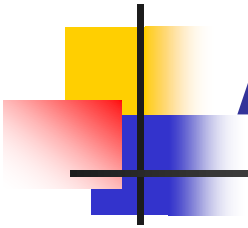
Show the output line (or lines) generated by the following statements.

```
printf("Sum = %5i; Average = %7.1f  \n", sum, average);
printf("Sum = %4i \n Average = %8.4f \n", sum, average);
printf("Sum and Average \n\n %d %.1f \n", sum, average);
printf("Character is %c; Sum is %c \n", ch, sum);
printf("Character is %i; Sum is %i \n", ch, sum);
```

Exercise (cont'd)

- Solution

```
Sum =     65; Average =     12.4
Sum =    65
 Average =  12.3680
Sum and Average


 65 12.4
Character is b; Sum is A
Character is 98; Sum is 65
```



A useful feature of printf

```
printf("%5d\n", value);
```

We can have a more general form of this as

```
printf("%*d\n", fieldWidth, value);
```

Standard Input

- **scanf** Function
  - inputs values from the keyboard
  - required arguments
    - control string
    - memory locations that correspond to the specifiers in the control string
- Example:

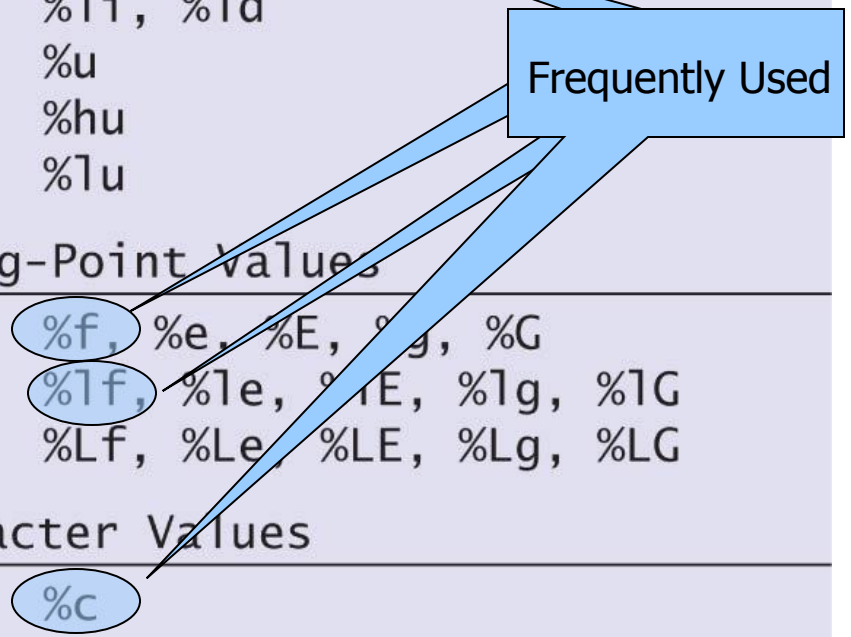
```
double distance;
char unit_length;
scanf("%lf %c", &distance, &unit_length);
```

- *It is very important to use a specifier that is appropriate for the data type of the variable*

What will happen if you forget **&** before the variable name?

Conversion Specifiers for Input Statements

Variable Type	Specifier
Integer Values	
int	%i, %d
short	%hi, %hd
long int	%li, %ld
unsigned int	%u
unsigned short	%hu
unsigned long	%lu
Floating-Point Values	
float	%f, %e, %E, %g, %G
double	%lf, %le, %lE, %lg, %lG
long double	%Lf, %Le, %LE, %Lg, %LG
Character Values	
char	%c



Exercise

```
/* if you use  the
   standard library */

#include <stdio.h>
…

float f;
int i;

scanf("%f %d", &f, &i);
```

```
/* if you use some
   textbooks' library */

#include "genlib.h"
#include "simpio.h"
…
float f;
int i;

f=GetReal();
i=GetInteger();
```

- What will be the values stored in `f` and `i` after scanf statement if following values are entered

```
12.5 1
12      45
12   23.2
12.1         10
12
1
```

Good practice

- You don’t need to have a printf before scanf, but it is good to let user know what to enter:

```
printf(“Enter x y : ”);
scanf(“%d %d”, &x, &y);
```

- Otherwise, user will not know what to do!

Exercise: How to input two points without re-compiling the program

```
/*-------------------------------------------------------------*/
/*  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;
}
/*-------------------------------------------------------------*/
```

```
/*if you use the stdio.h library*/

printf("enter x1 y1: ");

scanf("%lf %lf", &x1, &y1);

printf("enter x2 y2: ");

scanf("%lf %lf", &x2, &y2);
```

```
/*if you use the textbook's library */
#include "genlib.h"
#include "simpio.h"

printf("enter x1: "); x1=GetReal();

printf("enter y1: "); y1=GetReal();

printf("enter x2: "); x2=GetReal();

printf("enter y2: "); y2=GetReal();
```

C Programming Language: **Expressions**

- Often we need to compute some math formulas/expressions…
- C expressions composed of terms (variables) and operators (+-*/%) are very similar to the ones in math,
- So you can easily transform one to another

Assignment Statements

- Used to assign a value to a variable
- General Form:

identifier = expression;

/* '=' means assign **expression** to **identifier** */

- Example 1

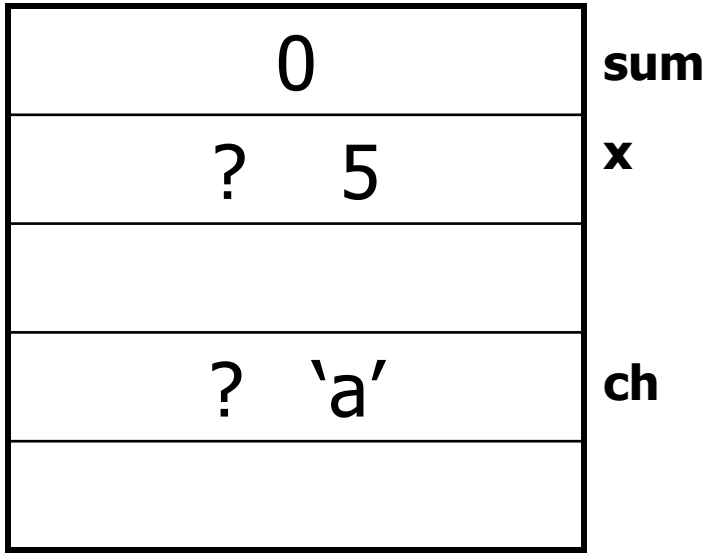
  double sum = 0;

- Example 2

  int x;
  x=5;

- Example 3

  char ch;
  ch = 'a';



Assignment examples (cont'd)

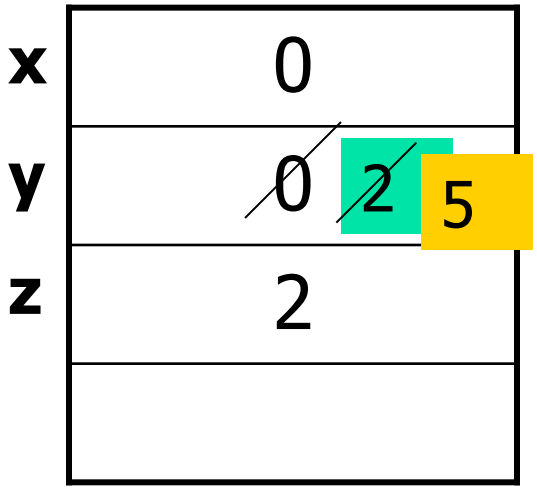
- Example 3

int x, y, z;

x = y = 0;

right to left!

Z = 1+1;



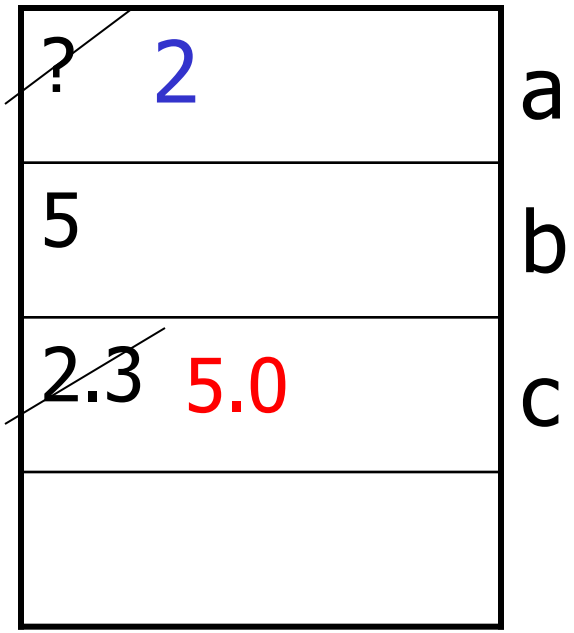
- Example 4

y=z;

y=5;

Assignment examples with different types

```
int a, b=5;
double c=2.3;
…
a=c; /* data loss */
c=b; /* no data loss */
```



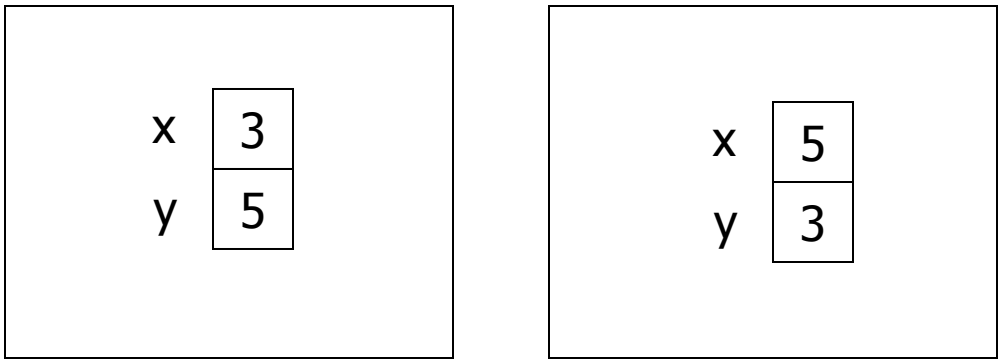
long double, double, float, long integer, integer, short integer, char

→ Data may be lost. Be careful!

← No data loss

Exercise: swap

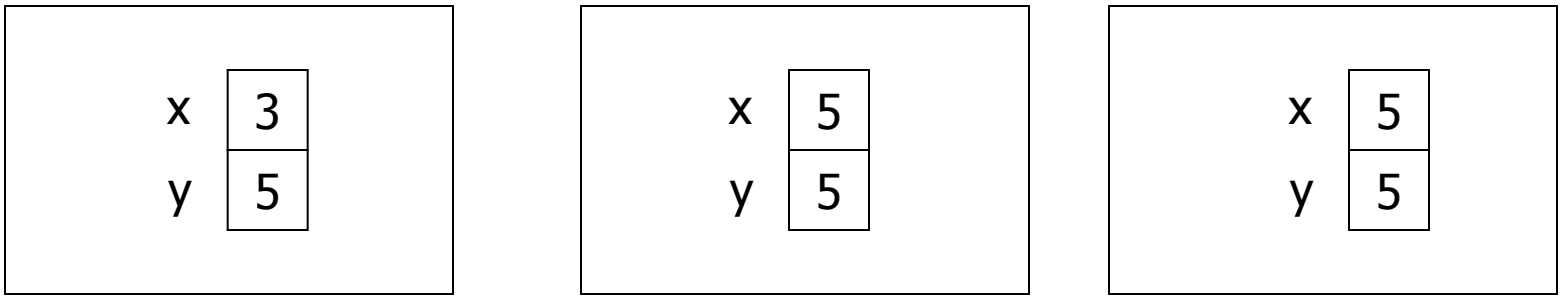
- Write a set of statements that swaps the contents of variables x and y



Exercise: swap

First Attempt

```
x=y;
y=x;
```

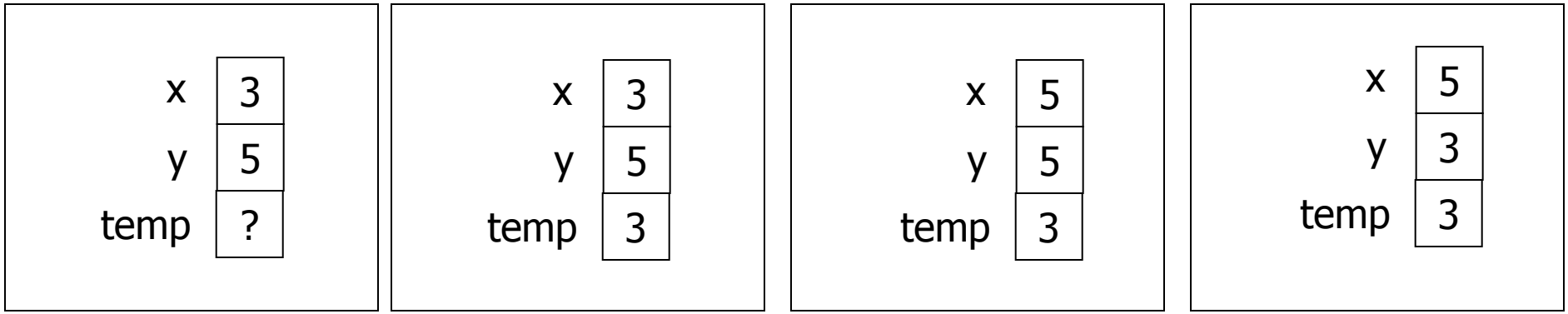


Before | After x=y | After y=x

Exercise: swap

Solution

```
temp= x;
x=y;
y=temp;
```



Before | after temp=x | after x=y | after y = temp

Will the following solution work, too?

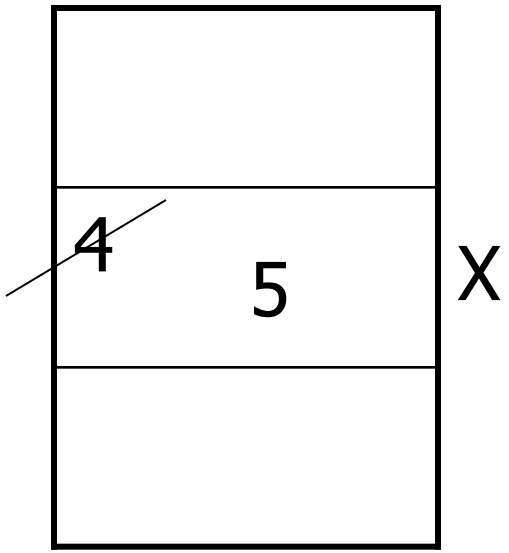
```
temp= y;
y=x;
x=temp;
```

Arithmetic Operators

- Addition + sum = num1 + num2;
- Subtraction - age = 2007 – my_birth_year;
- Multiplication * area = side1 * side2;
- Division / avg = total / number;
- Modulus % lastdigit = num % 10;
  - Modulus returns remainder of division between two *integers*
  - Example 5%2 returns a value of 1
- Binary vs. Unary operators
  - All the above operators are binary (why)
  - - is an unary operator, e.g., a = -3 * -4

Arithmetic Operators (cont'd)

- Note that 'id = exp' means assign the result of exp to id, so
- X=X+1 means
  - first perform X+1 and
  - Assign the result to X
- Suppose X is 4, and
- We execute X=X+1



Integer division vs Real division

- Division between two integers results in an integer.
- The result is truncated, not rounded
- Example:

  int A=5/3; → A will have the value of 1

  int B=3/6; → B will have the value of 0
- To have floating point values:

  double X=5.0/3; → X will have the value of 1.666

  double Y=3.0/6.0; → Y will have the value of 0.5
- Type Cast

  X = **(double)** 5 /3; X will have the value of 1.666

Implement a program that computes/prints simple arithmetic operations

Declare a=2, b=5, c=7, d as int

Declare x=5.0, y=3.0, z=7.0, w as double

d = c%a	Print d
d = c/a	Print d
w = z/x	Print w
d = z/x	Print d
w = c/a	Print w
a=a+1	Print a

… try other arithmetic operations too..

Mixed operations and Precedence of Arithmetic Operators

int a=4+6/3*2; → a=? a= 4+2*2 = 4+4 = 8

int b=(4+6)/3*2; → b=? b= 10/3*2 = 3*2= 6

Precedence	Operator	Associativity
1	Parentheses: ( )	Innermost first
2	Unary operators: + − (type)	Right to left
3	Binary operators: * / %	Left to right
4	Binary operators: + −	Left to right
5	assign =	Right to left

Extend the previous program to compute/print mixed arithmetic operations

Declare a=2, b=5, c=7, d as int
Declare x=5.0, y=3.0, z=7.0, w as double

d = a+c%a	Print d
d = b*c/a	Print d
w = y*z/x+b	Print w
d = z/x/y*a	Print d
w = c/(a+c)/b	Print w
a=a+1+b/3	Print a

… try other arithmetic operations too..

Increment and Decrement Operators

- Increment Operator ++
  - post increment x++;
  - pre increment ++x;

  } x=x+1;
- Decrement Operator --
  - post decrement x--;
  - pre decrement --x;

  } x=x-1;

But, the difference is in the following example. Suppose x=10;

A = x++ - 5; means A=x-5; x=x+1; so, A= 5 and x=11

B =++x - 5; means x=x+1; B=x-5; so, B=6 and x=11

Abbreviated Assignment Operator

operator	example	equivalent statement
+=	x+=2;	x=x+2;
-=	x-=2;	x=x-2;
*=	x*=y;	x=x*y;
/=	x/=y;	x=x/y;
%=	x%=y;	x=x%y;

!!! x *= 4+2/3 → x = x*4+2/3 wrong

x=x*(4+2/3) correct

Precedence of Arithmetic Operators (updated)

Precedence	Operator	Associativity
1	Parentheses: ( )	Innermost first
2	Unary operators: + - ++ -- (type)	Right to left
3	Binary operators: * / %	Left to right
4	Binary operators: + -	Left to right
5	Assignment operators: = += -= *= /= %=	Right to left

Write a C statement for a given MATH formula/expression

- Area of trapezoid

$$area = \frac{base*(height_1 + height_2)}{2}$$

```
area = base*(height1 + height2)/2;
```

- How about this

$$Tension = \frac{2m_1m_2}{m_1 + m_2} \times g$$

Exercise

$$Tension = \frac{2m_1m_2}{m_1 + m_2} \times g$$

Tension = 2*m1*m2 / m1 + m2 * g; wrong

Tension = 2*m1*m2 / (m1 + m2) * g

- Write a C statement to compute the following

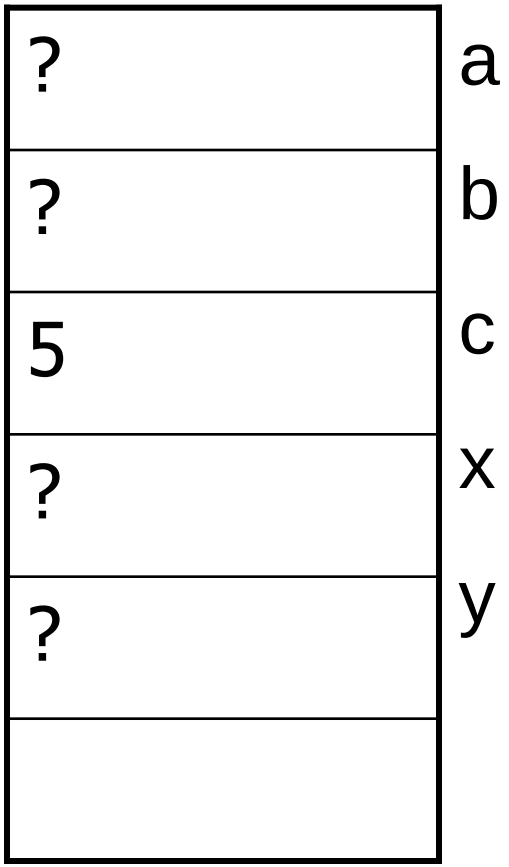
$$f = \frac{x^3 - 2x^2 + x - 6.3}{x^2 + 0.05x + 3.14}$$

f = (x*x*x-2*x*x+x-6.3)/(x*x+0.05*x+3.14);

Exercise: Arithmetic operations

- Show the memory snapshot after the following operations by hand

```
int a, b, c=5;
double x, y;
a = c * 2.5;
b = a % c * 2 - 1;
x = (5 + c) * 2.5;
y = x - (-3 * a) / 2;
```



Write a C program and print out the values of a, b, c, x, y and compare them with the ones that you determined by hand.

a = 12 b = 3 c= 5 x = 25.0000 y = 43.0000

Exercise: Arithmetic operations

- Show how C will perform the following statements and what will be the final output?

```
int a =  6, b = -3, c =  2;
c= a - b * (a + c * 2) + a / 2 * b;
printf("Value of c = %d \n", c);
```

Step-by-step show how C will perform the operations

c = 6 - -3 * (6 + **2 * 2**) + 6 / 2 * -3;
c = 6 - -3 * (6 + 4) + 3 * -3
c = 6 - -3 *10 + -9
c = 6 - -30 + -9
c = 36 + -9
c = 27

- output:

  Value of c = 27

Step-by-step show how C will perform the operations

```
int a = 8, b = 10, c = 4;

c = a % 5 / 2 + -b / (3 – c) * 4 + a / 2 * b;

printf("New value of c is %d \n", c);
```

Programming exercise

- Write a C program that asks user to enter values for the double variables (a, b, c, d) in the following formula. It then computes the result (res) and prints it with three digits after .

$$res = \frac{a+b}{c-d} + \frac{\sqrt{a+c}}{a-b}\,\frac{c+b}{a+c}$$

Exercise: reverse a number

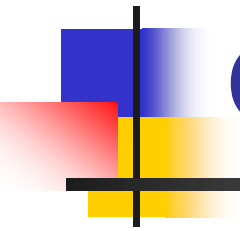
- Suppose you are given a number in the range [100 999]
- Write a program to reverse it
- For example,

  num is 258

  reverse is 852

```
int d1, d2, d3, num=258, reverse;
d1 = num / 100;
d2 = num % 100 / 10;
d3 = num % 10;
reverse = d3*100 + d2*10 + d1;
printf("reverse is %d\n", reverse);
```

```
d1 = num / 100;
d3 = num % 10;
reverse = num – (d1*100+d3) +
          d3*100 + d1;
```

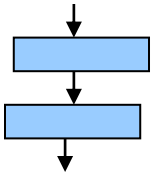


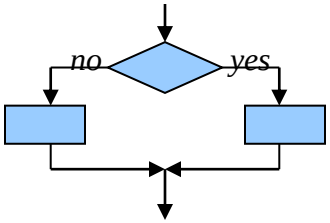
C Programming Language: **Control Structures/Flow**

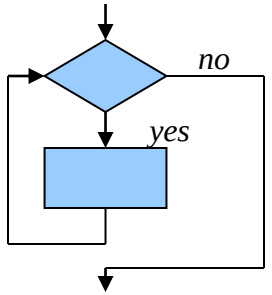
- So far, we considered very simple programs (read, compute, print)
- How can we deal with real-world problems involving conditions, selections, repetitions?

Algorithm Development

- Use simple control structures to organize the solution to a problem
  - Sequence
  - Selection
  - Repetition

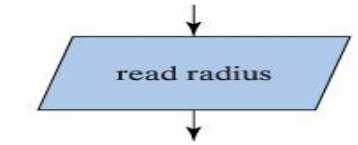
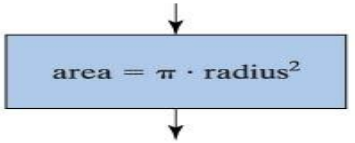
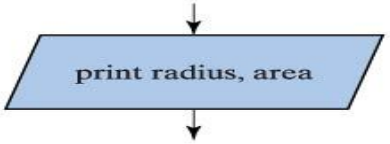
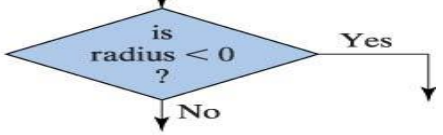
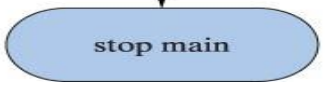


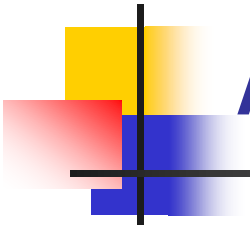




- Refinement with Pseudo-code (English like statements) and Flowchart (diagram, graph)

Pseudo-code Notation and Flowchart Symbols

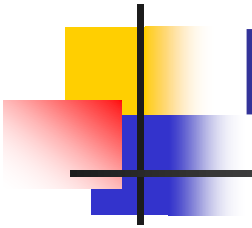
Basic Operation	Pseudocode Notation	Flowchart Symbol
Input	read radius	
Computation	set area to $\pi \cdot \text{radius}^2$	
Output	print radius, area	
Comparisons	if radius < 0 then ...	
Beginning of algorithm	main:	
End of algorithm		



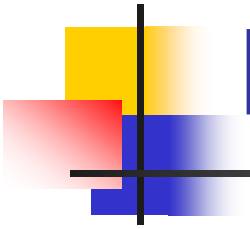
A few notes before we start…

- Evaluate alternative solutions
  - A problem can be solved in many different ways
  - Which is the best (e.g., faster, less memory)
- Check error conditions
  - Do not trust user! Check the data. A=b/c;
  - Be clear about specifications
- Generate a lot of (smart) Test Data
  - Test each of the error conditions
  - Program validation and verification
  - Program walkthrough

Condition (Boolean/Logic) Expressions



- Selection and repetition structures use conditions, so we will first discuss them
- A **condition** is an expression (e.g., a > b) that can be evaluated to be
  - TRUE (any value > 0) or
  - FALSE (value of 0)
- Conditional Expression is composed of expressions combined with relational and/or logical operators



Relational Operators

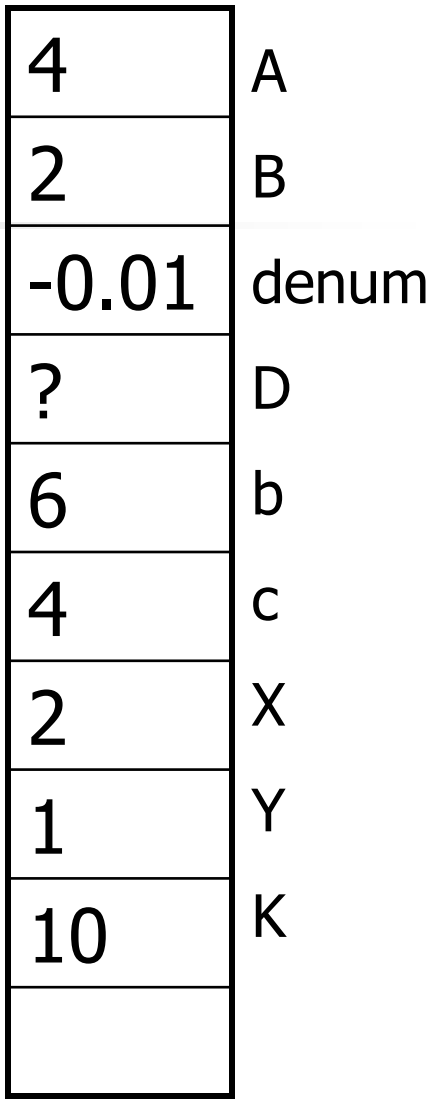
==	equality	(x == 3)
!=	non equality	(y != 0)
<	less than	(x < y)
>	greater than	(y > 10)
<=	less than equal to	(x <= 0)
>=	greater than equal to	(x >= y)

!!! a==b vs. a=b !!!

Examples

- A < B
- fabs(denum) < 0.0001
- D = b > c;
- if (D)

  A=b+c;
- Mixing with arithmetic op
  - X+Y >= K/3



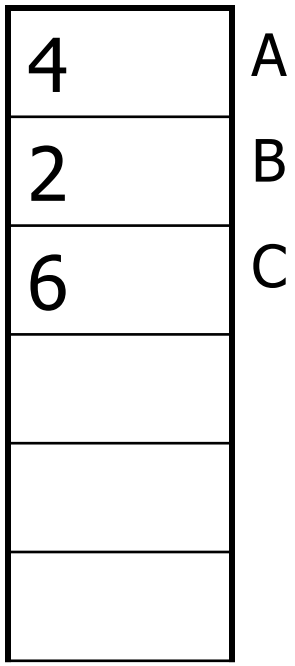
Logical Operators

- ! not !(x==0)
- && and (x>=0) && (x<=10)
- || or (x>0) || (x<0)

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

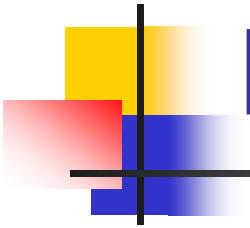
Examples

- A<B && C>=5
- A+B * 2 < 5 && 4>=A/2
- A<B || C<B && A-2 < 10
- A < B < C ????
- A<B<C is not the same as
  - (A<B) && (B<C)



Precedence for Arithmetic, Relational, and Logical Operators

Precedence	Operation	Associativity
1	( )	Innermost first
2	++ -- + - ! (type)	Right to left (unary)
3	* / %	Left to right
4	+ -	Left to right
5	< <= > >=	Left to right
6	== !=	Left to right
7	&&	Left to right
8	||	Left to right
9	= += -= *= /= %=	Right to left



Exercise

- Assume that following variables are declared

  a = 5.5 b = 1.5 k = -3

- Are the following true or false

  a < 10.0 + k

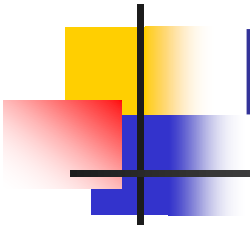
  a + b >= 6.5

  k != a-b

  !(a == 3*b)

  a<10 && a>5

  fabs(k)>3 || k<b-a

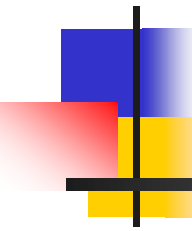


Bitwise Operators

- & bitwise AND
- | bitwise inclusive OR
- ^ bitwise exclusive OR
- << left shift
- >> right shift
- ~ one’s complement

Operator	Description	Associativity
()	Parentheses (function call) (see Note 1)	left-to-right
[]	Brackets (array subscript)	
.	Member selection via object name	
->	Member selection via pointer	
++ --	Postfix increment/decrement (see Note 2)	
++ --	Prefix increment/decrement (see Note 2)	**right-to-left**
+ -	Unary plus/minus	
! ~	Logical negation/bitwise complement	
(type)	Cast (change type)	
*	Dereference	
&	Address	
sizeof	Determine size in bytes	
* / %	Multiplication/division/modulus	left-to-right
+ -	Addition/subtraction	left-to-right
<< >>	Bitwise shift left, Bitwise shift right	left-to-right
< <=	Relational less than/less than or equal to	left-to-right
> >=	Relational greater than/greater than or equal to	
== !=	Relational is equal to/is not equal to	left-to-right
&	Bitwise AND	left-to-right
^	Bitwise exclusive OR	left-to-right
|	Bitwise inclusive OR	left-to-right
&&	Logical AND	left-to-right
||	Logical OR	left-to-right
?:	Ternary conditional	**right-to-left**
=	Assignment	**right-to-left**
+= -=	Addition/subtraction assignment	
*= /=	Multiplication/division assignment	
%= &=	Modulus/bitwise AND assignment	
^= |=	Bitwise exclusive/inclusive OR assignment	
<<= >>=	Bitwise shift left/right assignment	
,	Comma (separate expressions)	left-to-right

Note 1: Parentheses are also used to group sub-expressions to force a different precedence; such parenthetical expressions can be nested and are evaluated from inner to outer. **Note 2**: Postfix increment/decrement have high precedence, but the actual increment or decrement of the operand is delayed (to be accomplished sometime before the statement completes execution). So in the statement y = x * z++; the current value of z is used to evaluate the expression (i.e., z++ evaluates to z) and z only incremented after all else is done.
Compiler dependent side effects: printf("%d %d\n", ++n, pow(2,n)); or A[i] = i++;
Avoid side effects! If you are not sure about side effects, you wont take advantage of idiomatic expressions of C.



Selection Statements

- if
- if else
- switch

if statement

- if(Boolean expression)
  statement; /* single statement */

- if(Boolean expression) {
  /* more than one statement */

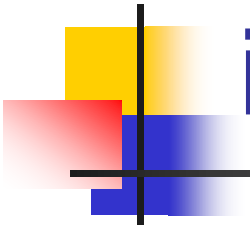
/* block is referred to as compound statement */

  statement1;
  …
  statement n;
  }

if statement - examples

```
if (x > 0)
     k++;

if(x > 0) {
     y = sqrt(x);
     k++;
}

if(x > 0) /* a common mistake */
     y = sqrt(x);
     k++;
```

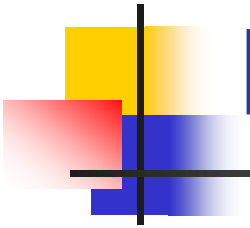
Name	Addr	Content
x		9
y		5
k		4



if else statement

- if(Boolean expression)
  statement for TRUE;
  else
  statement for FALSE;

- if(Boolean expression) {
  statement block for TRUE
  } else {
  statement block for FALSE
  }



Even or Odd

```
main()
{
    int n;
    printf("This program labels a number as"
        " even or odd.\n");
    printf("Enter a number: ");
    n = GetInteger();
    if (n % 2 == 0) {
        printf("That number is even.\n");
    } else {
        printf("That number is odd.\n");
    }
}
```

if else statement

- What does the following program do?
- Assume that x, y, temp are declared.

```
if (x > y)
   temp = x;
else
   temp = y;
```

Name	Addr	Content
x		9
y		5
temp		?

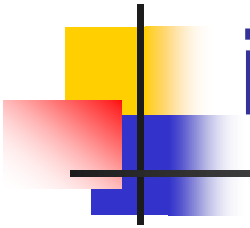
```
temp= x > y ? x : y;
```

Exercise

- Write an if-else statement to find both the maximum and minimum of two numbers.
- Assume that x, y, min, max are declared.

```
if (x > y) {
   max = x;
   min = y;
} else {
    max = y;
    min = x;
}
```

Name	Addr	Content				
x		9	9	3	3	6	6
y		5	5	8	8	6	6
max		?	9	9	8	8	6
min		?	5	5	3	3	6



if else statement

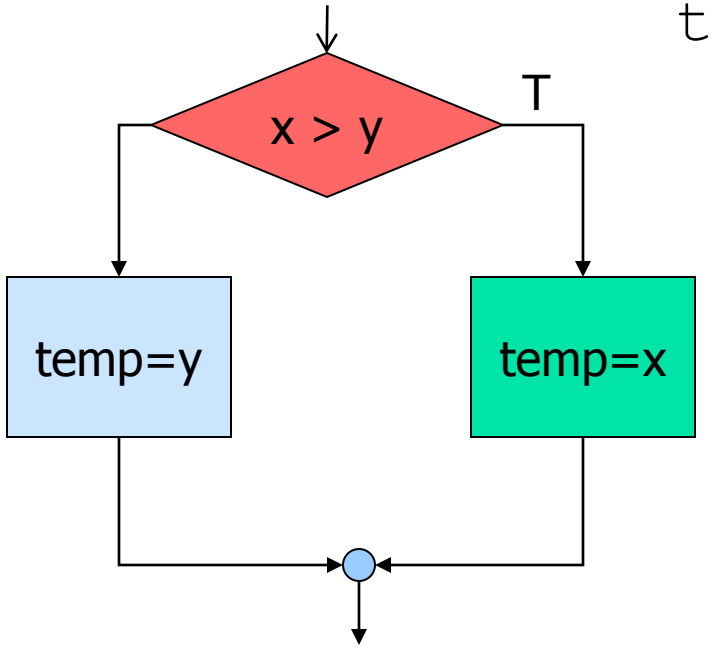
Split the following statement into two separate if statements

```
if (x > y)
  temp = x;
else
  temp = y;
```

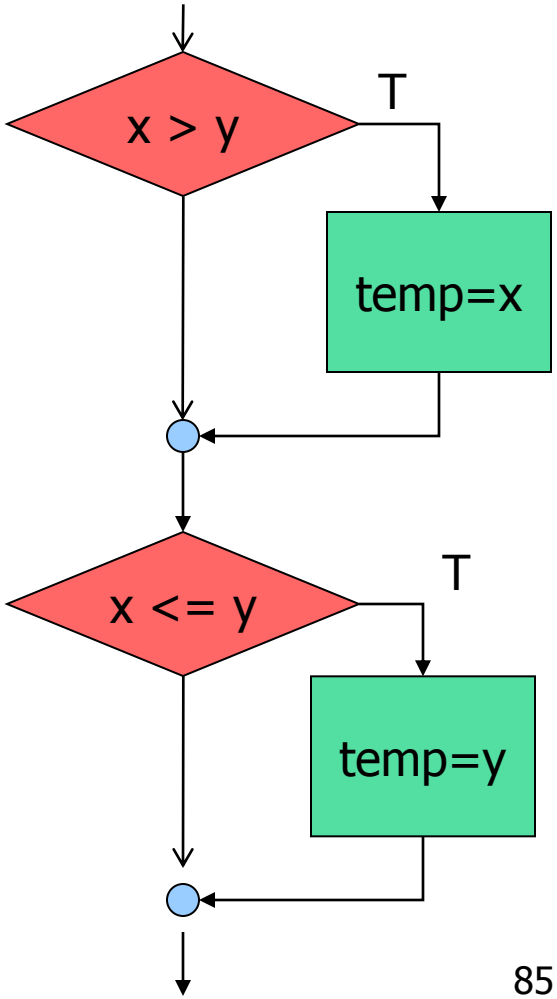
```
if (x > y)
  temp = x;

if (x <= y)
  temp = y;
```

Flow chart for previous slide

```
if (x > y)
   temp = x;
else
   temp = y;
```

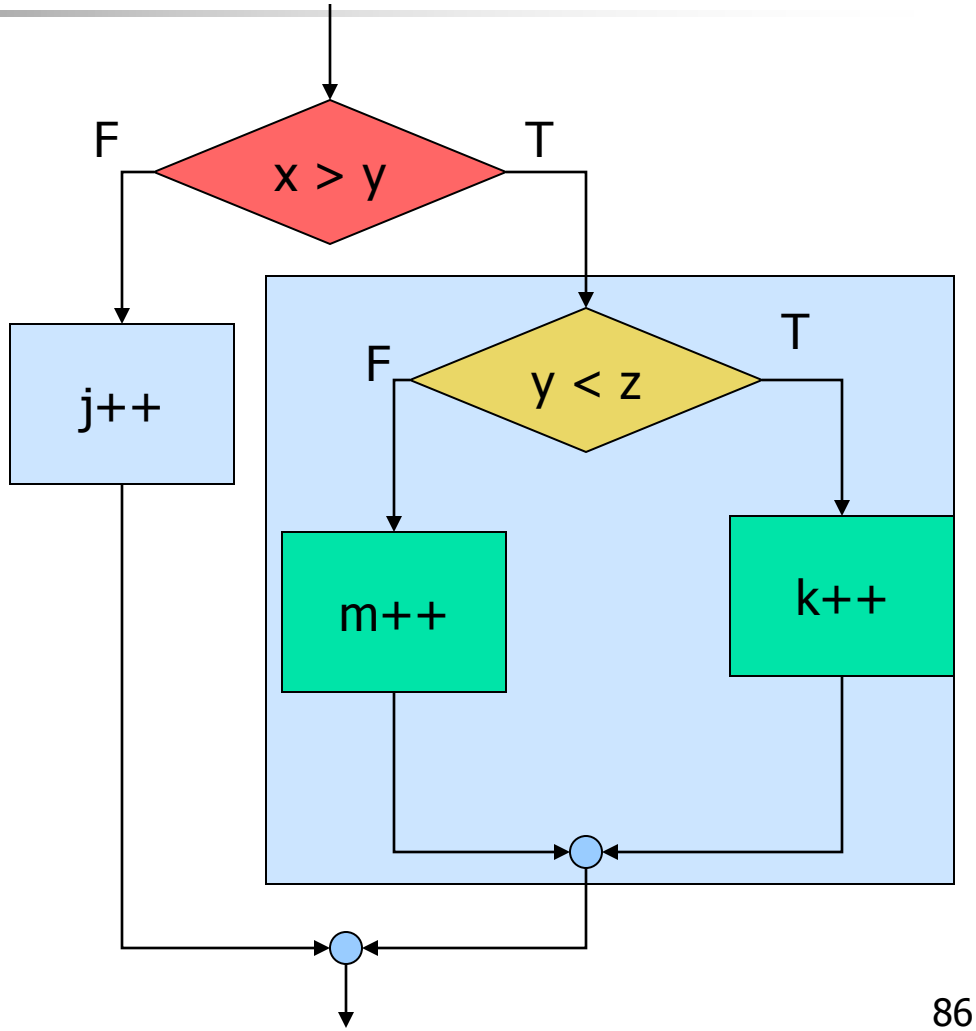



```
if (x > y)
   temp = x;

if (x <= y)
   temp = y;
```



nested if-else

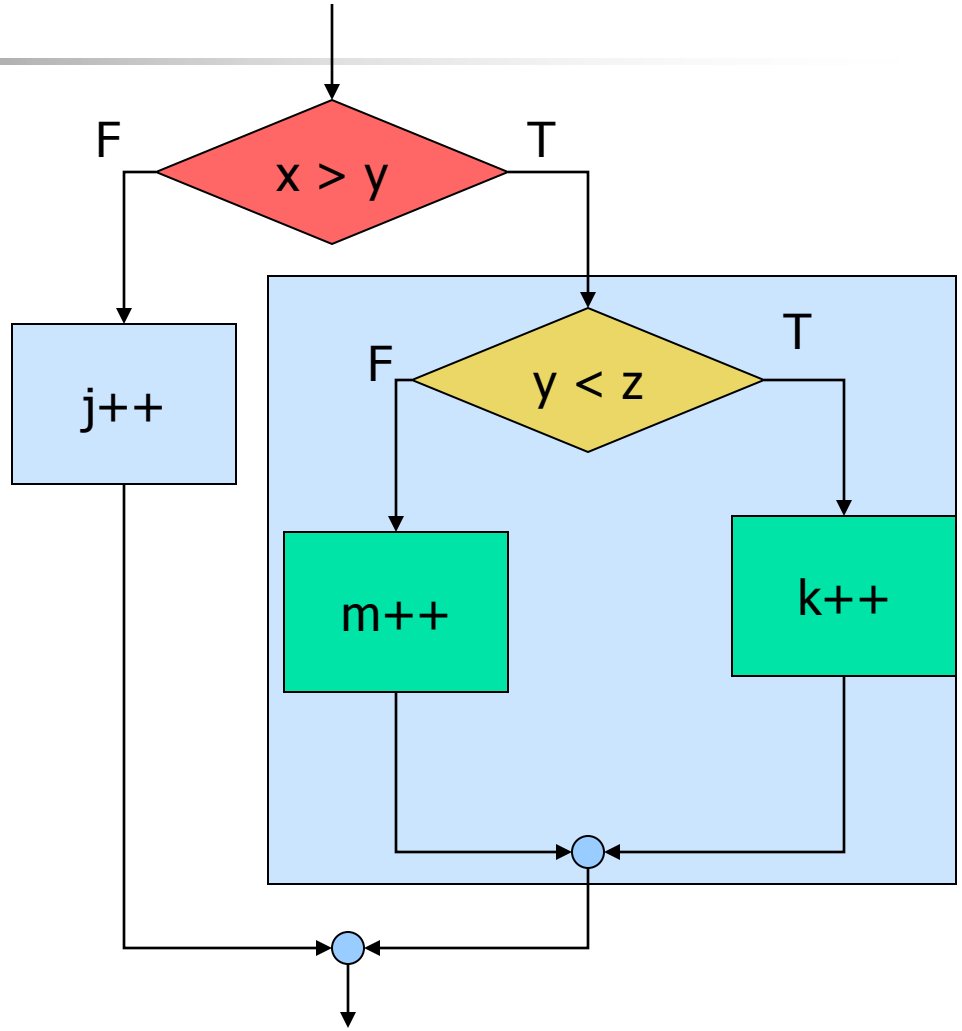
```
if(x > y) {
    if(y < z) {
        k++;
    } else {
        m++;
    }
} else {
    j++;
}
```



Exercise

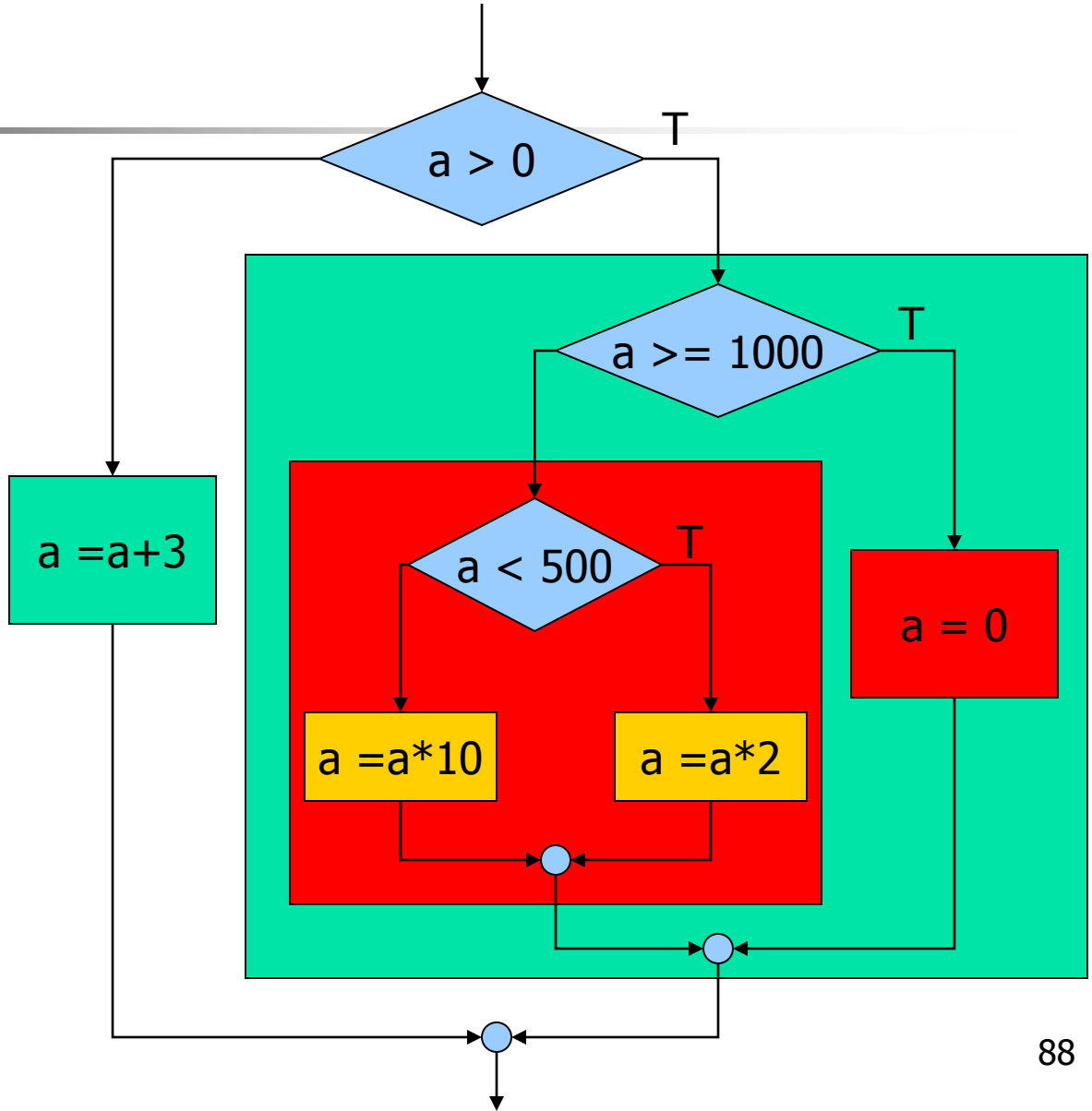
```
int x=9, y=7, z=2, k=0, m=0, j=0;
if(x > y)
    if(y < z)
        k++;
    else
        m++;
else
    j++;
```

What are the values of j, k and m?



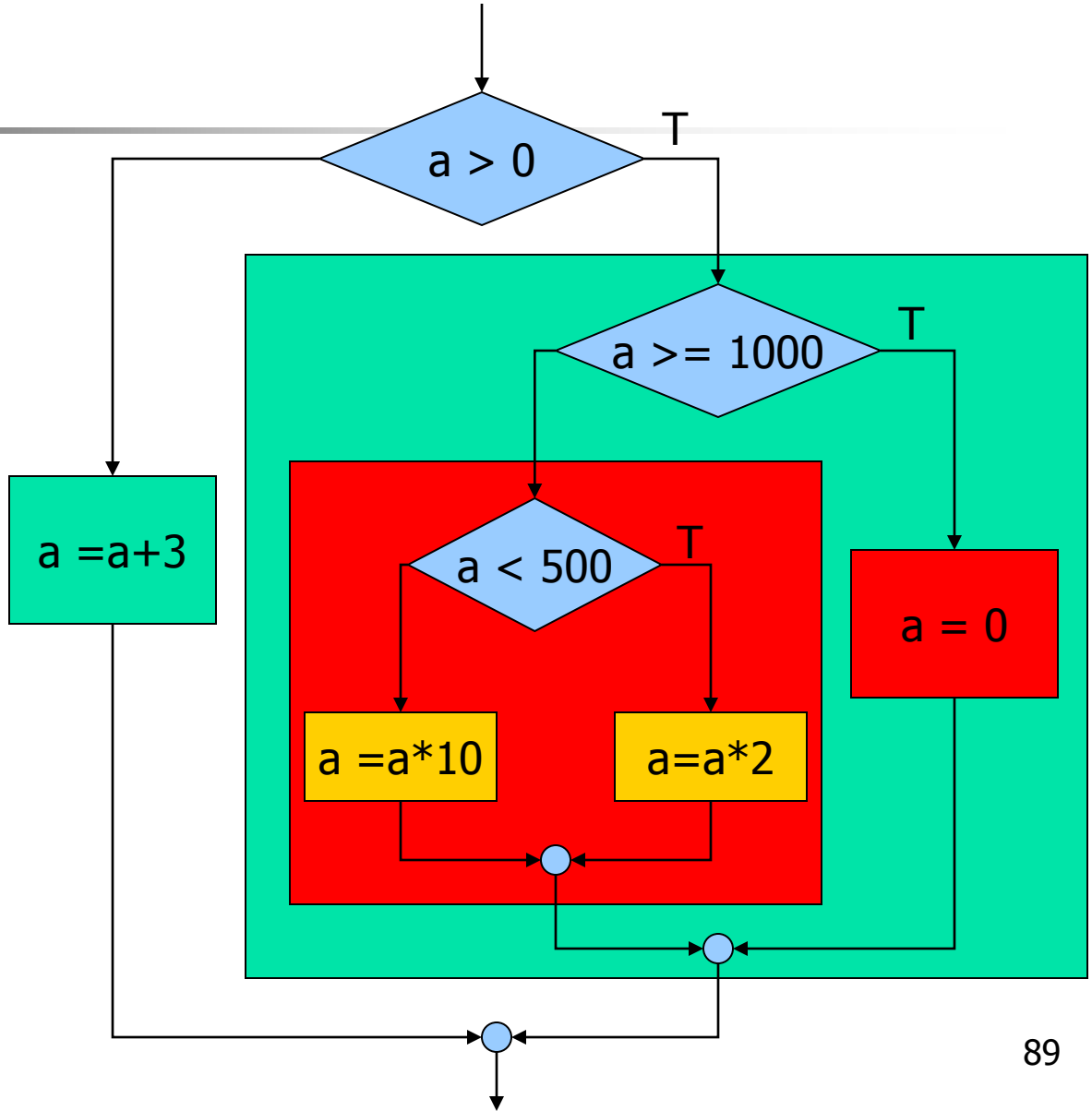
Exercise: Find the value of **a**

```
int a = 750;
if (a>0)
    if (a >= 1000)
        a = 0;
    else
        if (a <500)
            a =a*2;
        else
            a =a*10;
else
    a =a+3;
```

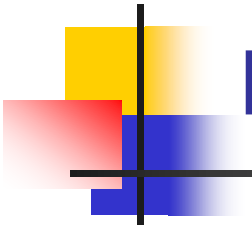


Exercise: Find the value of **a**

```
int a = 750;
if (a>0) {
    if (a >= 1000) {
        a = 0;
    } else {
        if (a <500) {
            a =a*2;
        } else {
            a =a*10;
        }
    }
} else {
    a =a*3;
}
```



Exercise: which task takes more time



- Suppose we have two tasks A and B
  - A takes Ah hours, Am minutes, and As seconds
  - B takes Bh hours, Bm minutes, and Bs seconds
- Write if-else statements to print out which task takes more time?

Indentation

```
int a = 750;
if (a>0)
    if (a >= 1000)
        a = 0;
    else
        if (a <500)
            a=a*2;
        else
            a=a*10;
else
    a =a+3;
```

Good

```
int a = 750;
if (a>0)
if (a >= 1000)
a = 0;
else
if (a <500)
a=a*2;
else
a=a*10;
else
a = a+3;
```

Not good

Indentation (cont'd)

- What is the output of the following program

  int a = 5, b = 3;

```
if (a>10)
      a = 50;
      b = 20;

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

Not good

```
if (a>10) {
      a = 50;
b = 20;
}
printf(" a = %d, b = %d\n",a, b);
```

```
if (a>10)
      a = 50;
b = 20;

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

Good

```
if (a>10) {
      a = 50;
      b = 20;
}
printf(" a = %d, b = %d\n",a, b);
```

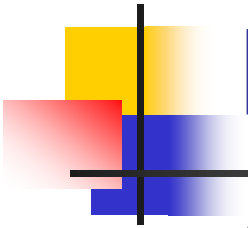
Switch Statement

```
switch(expression) {
        case constant:
                statement(s);
                break;
        case constant:
                statement(s);
                break;
         default:        /* default is optional */
                statement(s);
}
```

Switch Statement

- *Expression* must be of type integer or character
- The keyword **case** must be followed by a *constant*
- **break** statement is required unless you want all subsequent statements to be executed.

```
switch (op_code) {
   case 'N':
      printf("Normal\n");
      break;
   case 'M':
      printf("Maintenance Needed\n");
      break;
  default:
      printf("Error\n");
      break;
}
```



Exercise

- Convert the switch statement into if statement.

```
switch (op_code) {
    case 'N':
        printf("Normal\n");
        break;
    case 'M':
        printf("Maintenance Needed\n");
        break;
    default:
        printf("Error\n");
        break;
}
```

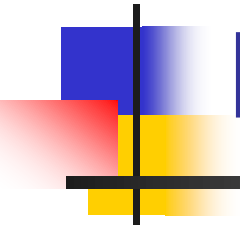
```
if (op_code == 'N')
    printf("Normal\n");
else if (op_code == 'M')
    printf("Maintenance Needed\n");
else
    printf("Error\n");
```

Exercise

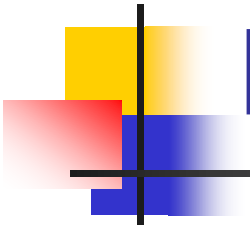
Convert the following nested `if/else` statements to a `switch` statement

```
if (rank==1 || rank==2)
   printf("Lower division \n");
else
{
   if (rank==3 || rank==4)
      printf("Upper division \n");
   else
   {
      if (rank==5)
         printf("Graduate student \n");
      else
         printf("Invalid rank \n");
   }
}
```

```
switch(rank) {
        case 1:
        case 2:
           printf("Lower division \n");
           break;
        case 3:
        case 4:
           printf("Upper division \n");
           break;
        case 5:
            printf("Graduate student \n");
            break;
        default:
         printf("Invalid rank \n");
}
```



More selection examples



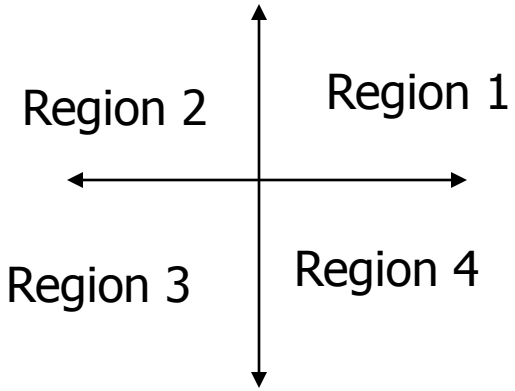
Max, Min, Median

- Write a program that reads 3 numbers a, b and c from user and computes minimum, median and maximum of the numbers.

- Example:
  - a = 2, b = 5, c = 3
    - minimum = 2, maximum = 5, median = 3
  - a = 2, b = 2, c = 3
    - minimum = 2, maximum = 3, median = 2

Region in a plane

- Write a program that reads a point (x, y) from user and prints its region



For example

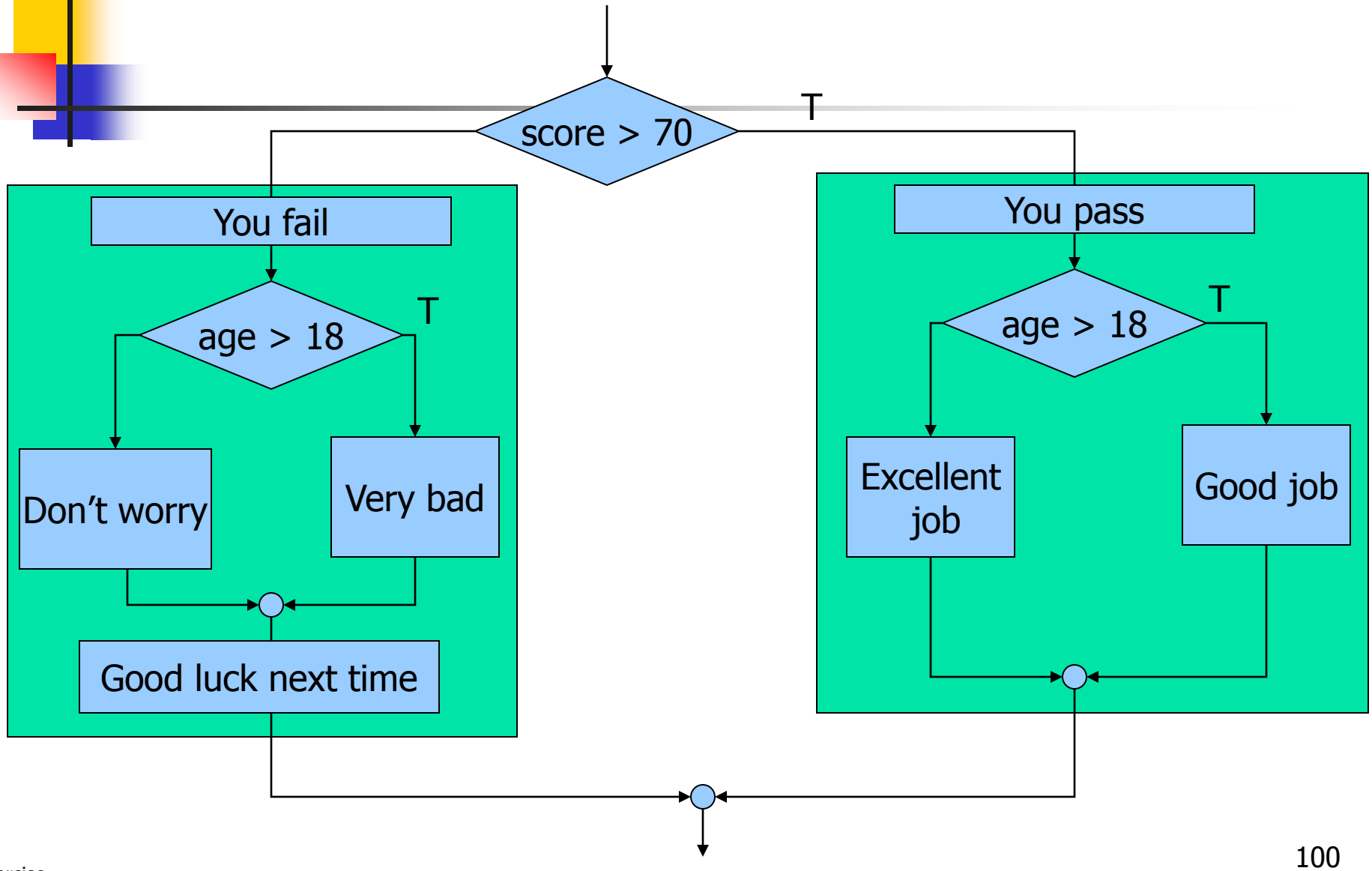
Enter x, y: 3 -1

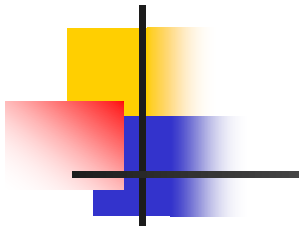
This point is in Region 4

Enter x, y: -1 -5

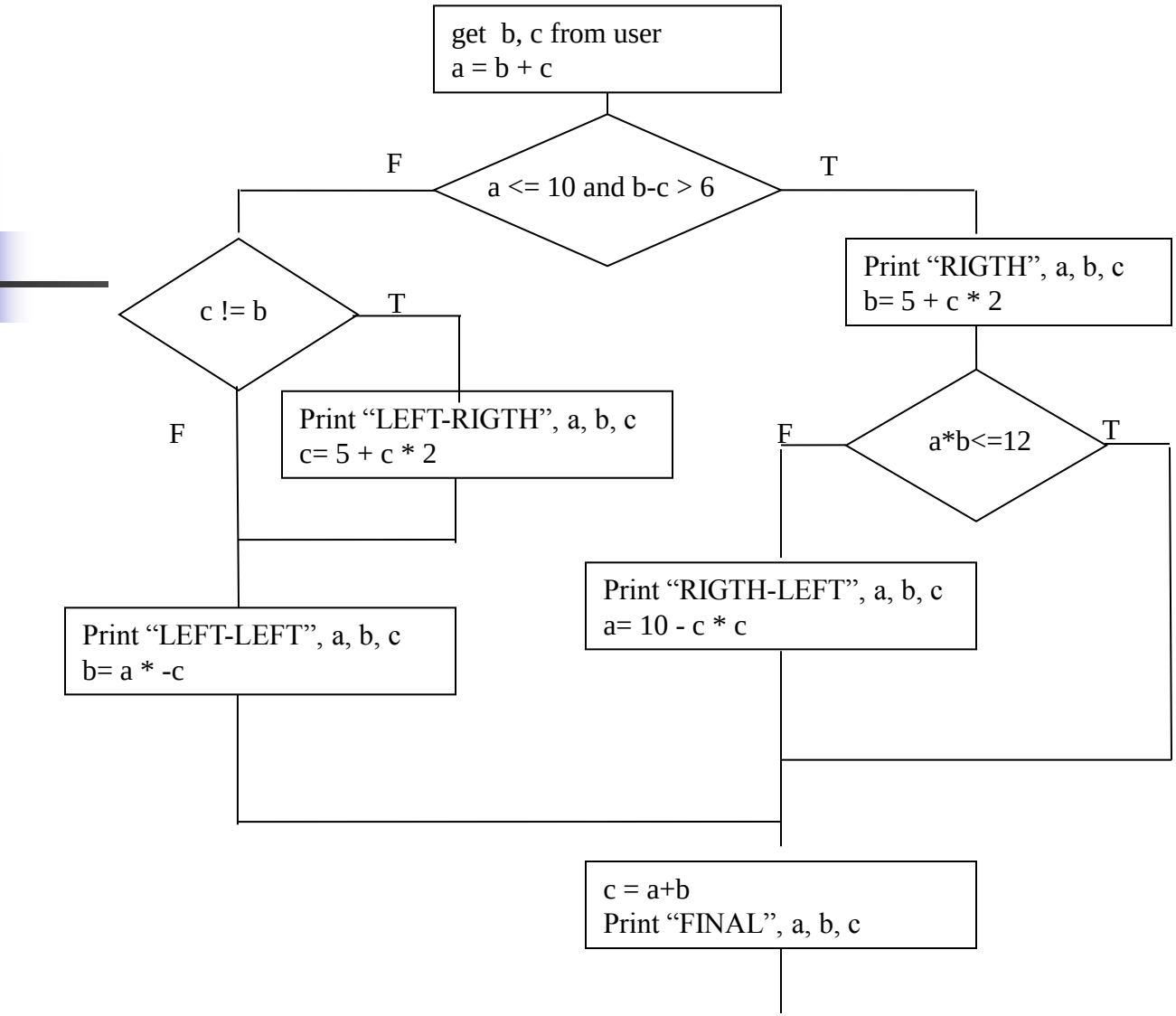
This point is in region 3

Write if-else statement

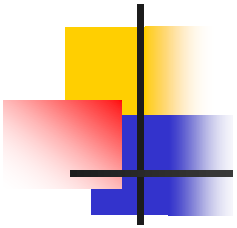




```
if (score > 70) {
    printf("You Pass\n");
    if (age > 18) {
            printf("Good job \n");
    } else {
            printf("Excellent job\n");
    }
} else {
    printf("You Fail\n");
    if (age > 18) {
            printf(" Very bad \n");
    } else {
            printf(" Don't worry \n");
    }
    printf(" Good luck next time \n");
}
```



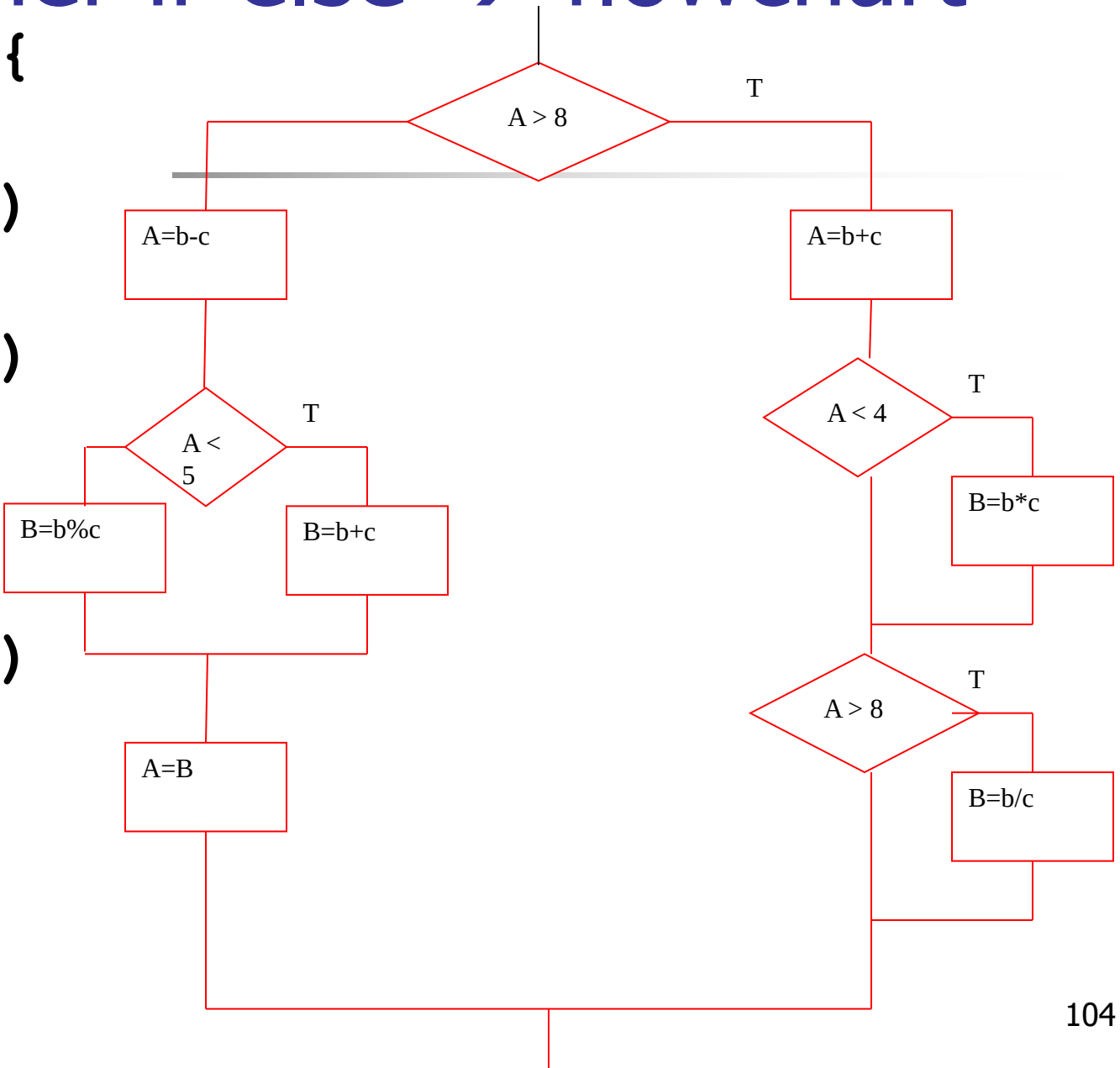
Print "RIGHT", a, b, c means
printf("RIGHT a=%lf b=%lf c=%lf \n",a, b, c);



```
a=b+c;
if (a<=10 && b-c>6) {
            printf("RIGHT a=%lf  b=%lf  c=%lf \n", a, b, c);
            b=5+c*2;
            if (a*b<=12) {
            } else {
               printf("RIGHT-LEFT a=%lf  b=%lf  c=%lf \n",a, b, c);
               a=10-c*c;
            }
} else {
            if (c != b)  {
               printf("LEFT-RIGHT a=%lf  b=%lf  c=%lf \n",a, b, c);
               c=5+c*2;
             }
             printf("LEFT-LEFT a=%lf  b=%lf  c=%lf \n",a, b, c);
             b=a*-c;
}
c=a+b;
printf("Final a=%lf  b=%lf  c=%lf \n",a, b, c);
```

Another if-else → flowchart

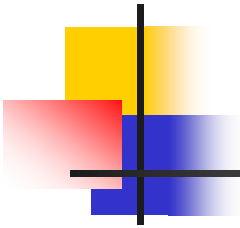
```
if( A > 8) {
    A=b+c;
    if(A < 4)
      B=b*c;
    if(A > 8)
      B=b/c;
} else {
    A=b-c;
    if(A < 5)
      B=b+c;
    else
      B=b%c;
    A=B;
}
```



Exercise: Assign Letter Grade

- Given a score and the following grading scale write a program to find the corresponding grade.

90-100	A
80-89	B
70-79	C
60-69	D
0-59	F



Solution-1

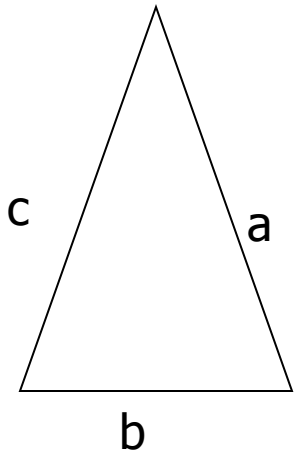
```
 if ((score >= 90) && (score <=100))
   grade = 'A';
else if ((score >= 80) && (score <= 89))
   grade = 'B';
else if ((score >= 70) && (score <= 79))
   grade = 'C';
else if ((score >= 60) && (score <= 69))
   grade = ‘D';
else if ((score >= 0) && (score <= 59))
   grade = ‘F';
else
   printf("Invalide Score\n");
```

Solution-2

```
if ((score >= 0) && (score <= 100))
    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';
else
   printf("Invalide Score\n");
```

Triangle inequality

- Suppose we want to check if we can make a triangle using a, b, c



|a-b| <= c |a-c| <= b |b-c| <= a

a+b >= c a+c >= b b+c >= a

Charge for money transfer

- Suppose you transfer $N and bank's charge occurs as follows.

$$\text{cost} = \begin{cases} \$10 & \text{if } N \le \$500 \\ \$10 + 2\% \text{ of } N & \text{if } 500 < N \le 1000 \\ \$15 + 0.1\% \text{ of } N & \text{if } 1000 < N < 10000 \\ \$30 & \text{Otherwise} \end{cases}$$

- Write a program that reads N and computes cost

$\mu^2\sigma^2)$ if $0 < \rho$

Compute Queuing Delay

- Write C program that gets ρ, μ, and σ then computes and prints out average delay in a queuing system, where the average delay is given as follows

$$AvgDelay = \begin{cases} \dfrac{\rho}{1-\rho} - \dfrac{\rho^2}{2(1-\rho)}(1-\mu^2\sigma^2) & \text{if } 0 < \rho < 1 \\ \infty & \text{if } \rho \geq 1 \end{cases}$$

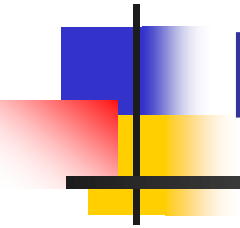
```
#include <stdio.h>
int main(void)
{
   /*  Declare variables.  If needed, you can declare more*/
   double rho, mu, sigma, AvgDelay;

   printf("Enter rho(utilization), mu(service time) and "
       "sigma (standard deviation of service time) : ");
   scanf("%lf %lf %lf", &rho, &mu, &sigma);
  /* Compute and print the average delay using rho, mu, sigma */

        if( rho > 0 && rho < 1) {
            AvgDelay = (rho / (1 - rho)) -
                       rho*rho / (2 * (1-rho)) *
                       (1-mu*mu*sigma*sigma);
            printf("AvgDelay = %lf \n", AvgDelay);
        } else if (rho >=1){
            printf("AvgDelay is infinity \n");
        } else
            printf("rho cannot be negative \n");
   /*  Exit program.  */
   return 0;
}
```

Spell out a number in text using if-else and switch

- Write a program that reads a number between 1 and 999 from user and spells it out in English.

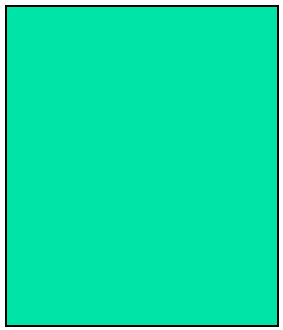
For example:

- 453 → Four hundred fifty three
- 37 → Thirty seven
- 204 → Two hundred four



Loop (Repetition) Structures

Problem: Conversion table degrees → radians



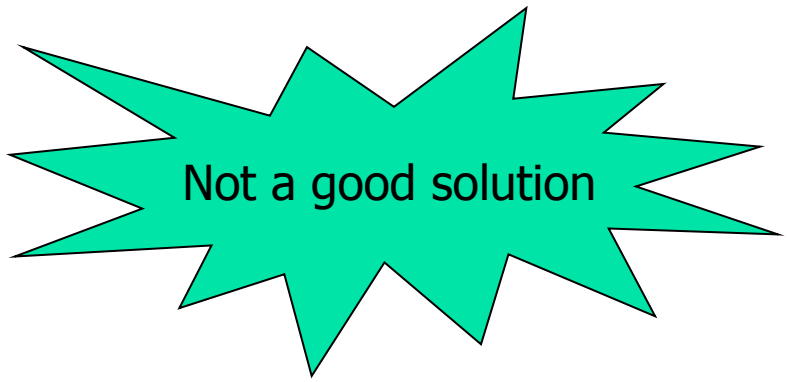
radians = degrees * PI / 180;

Degrees to Radians

0	0.000000
10	0.174533
20	0.349066
30	0.523599
…	
340	5.934120
350	6.108653
360	6.283186

Sequential Solution

```
degrees = ???
radians = degrees*PI/180;
printf("%6i %9.6f \n", degrees, radians);
```



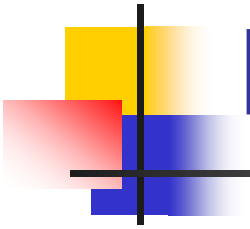

```
#include <stdio.h>
#define PI 3.141593

int main(void)
{
   int degrees=0;
   double radians;

   printf("Degrees to Radians \n");

  degrees = 0;
  radians = degrees*PI/180;
  printf("%6i %9.6f \n", degrees, radians);

  degrees = 10;
  radians = degrees*PI/180;
  printf("%6i %9.6f \n", degrees, radians);

  degrees = 20;
  radians = degrees*PI/180;
  printf("%6i %9.6f \n", degrees, radians);
...
  degrees = 360;
  radians = degrees*PI/180;
  printf("%6i %9.6f \n", degrees, radians);
}
```

Loop Solution

```
#include <stdio.h>
#define PI 3.141593

int main(void)
{
   int degrees=0;
   double radians;

   printf("Degrees to Radians \n");

  while (degrees <= 360)   {

      radians = degrees*PI/180;
      printf("%6i %9.6f \n", degrees, radians);
      degrees += 10;
   }
 }
```

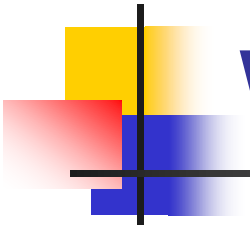
```
degrees = ???
radians = degrees*PI/180;
printf("%6i %9.6f \n", degrees, radians);
```

degrees+=10
means
degrees= degrees+10



Loop (Repetition) Structures

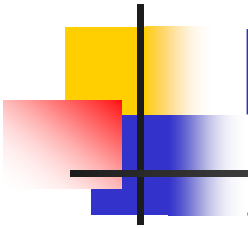
- **while** statement
- **do while** statement
- **for** statement
- Two new statements used with loops
  - break and continue



while statement

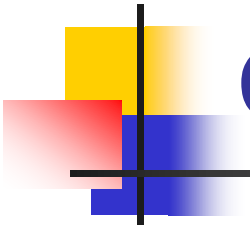
- while(*expression*)
  statement;

- while(*expression*) {
  statement;
  statement;
  …
  }



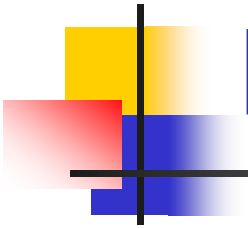
Example

```
#include <stdio.h>
#define PI 3.141593

int main(void)
{
   int degrees=0;
   double radians;

   printf("Degrees to Radians \n");
   while (degrees <= 360)
   {
      radians = degrees*PI/180;
      printf("%6i %9.6f \n", degrees, radians);
      degrees += 10;
   }
return 0;
}
```



do while

- do
  statement;
  while(expression);
- do {
  statement1;
  statement2;
  ...
  } while(expression);
- note - the expression is tested *after* the statement(s) are executed, so statements are executed *at least once*.



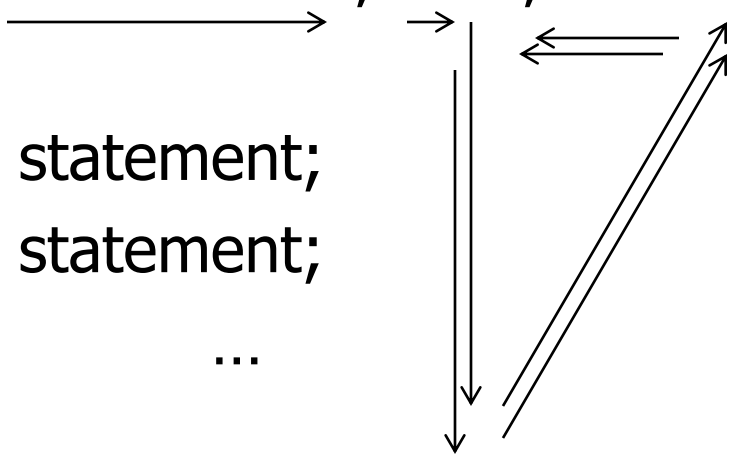
Example

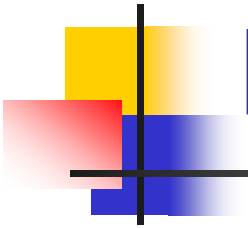
```
#include <stdio.h>
#define PI 3.141593

int main(void)
{
  int degrees=0;
   double radians;

   printf("Degrees to Radians \n");
   do
   {
      radians = degrees*PI/180;
      printf("%6i %9.6f \n",degrees,radians);
      degrees += 10;
   } while (degrees <= 360);
   return 0;
}
```

for statement

- for(initialization ; test ; increment or decrement)
  statement;

- for(initialization ; test ; increment or decrement)
  {
  statement;
  statement;
  …
  }





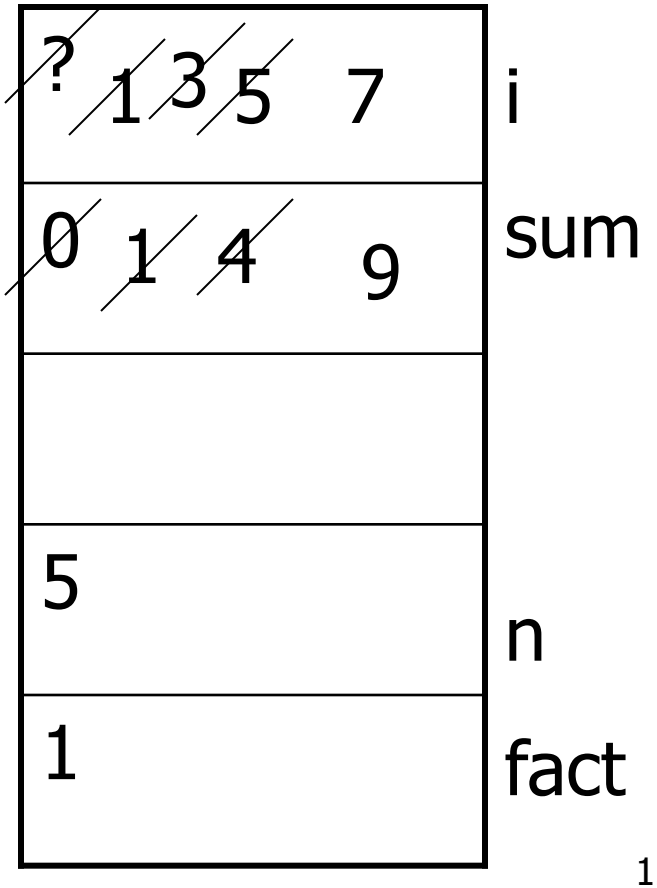
Example

```
#include <stdio.h>
#define PI 3.141593

int main(void)
{
   int degrees;
   double radians;

   printf("Degrees to Radians \n");
   for (degrees=0; degrees<=360; degrees+=10)
   {
      radians = degrees*PI/180;
      printf("%6i %9.6f \n", degrees, radians);
   }
   return 0;
}
```

Examples

```
int sum =0, i;
for( i=1 ;  i < 7; i=i+2 ){
        sum = sum+i;
}

int fact=1, n;
for( n=5 ;  n>1 ; n--){
      fact = fact * n;
}
```

```
n--; means n=n-1;
n++; means n=n+1;
```



Exercise

Determine the number of times that each of the following `for` loops are executed.

```
for (k=3; k<=10; k++){
        statements;
}
```

$$\left\lfloor \frac{final - initial}{increment} \right\rfloor + 1$$

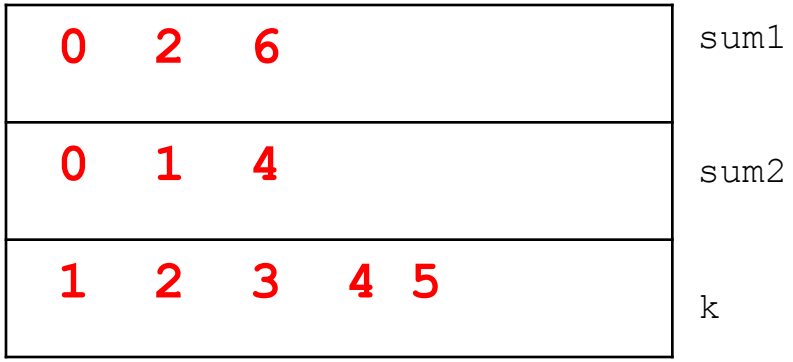
```
for (k=3; k<=10; ++k) {
        statements;
}
```

```
for (count=-2; count<=5; count++) {
        statements;
}
```

Example

- What will be the output of the following program, also show how values of variables change in the memory.

```
int sum1, sum2, k;
sum1 = 0;
sum2 = 0;
for( k = 1; k < 5; k++) {
     if( k % 2 == 0)
       sum1 =sum1 + k;
     else
       sum2 = sum2 + k;
}
printf("sum1 is %d\n", sum1);
printf("sum2 is %d\n", sum2);
```




```
sum1 is 6
sum2 is 4
```

For vs. while loop

Convert the following for loop to while loop

```
for( i=5; i<10; i++) {
   pritntf(" i = %d \n", i);
}

i=5;
while(i<10){
   pritntf(" i = %d \n", i);
   i++;
}
```

break statement

- break;
  - terminates loop (In cases of nested loops, break only breaks the innermost loop.)
  - execution continues with the first statement following the loop

```
sum = 0;
for (k=1; k<=5; k++) {
    scanf("%lf",&x);
    if (x > 10.0)
        break;
    sum +=x;
}
printf("Sum = %f \n",sum);
```

```
sum = 0;
k=1;
while (k<=5) {
      scanf("%lf",&x);
    if (x > 10.0)
        break;
    sum +=x;
    k++;
}
printf("Sum = %f \n",sum);
```

continue statement

- continue;
  - forces next iteration of the loop, skipping any remaining statements in the loop

```
sum = 0;
for (k=1; k<=5; k++) {
    scanf("%lf",&x);
    if (x > 10.0)
        continue;
    sum +=x;
}
printf("Sum = %f \n",sum);
```

```
sum = 0;
k=1;
while (k<=5) {
      scanf("%lf",&x);
    if (x > 10.0){
        k++;
        continue;
    }
    sum +=x;
    k++;
}
printf("Sum = %f \n",sum);
```

Example: what will be the output

```
int main()
{
  int a, b, c;
  a=5;
  while(a > 2) {
    for (b = a ; b < 2 * a ; b++ ) {
        c = a + b;
        if (c < 8) continue;
        if (c > 11) break;
        printf( "a = %d   b = %d    c = %d \n", a, b, c);
    } /* end of for-loop */
    a--;
  } /* end of while loop */
}
```

```
a = 5   b = 5   c = 10
a = 5   b = 6   c = 11
a = 4   b = 4   c = 8
a = 4   b = 5   c = 9
a = 4   b = 6   c = 10
a = 4   b = 7   c = 11
a = 3   b = 5   c = 8
```

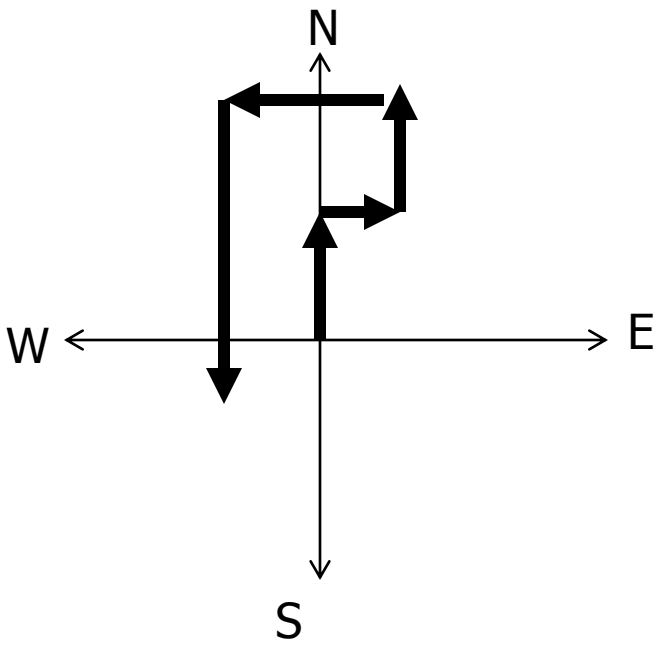
For vs. while loop : Convert the following for loop to while loop

```
for( i=5; i<10; i++) {
    printf("AAA %d \n", i);
    if (i % 2==0) continue;
    pritntf("BBB %d \n", i);
}
```

```
i=5;
while(i<10) {
    printf("AAA %d \n", i);
    if (i % 2==0) {
        i++;
        continue;
    }
    pritntf("BBB %d \n", i);
    i++;
}
```

Example: A man walks

- Suppose a man (say, A) stands at (0, 0) and waits for user to give him the direction and distance to go.
- User may enter N E W S for north, east, west, south, and any value for distance.
- When user enters 0 as direction, stop and print out the location where the man stopped



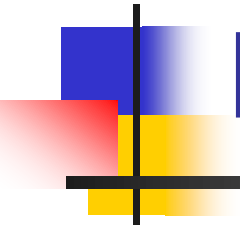
```
float x=0, y=0;
 char direction;
 float distance;
 while (1) {
      printf("Please input the direction as N,E,W,S (0 to exit): ");
      scanf("%c", &direction);          fflush(stdin);
      if (direction=='0'){
         break;    /* stop input, get out of the loop */
      }
      if (direction!='N' && direction!='S' && direction!='E' && direction!='W') {
         printf("Invalid direction, re-enter \n");
         continue;
      }
      printf("Please input the mile in %c direction: ", direction);
      scanf ("%f", &distance);    fflush(stdin);
      if (direction == 'N'){                  /*in north, compute the y*/
         y = y + distance;
      } else if (direction == 'E'){           /*in east, compute the x*/
         x = x + distance;
      } else if (direction == 'W'){           /*in west, compute the x*/
         x= x - distance;
      } else if (direction == 'S'){           /*in south, compute the y*/
         y = y- distance;
      }
 }
 printf("\nCurrent position of A: (%4.2f, %4.2f)\n", x, y);   /* output A's location */
```


Exercise

Goto and Labels (usually we don't use this)

- In some rare cases we may need it to branch to another part of the program marked by label (e.g., break deeply nested loops)

```
for (…){
   for (…) {
        …
        if (disaster) goto error;
   }
}
error: printf(" clean up the mess ");
…
```



More loop examples

Exercise

- What is the output of the following program?

```
for (i=1; i<=5; i++) {
        for (j=1; j<=4; j++){
            printf("*");
        }
        printf("\n");
}
```

Output

```
****
****
****
****
****
```

Exercise

- What is the output of the following program?

```
for (i=1; i<=5; i++) {
        for (j=1; j<=i; j++){
             printf("*");
        }
        printf("\n");
}
```

Output

```
*
**
***
****
*****
```

Example: **nested loops** to generate the following output

```
i=1 *
i=2 + +
i=3 * * *
i=4 + + + +
i=5 * * * * *
```

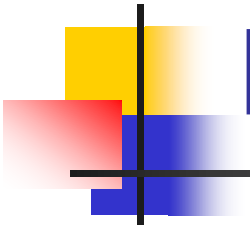
```
int i, j;
for(i=1; i <= 5; i++){
   printf("i=%d ", i);
   for(j=1; j <= i; j++) {
      if (i % 2 == 0)
         printf("+ ");
      else
         printf("* ");
   }
   printf("\n");
}
```

Exercise: Modify the following program to produce the output.

```
for (i=A; i<=B; i++) {
        for (j=C; j<=D; j++) {
            printf("*");
        }
        printf("\n");
}
```

Output

```
*****
****
***
**
*
```

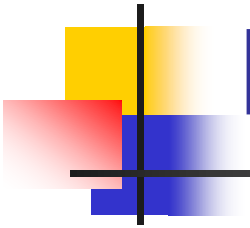


Exercise

- Write a program using loop statements to produce the following output.

Output

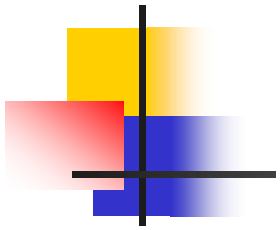
```
    *
   **
  ***
 ****
*****
```



Example

- Write a program that prints in two columns n even numbers starting from 2, and a running sum of those values. For example suppose user enters 5 for n, then the program should generate the following table:

```
Enter  n (the number of even numbers): 5
Value   Sum
  2       2
  4       6
  6      12
  8      20
10       30
```



```
#include <stdio.h>
int main(void)
{
    /*  Declare variables.  */
    int n;
    int sum, i;

    printf("Enter  n ");
    scanf("%d",&n);

    printf("Value \t Sum\n");
    sum = 0;
    for(i=1; i <=n; i++){
      sum = sum + 2*i;
      printf("%d \t %d\n", 2*i, sum);
    }
    return 0;
}
```

Compute x^y when y is integer

- Suppose we don't have pow(x,y) and y is integer, write a loop to compute x^y

```
printf("Enter x, y :");
scanf("%d %d", &x, &y);
res=1;
for(i=1; i<=y; i++){
   res = res * x;
}
```

Exercise: sum

- Write a program to compute the following

$$\sum_{i=1}^{n} i = 1+2+3+\ldots+n$$

$$total = 2+4+6+\ldots+2n$$

```
Enter n
total=0;
for(i=1; i<=n; i++)
   total = total + i ;
print total
```

```
Enter n
total=0;
for(i=1; i<=n; i++)
   total = total + 2 * i ;
print total
```

Exercise: sum

- Write a program to compute the following

$$\sum_{i=0}^{m} x^i = x^0 + x^1 + x^2 + x^3 + x^4 + \cdots + x^m$$

```
Enter x and m
total=0;
for(i=0; i<=m; i++)
   total = total + pow(x, i);
print total
```

```
Enter x and m
total=0; sofarx=1;
for(i=0; i<=m; i++) {
   total = total +sofarx;
   sofarx = sofarx * x;
}
print total
```

Exercise: ln 2

- Write a program to compute the following

$$\ln 2 = \frac{1}{1} - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \frac{1}{5} - \frac{1}{6} + \frac{1}{7} - \cdots \pm \frac{1}{n}$$

```
Enter n
ln2=0;
for(i=1; i<=n; i++)
   if ( i % 2 == 0)
          ln2 = ln2 - 1.0 / i;
   else
          ln2 = ln2 + 1.0 / i;
print total
```

Exercise: e^x

- Write C program that reads the value of *x* and *n* from the keyboard and then approximately computes the value of e^x using the following formula:

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!}$$

- Then compare your approximate result to the one returned by exp(x) in C library, and print out whether your approximation is higher or lower.

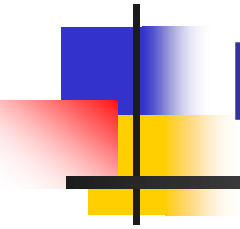
```
int    i, n;
double x, ex;
double powx, fact;

printf("Enter the value of x and n : ");
scanf("%lf %d",&x, &n);

/* Write a loop to compute e^x using the above formula */
ex=1.0;   fact=1.0;    powx=1.0;
for(i=1; i<=n; i++){
    powx = powx * x;
    fact = fact * i;
    ex = ex + powx / fact;
}
printf("Approx value of e^x is %lf when n=%d\n",ex, n);

/* Check if ex is higher/lower than exp(x) in math lib.*/
if(ex < exp(x))
   printf("ex est is lower than exp(x)=%lf\n",exp(x));
else if (ex > exp(x))
   printf("ex est is higher than exp(x)=%lf\n",exp(x));
else
   printf("ex est is the same as exp(x)\n");
```

Exercise: sin x

- Compute sin x using

$$\sin x = \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots + (-1)^n \frac{x^{2n+1}}{(2n+1)!}$$

```
printf("Enter x n :");  scanf("%lf %d", &x, &n);
total=x;  xpowk=x;  factk=1;
for(i=1; i <= n; i++){
    k= 2*i+1;
    xpowk = xpowk * x * x;
    factk = factk * k * (k-1);
    if (i%2==0)  total= total + xpowk/factk;
    else  total= total - xpowk/factk;
}
printf( "sin(%lf) is %lf\n", x, total);
```



C Programming Language: **Modular Programming With Functions**

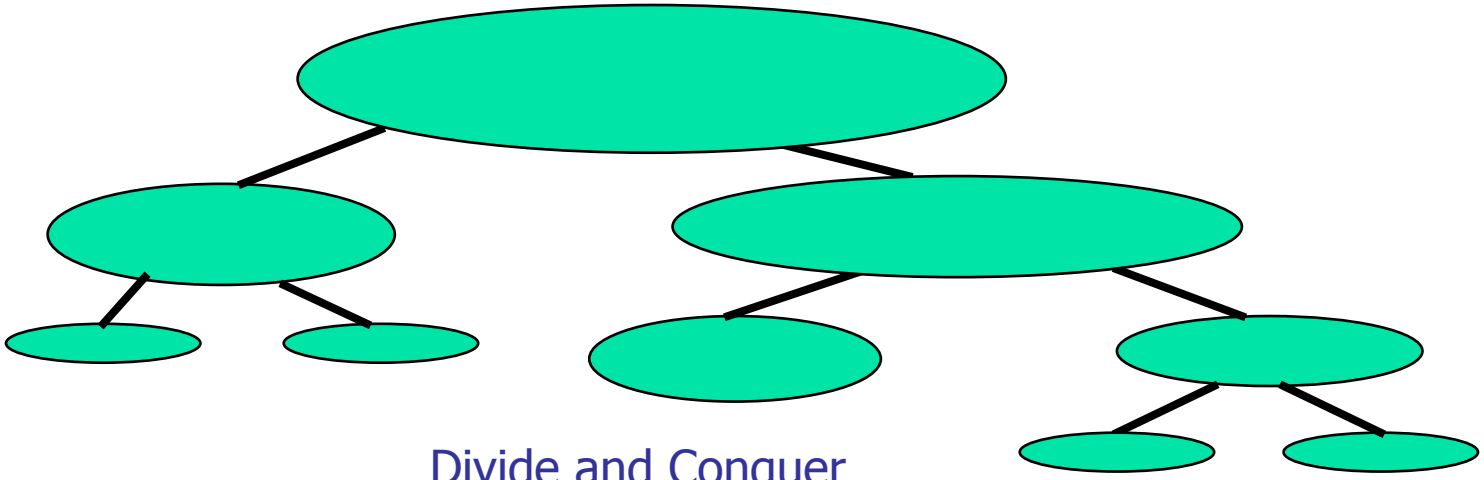
- How do you solve a big/complex problem?

```c
/*
 * File: powertab.c
 * ----------------
 * This program generates a table comparing values
 * of the functions n^2 and 2^n.
 */
#include <stdio.h>
#include "genlib.h"
/*
 * Constants
 * ---------
 * LowerLimit -- Starting value for the table
 * UpperLimit -- Final value for the table
 */
#define LowerLimit  0
#define UpperLimit 12

/* Private function prototypes */
static int RaiseIntToPower(int n, int k);
/* Main program */
main()
{
    int n;

    printf("    |   2 |    N\n");
    printf("  N |  N  |   2\n");
    printf("----+-----+------\n");
    for (n = LowerLimit; n <= UpperLimit; n++) {
        printf(" %2d | %3d | %4d\n", n,
               RaiseIntToPower(n, 2),
               RaiseIntToPower(2, n));
    }
}
/*
 * Function: RaiseIntToPower
 * Usage: p = RaiseIntToPower(n, k);
 * ---------------------------------
 * This function returns n to the kth power.
 */
static int RaiseIntToPower(int n, int k)
{
    int i, result;
    result = 1;
    for (i = 0; i < k; i++) {
        result *= n;
    }
    return (result);
}
```

```java
/*
 * File: powertab.java
 * -------------------
 * This program generates a table comparing values
 * of the functions n^2 and 2^n.
 */
import java.io.*;
public class powertab {
/*
 * Constants
 * ---------
 * LowerLimit -- Starting value for the table
 * UpperLimit -- Final value for the table
 */
public static final init LowerLimit = 0;
public static final init UpperLimit = 12;



/* Main program */
public static main()
{
    int n;

    System.out.println("    |   2 |    N");
    System.out.println("  N |  N  |   2");
    System.out.println("----+-----+------");
    for (n = LowerLimit; n <= UpperLimit; n++) {
      System.out.format(" %2d | %3d | %4d\n", n,
               RaiseIntToPower(n, 2),
               RaiseIntToPower(2, n));
    }
}
/*
 * Function: RaiseIntToPower
 * Usage: p = RaiseIntToPower(n, k);
 * ---------------------------------
 * This function returns n to the kth power.
 */
private static int RaiseIntToPower(int n, int k)
{
    int i, result;
    result = 1;
    for (i = 0; i < k; i++) {
        result *= n;
    }
    return (result);
}
}
```

Modular design

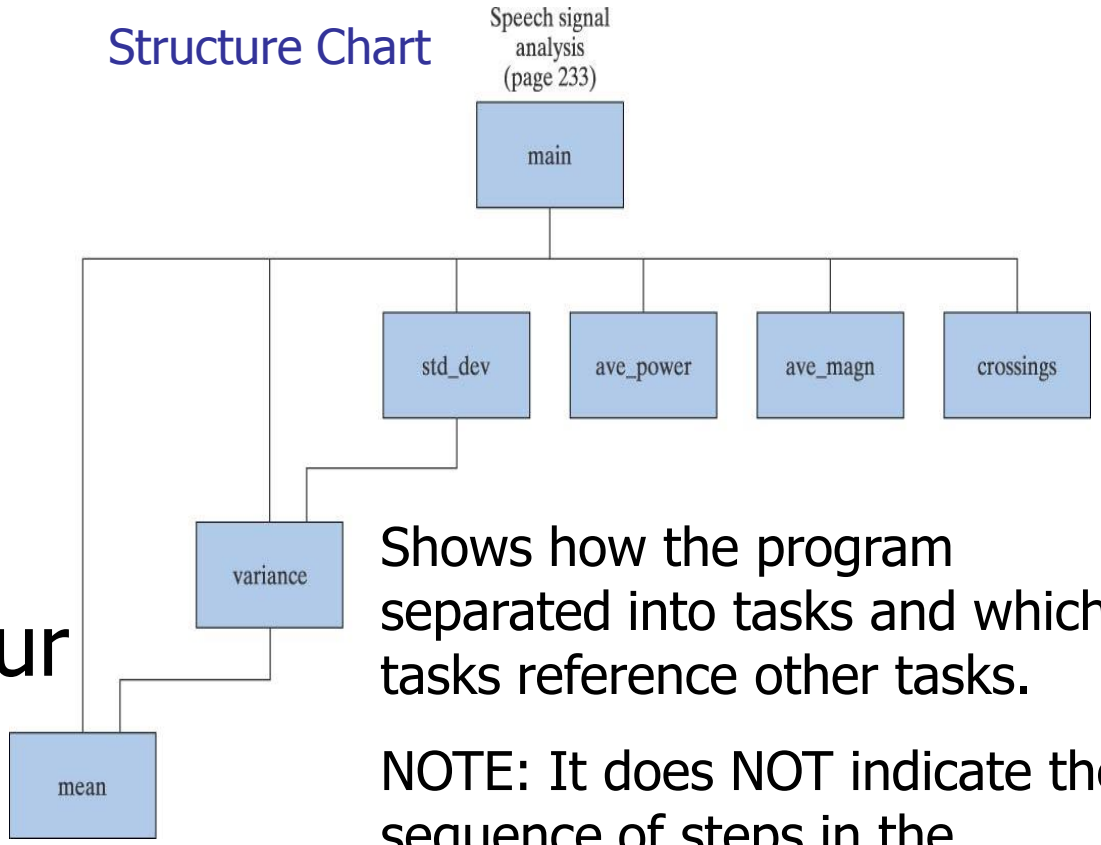
- Top-down Design
  - Start from the big picture
  - Use a process called divide-and-conquer
  - Keep dividing the problem into small tasks and solve each task.
  - Then combine these solutions.



Modularity (cont'd)

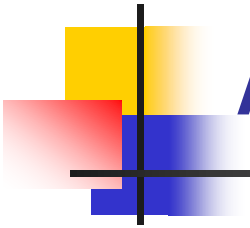
- In C we use **functions** also referred to as **modules** to perform specific tasks that we determined in our solution

Structure Chart



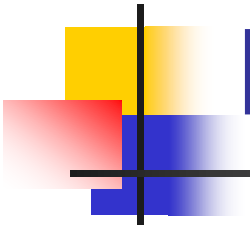
Shows how the program separated into tasks and which tasks reference other tasks.

NOTE: It does NOT indicate the sequence of steps in the program!



Advantages of using modules

- Modules can be written and tested separately
- Modules can be reused
- Large projects can be developed in parallel
- Reduces length of program, making it more readable
- Promotes the concept of **abstraction**
  - A module hides details of a task
  - We just need to know what this module does
  - We don’t need to know how it does it



Functions

- Every C program starts with `main()` function
- Additional functions are called or invoked when the program encounters function names
- Functions could be
  - Pre-defined library functions (e.g., printf, sin, tan) or
  - Programmer-defined functions (e.g., my_printf, area)
- Functions
  - **Perform a specific task**
  - May take arguments
  - May return a single value to the calling function
  - May change the value of the function arguments (call by reference)

Function definition

```
return_type function_name (parameters)
{
      declarations;
      statements;
}
```

```
int my_add_func(int a, int b)
{
   int sum;
   sum = a + b;
   return sum;
}
```

Programmer-Defined Functions Terminology

- Function Prototype describes how a function is called

```
int my_add_func(int a, int b);
```

- Function Call

```
result = my_add_func(5, X);
```

- Function implementation

```
int my_add_func(int a, int b)
{
…
}
```

- Function parameters
  - Formal parameters
  - Actual parameter
  - Formal parameters must match with actual parameters in *order*, *number* and *data type*.
  - If the type is not the same, type conversion will be applied (coercion of arguments). But this might cause some errors (double→int) so you need to be careful!

Example: Pre-defined Functions

So far, we used several pre-defined functions!

```
#include <stdio.h>
#include <math.h>
int main(void)
{
 double angle;
 printf("Input angle in radians: \n");
 scanf("%lf", &angle);
 printf("The sine of the angle is %f\n",
            sin(angle) );
  return 0;
}
```

```
double sin(double radian);

double sin(double radian)
{
/* details of computing sin */
}
```

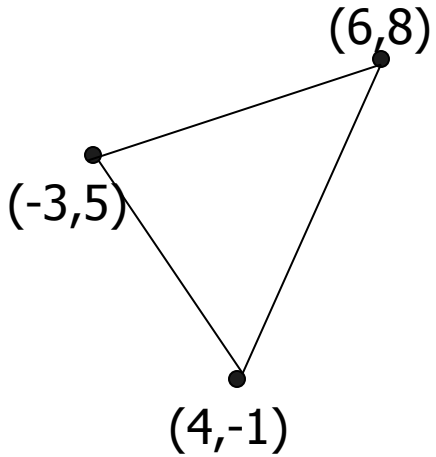
```
gcc prog.c -o prog -lm
```

Example: Programmer-defined Functions

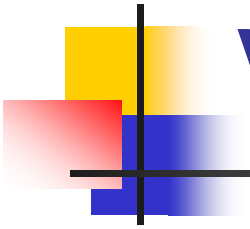
```
#include <stdio.h>

int main(void)
{
  double x1,y1,x2,y2, dist;
  printf("Enter x1 y1 x2 y2 :");
  scanf("%lf %lf %lf %lf",
        &x1,&y1,&x2,&y2);
  dist = sqrt(pow((x2-x1),2)
          + pow((y2-y1),2));
  printf("Distance is %lf\n",
   dist);
  return 0;
}
```

```
#include <stdio.h>
double distance(double x1,y1,x2,y2);
int main(void)
{
  double x1,y1,x2,y2, dist;
  printf("Enter x1 y1 x2 y2 :");
  scanf("%lf %lf %lf %lf",
        &x1,&y1,&x2,&y2);
  dist = distance(x1,y1,x2,y2);

  printf("Distance is %lf\n", dist);
  return 0;
}
double distance(double x1,y1,x2,y2)
{
  return  sqrt(pow((x2-x1),2)
          + pow((y2-y1),2));
}
```

Exercise

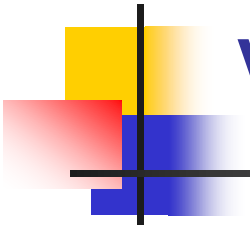


- Suppose you are given the coordinate points of a triangle as shown above, write a program that can find the length of each edge…



Value Returning Functions

- Function *returns* a single value to the calling program
- Function definition declares the type of value to be returned
- A **return** *expression;* statement is *required* in the function definition
- The value returned by a function can be assigned to a variable, printed, or used in an expression



Void Functions

- A void function may be called to
  - perform a particular task (clear the screen)
  - modify data
  - perform input and output
- A void function does not return a value to the calling program
- A return; statement can be used to exit from function without returning any value

Exercise: void function

- Write a program to generate the following output?

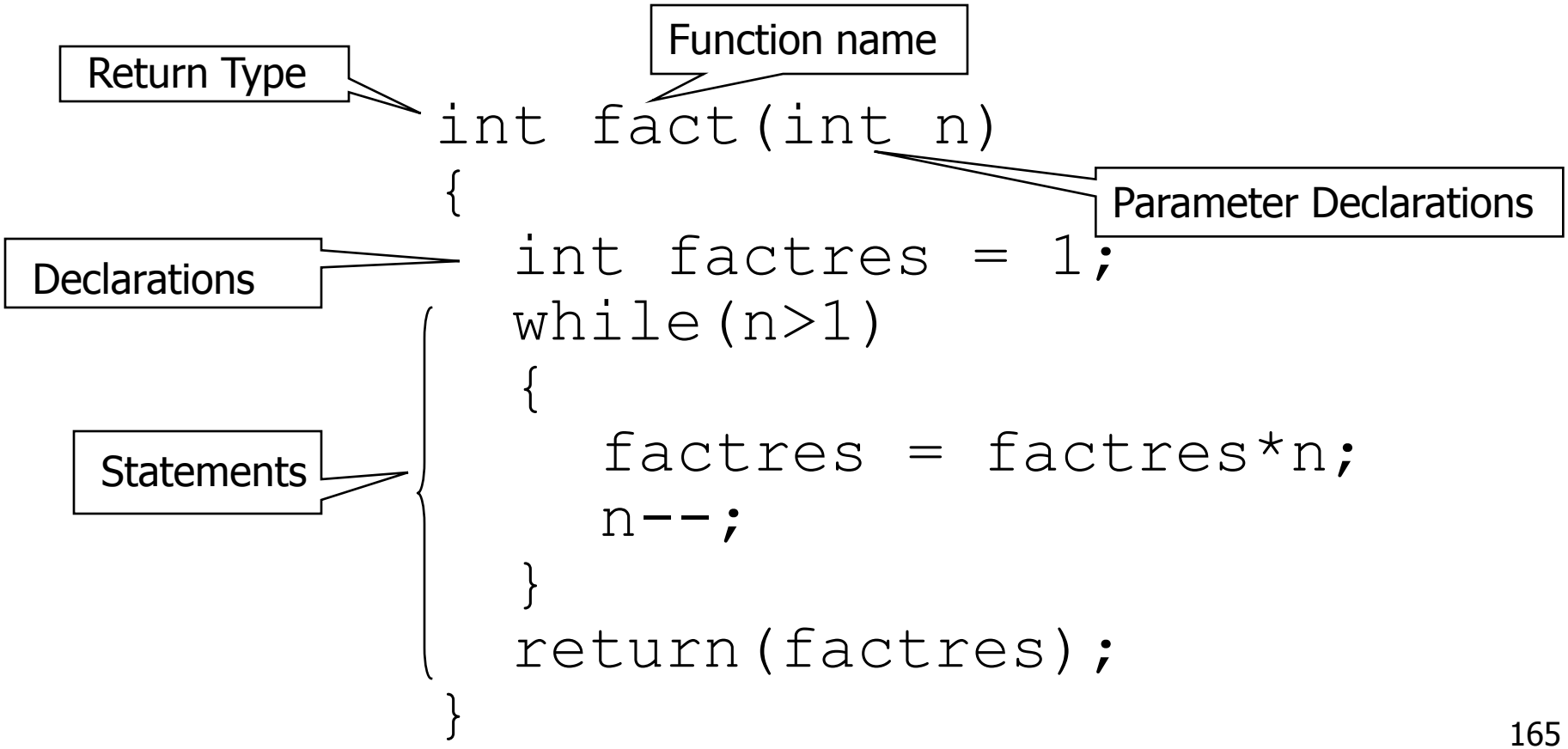
```
*
**
***
****
*****
```

```
for (i=1; i<=5; i++) {
  for (j=1; j<=i; j++)
      printf("*");
  printf("\n");
}
```

```
#include <stdio.h>
void print_i_star(int i);
main()
{
  int i;
  for (i=1; i<=5; i++) {
     print_i_star( i );
  }
}
void print_i_star(int i)
{
  int j;
  for (j=1; j<=i; j++)
      printf("*");
  printf("\n");
  return;
}
```

Example: value returning function

n!=n*(n-1)*…*1, 0! = 1 by definition



Example – use `fact()`

```
#include <stdio.h>
int fact(int n); /* prototype */

int main(void)
{
   int t= 5,s;

   s = fact(t) + fact(t+1);

   printf("result is %d\n", s);
   return 0;
}
```

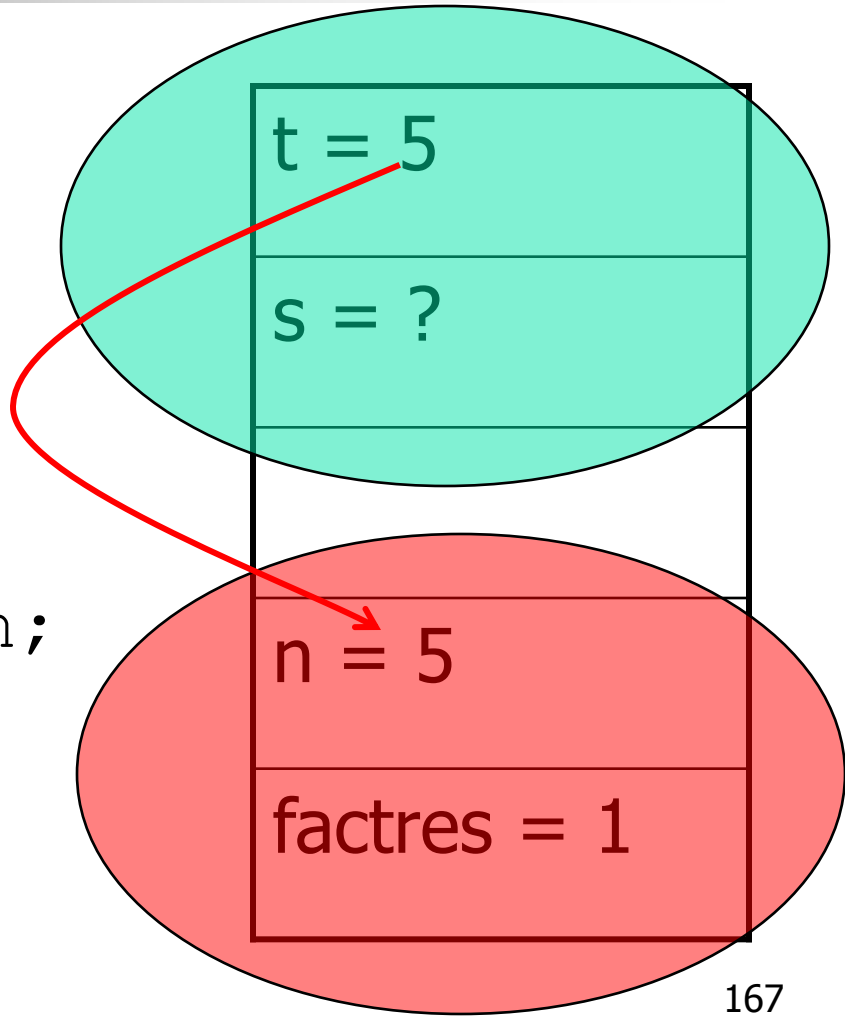
Function call

t = 5
s = ?

Example – execution of factorial function (cont'd)

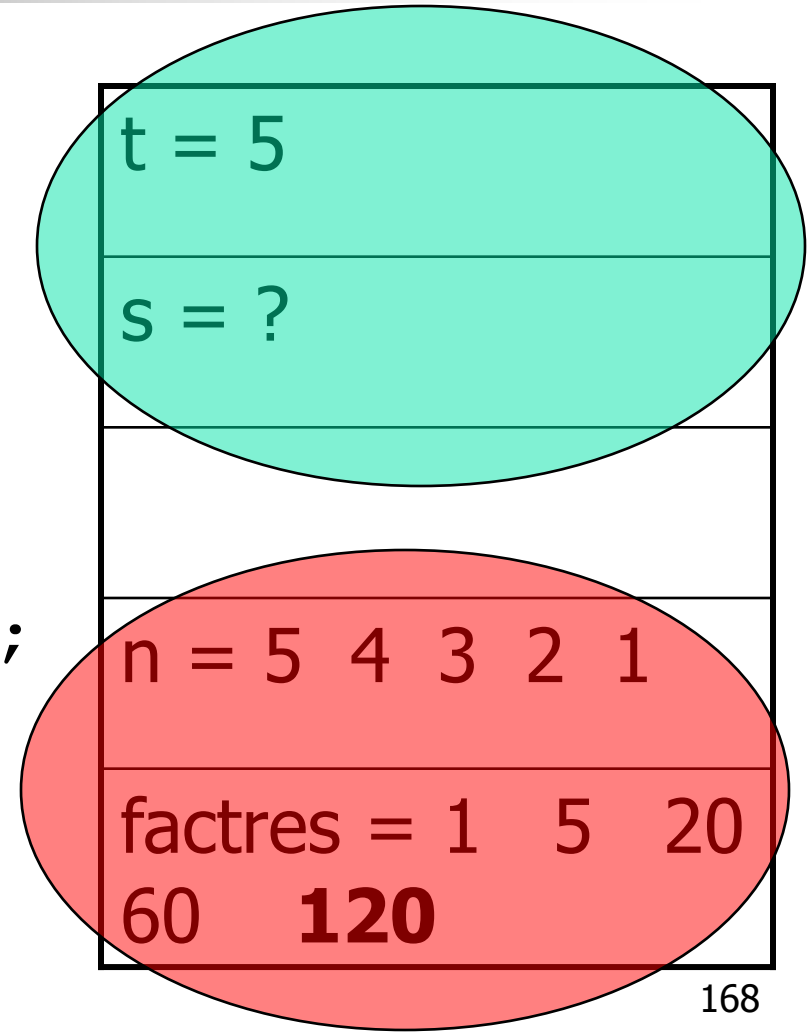
fact(5)

```
int fact(int n)
{
   int factres = 1;

 while(n>1)
   {
      factres = factres*n;
      n--;
   }
   return(factres);
}
```



Example – execution of factorial function (cont'd)

```
int fact(int n)
{
   int factres = 1;

 while(n>1)
   {
      factres = factres*n;
      n--;
   }
   return(factres);
}
```



Example – execution of factorial function (cont'd)

```
#include <stdio.h>
int fact(int n); /* prototype */

int main(void)
{
   int t= 5,s;

   s = 120 + fact(t+1);

   printf("result is %d\n", s);
   return 0;
}
```

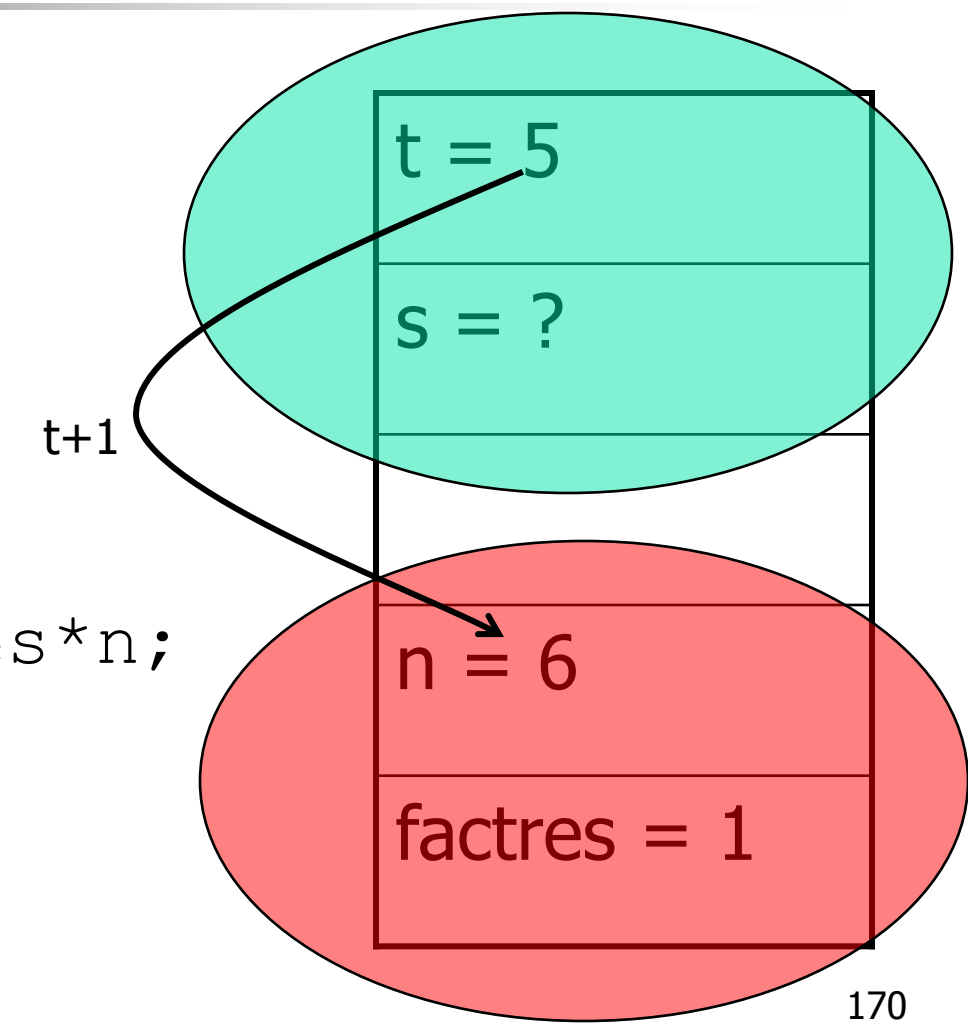
Function call

t = 5
s = ?

Example – execution of factorial function (cont'd)

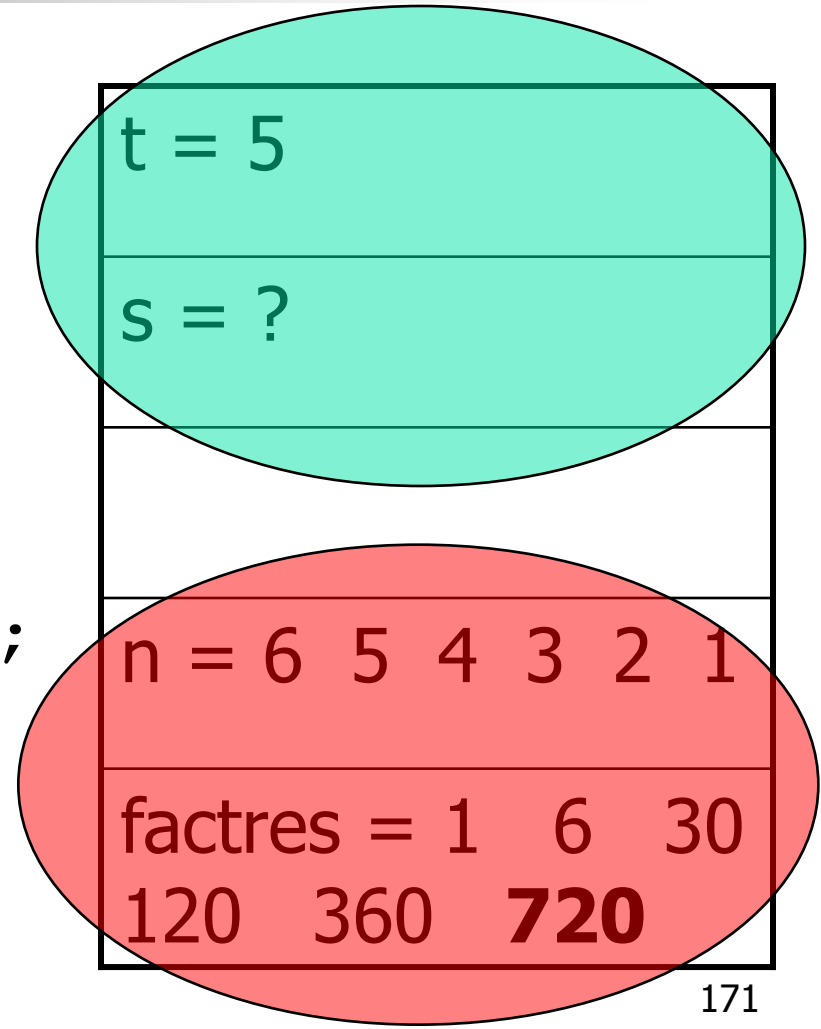
fact(6)

```
int fact(int n)
{
   int factres = 1;

 while(n>1)
   {
      factres = factres*n;
      n--;
   }
   return(factres);
}
```



Example – execution of factorial function (cont'd)

```
int fact(int n)
{
   int factres = 1;

 while(n>1)
   {
      factres = factres*n;
      n--;
   }
   return(factres);
}
```



Example – execution of factorial function (cont'd)

```
#include <stdio.h>
int fact(int n); /* prototype */

int main(void)
{
   int t= 5,s;

   s = 120 + 720;


   printf("result is %d\n", s);
   return 0;
}
```

t = 5
s = 840

result is 840

Example – reuse of factorial function

- Write a statement to compute

$$y = \frac{X!+Z!*5}{K!-D!}$$

Enter X, Z, K, D

…

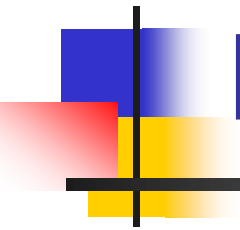
```
y=(fact(X)+fact(Z)*5)/(fact(K)-fact(D));
```

Example – reuse of factorial function in another function

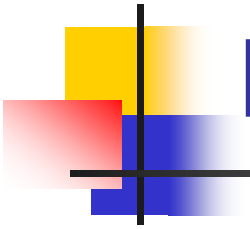
- Write a select function that takes n and k and computes "n choose k" where

$$\binom{n}{k} = \frac{n!}{(n-k)!\times k!}$$

```
int select(int n, int k)
{
   return fact(n)/(fact(n-k)*fact(k));
}
```



Parameter Passing



Parameter Passing

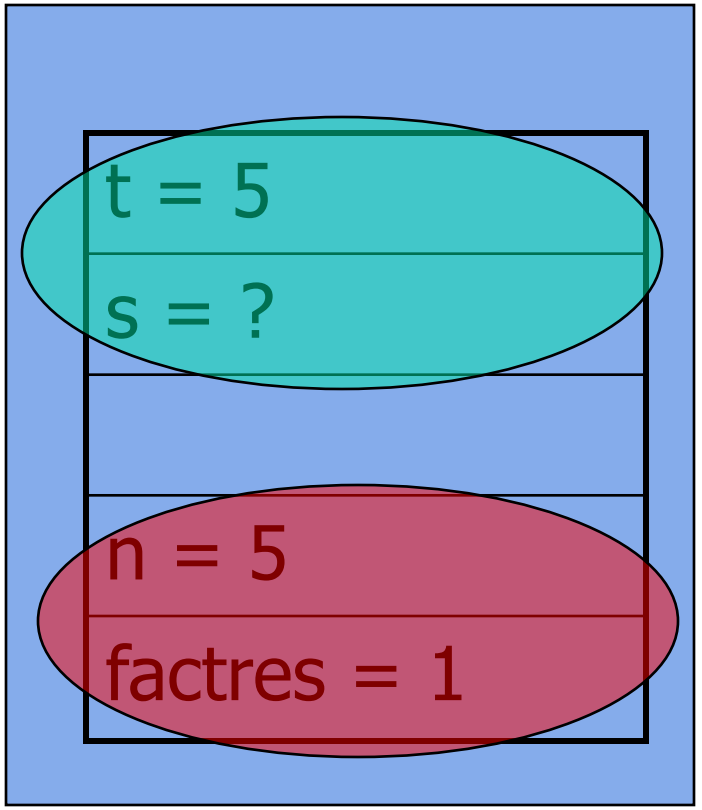
- Call by value
  - formal parameter receives the ***value*** of the actual parameter, as in the examples covered so far
  - function **CANNOT** change the value of the actual parameter (arrays are an exception)
- Call by reference
  - actual parameters are *pointers (ch 2)*
  - function can change the value of the actual parameter

Scope of a function or variable

- *Scope* refers to the portion of the program in which
  - It is valid to reference the function or variable
  - The function or variable is visible or accessible

```
#include <stdio.h>
int fact(int n); /* prototype */
int main(void)
{
  int t= 5,s;
  s = fact(t) + fact(t+1);
  printf("result is %d\n", s);
  return 0;
}
```

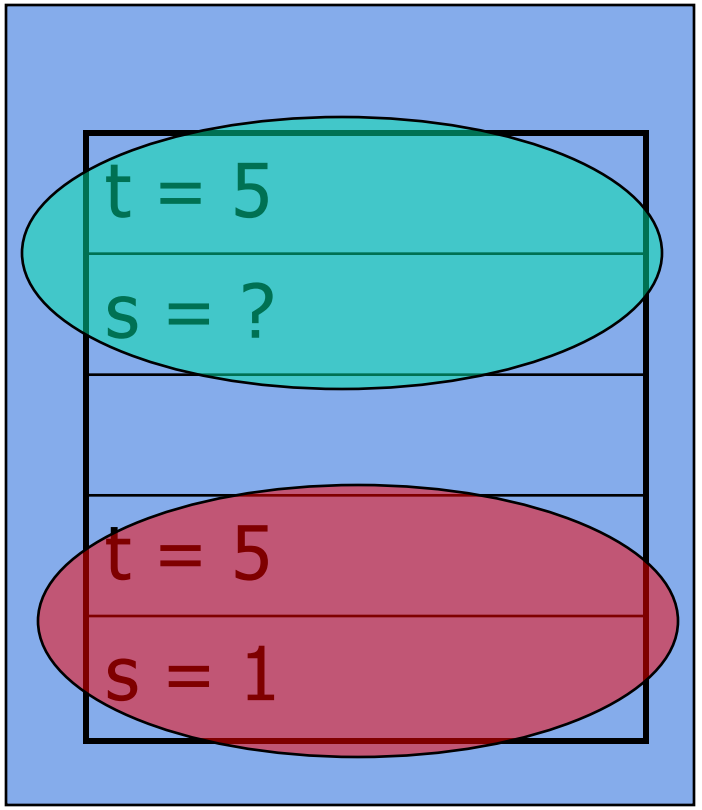
```
int fact(int n)
{
  int factres = 1;

 while(n>1) {
        factres = factres*n;
        n--;
  }
  return(factres);
}
```

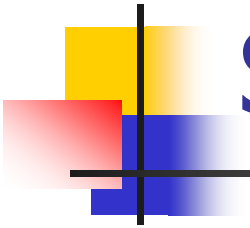


Scope of a function or variable

- *Same variable name can be used in different functions*

```
#include <stdio.h>
int fact(int n); /* prototype */
int main(void)
{
  int t= 5,s;
  s = fact(t) + fact(t+1);
  printf("result is %d\n", s);
  return 0;
}

int fact(int t)
{
  int s = 1;

 while(t>1) {
        s = s*t;
        t--;
  }
  return(s);
}
```



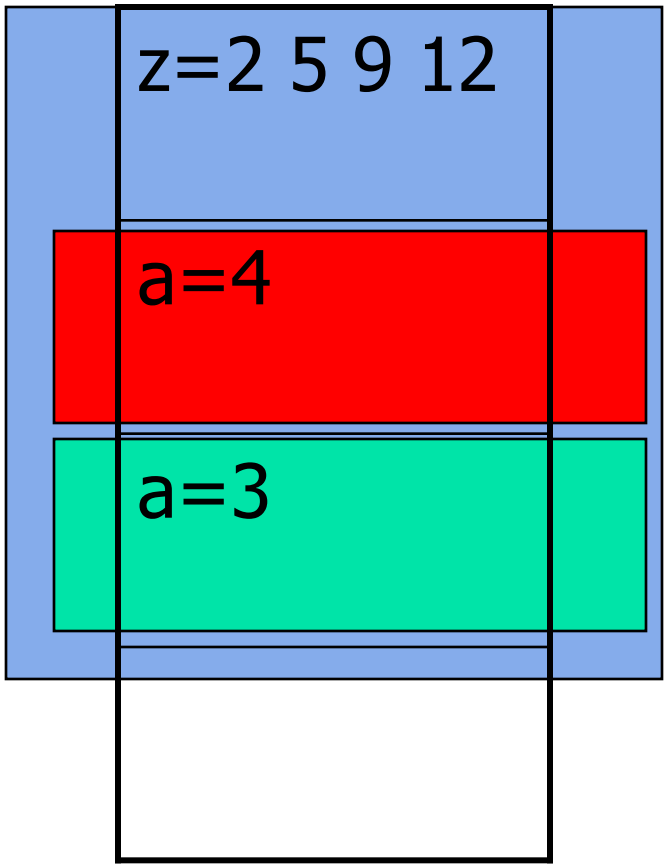


Scope

- *Local* scope
  - a local variable is defined within a function or a block and can be accessed only within the function or block that defines it
- *Global* scope
  - a global variable is defined outside the **main** function and can be accessed by any function within the program file.

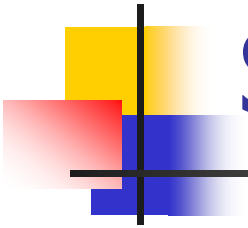
Global vs Local Variable

```
#include <stdio.h>
int z = 2;
void function1()
{ int a = 4;
  printf("Z = %d\n",z);
  z = z+a;
}

int main()
{ int a = 3;
  z = z + a;
  function1();
  printf("Z = %d\n",z);
  z = z+a;
  return 0;
}
```



Output

Z = 5
Z = 9



Storage Class - 4 types

Storage class refers to the lifetime of a variable

- *automatic* - key word **auto** - default for local variables
  - Memory set aside for local variables is not reserved when the block in which the local variable was defined is exited.
- *external* - key word **extern** - used for global variables
  - Memory is reserved for a global variable throughout the execution life of the program.
- *static* - key word **static**
  - Requests that memory for a local variable be reserved throughout the execution life of the program. The static storage class does not affect the scope of the variable.
  - If you want private use of variable/function in a source file…
  - Limit the scope of variable/function to other source files
- *register* - key word **register**
  - Requests that a variable should be placed in a high speed memory register because it is heavily used in the program.

Can be skipped

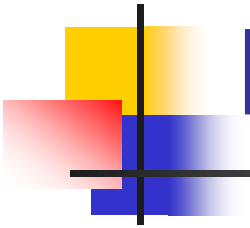
Recursive Functions

- A function that invokes itself is a recursive function.

```
int fact(int k)
{
   if (k == 0)
      return 1;
   else
      return k*fact(k-1);
}
```

k!=k*(k-1)!

More later

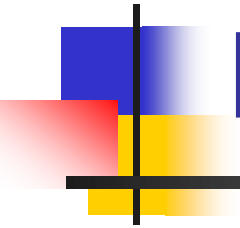


Macros

- #define macro_name(parameters) macro_text
- macro_text replaces macro_name in the program

- Examples
  - #define PI 3.14
  - #define area_tri(base,height) (0.5*(base)*(height))
  - #define max(A, B) (((A) > (B)) ? (A) : (B))

  - k=2*PI*r; → k=2*3.14*r;
  - z=x * area_tri(3, 5) + y; → z=x * (0.5*(3)*(5)) + y;
  - m=max(p+q, r+s); → m = (((p+q) > (r+s)) ? (p+q) : (r+s));



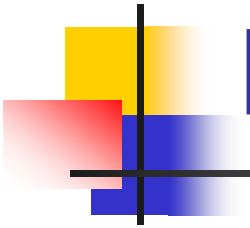
More Function Examples

Exercise

- Write a function to compute maximum and minimum of two numbers

```
int max(int a, int b)
{
   if (a > b)
      return a;
   else
      return b;
}
```

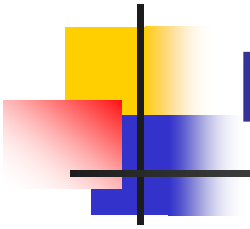
```
int min(int a, int b)
{
   if (a < b)
      return a;
   else
      return b;
}
```



Exercise

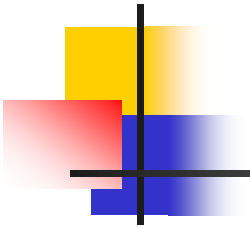
- Are following calls to max function valid?
- What will be the result?

```
int max(int a, int b);
int min(int a, int b);
int main()
{
  int x = 2, y = 3, z = 7, temp;
  temp = max(x,y);
  temp = max(4,6);
  temp = max(4,4+3*2);
  temp = max(x,max(y,z));
}
```



Example for void function

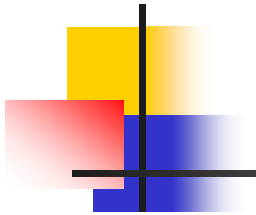
```
void print_date(int mo, int day, int year)
{
  /*output formatted date */
  printf("%i/%i/%i\n", mo, day, year );
  return;
}
```



Exercise

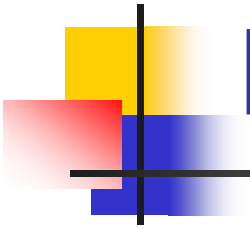
- Write a function that takes score as parameter and computes and returns letter grade based on the scale below.

80-100	A
60-79	B
40-59	C
0-39	D



Solution

```
char get_letter_grade(int score)
{
     char grade;
     if ((score >= 80) && (score <=100))
        grade = 'A';
     else if ((score >= 60) && (score <= 79))
        grade = 'B';
     else if ((score >= 40) && (score <= 59))
        grade = 'C';
     else if ((score >= 0) && (score <= 39))
        grade = 'D';
     return grade;
}
```



Exercise

- Write a function to compute $\log_b a$

$$\log_b a = \frac{\log_{10} a}{\log_{10} b}$$

```
double log_any_base(double a, double b)
{
    return log(a)/log(b);
}
```

Exercise: Trace functions

- What is the output of the following program

```
#include <stdio.h>
int function1(int x)
{
   x = 2;
   printf("Out1 = %d\n",x);
   return(x+1);
}
int main()
{
   int x = 4, y;
   y = function1(x);
   printf("Out2 = %d\n",x);
   printf("Out3 = %d\n",y);
   return 0;
}
```

Output

Out1 = 2
Out2 = 4
Out3 = 3

Exercise

- What is the output of the following program

```
#include <stdio.h>

void function2()
{
  printf("In function 2\n");
}

void function1()
{
  function2();
  printf("In function 1\n");
}
```

```
void function3()
{
  printf("In function 3\n");
  function2();
}

int main()
{
  function1();
  function3();
  return 0;
}
```

Output

In function 2
In function 1
In function 3
In function 2