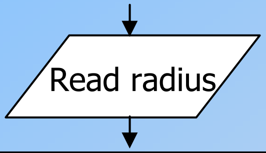
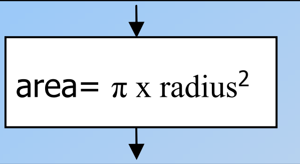
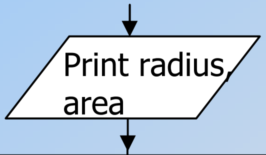
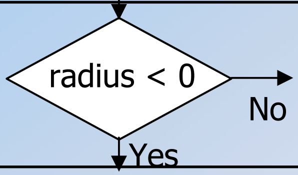
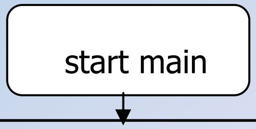
BIL 104E Introduction to Scientific and Engineering Computing

Lecture 5

Algorithm Development

- **Decomposition outlines** provide the first definition of a problem solution.
- This outline is written sequential steps and can be shown in a diagram or a step-by-step outline.
- For very simple problems we can go from the decomposition outline directly to the C statements.
- However, for most problems the decomposition outline needs to be refined into a description with more detail by breaking the problem solution into smaller and smaller portions.
- The refinement of an outline into more detailed steps can be done with pseudocode or a flowchart.
- **Pseudocode** uses English-like statements to describe the steps in an algorithm; a **flowchart** uses a diagram to describe the steps in an algorithm.

Algorithm Development

Basic Operation	Pseudocode Notation	
Input	Read radius	
Computation	Set area to π x radius^2	
Output	Print radius, area	
Comparisons	if radius < 0 then …	
Beginning of algorithm	main:	
End of algorithm		

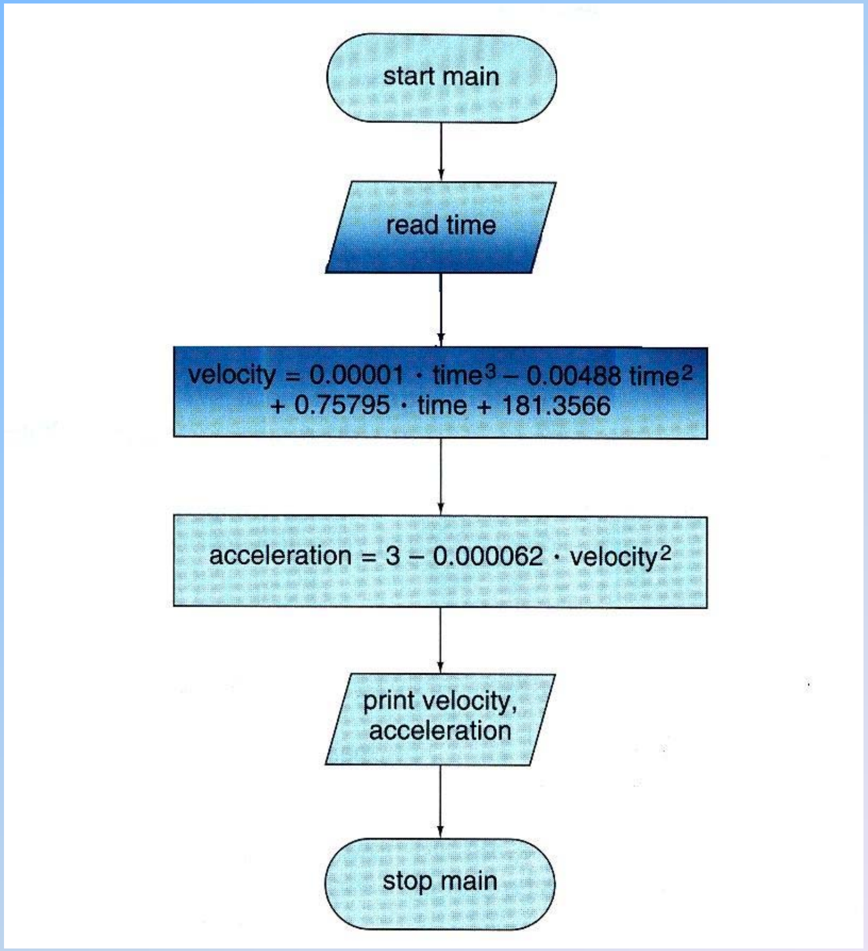
10.03.2007

Structured Program

- A **structured program** is one written using simple control structures,
  - sequence,
  - selection
  - repetition

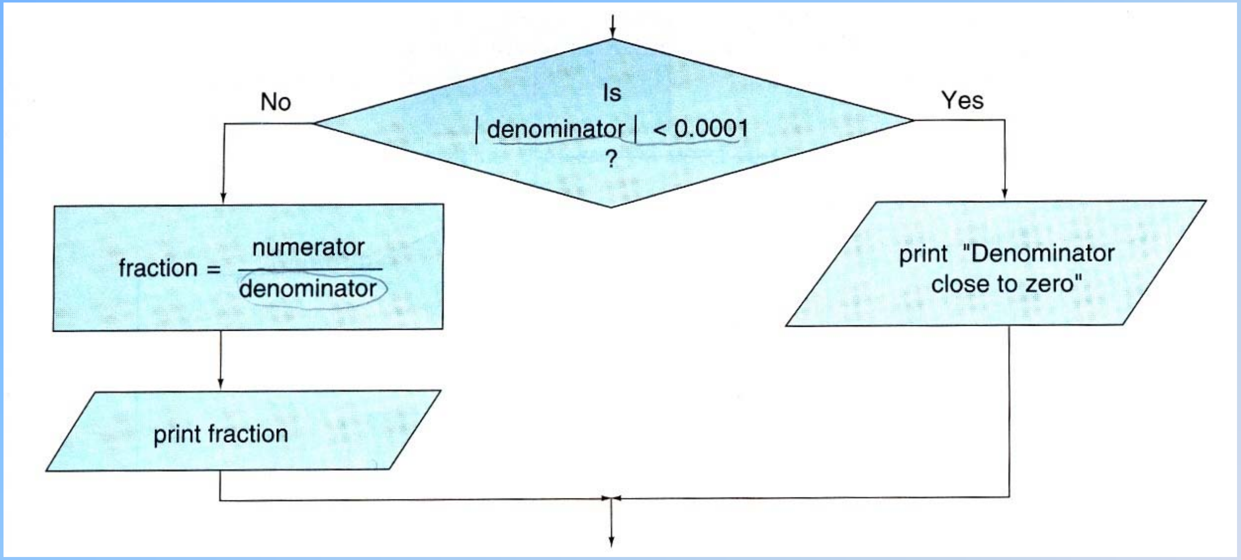
  to organize the solution into a problem.

- A **sequence structure** contains steps that are performed one after another. Figure on the right shows an example of sequence structure.


start main
read time
velocity = 0.00001 · time³ – 0.00488 time² + 0.75795 · time + 181.3566
acceleration = 3 – 0.000062 · velocity²
print velocity, acceleration
stop main

Structured Program

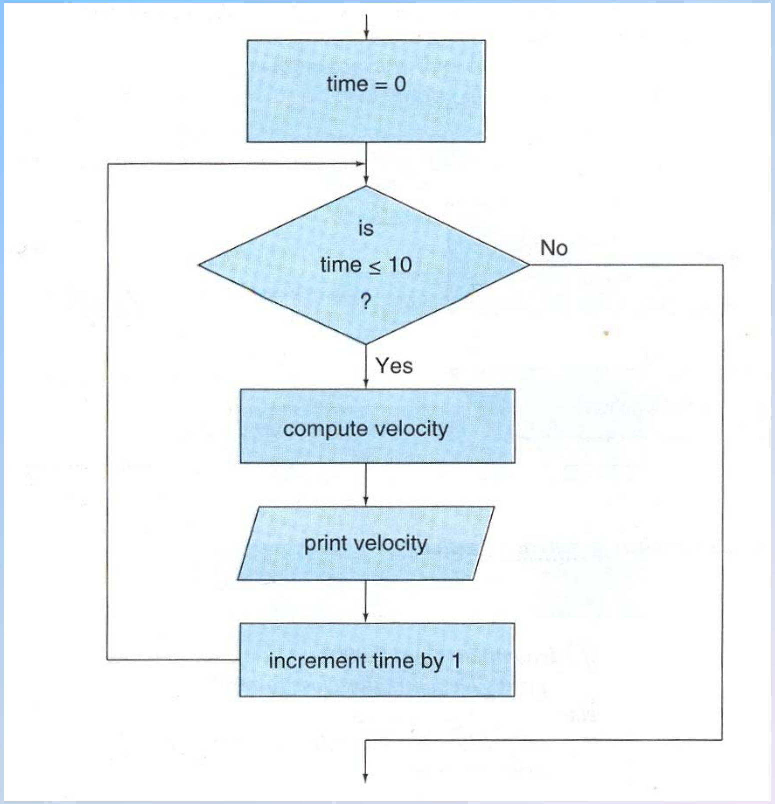
- A **selection structure** contains one set of steps that is performed if a condition is true and another set of steps that is performed if the condition is false.


Is |denominator| < 0.0001 ?
No
Yes
fraction = numerator / denominator
print "Denominator close to zero"
print fraction

Structured Program

- A **repetition structure** contains a set of steps that is repeated as long as a condition is true.

```
set time to 0
while time≤10
        compute velocity
        print velocity
        increment time by 1
```



Conditional Expressions

- A **condition** is an expression that can be evaluated to be **true** or **false** and is composed of expressions combined with **relational operators**; a condition can also include **logical operators**.
- The **relational operators** can be used to compare two expressions.

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

Relational Operators

- **Examples:**

```
a<b
x+y >= 10.5
fabs (denominator) < 0.0001
```

- A **true** condition is assigned to a value of **1**; a **false** condition is assigned to a value of **zero**. So the following statement is valid:

```
d = b>c;   /*if b>c then d=1, else d=0 */
```

- Because a condition is given a value, it is valid to use a value in place of a condition

Logical Operators

- **Logical operators** can be used to compare conditions and generate new conditions.

  Example:

  a<b && b<c   (The relational operators have higher precedence than the logical operator.)

Logical Operator	**Symbol**
not	!
and	&&
or	||

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

Precedence

- A condition can contain several logical operators. The hierarchy from highest to the lowest is **!**, **&&**, **||**, but parenthesis can be used to change the hierarchy.

Precedence	**Operator**	**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

if Statement

- **General Form:** if the condition is true then statement 1 is executed. If the condition is false then statement 1 is skipped.

```
if (condition)
        statement 1;
```

- A **compound statement** or **block** (which is composed of a set of statements enclosed in braces) can also be used.

```
if (condition)
{
        statement 1;
        statement 2;
        …
        statement n;
}
```

Example

```
if (a < 50)
{
    ++count;
    sum += a;
}
```

```
/* Using if statements, relational operators, and equality operators */

#include <stdio.h>
void relationship (int num1, int num2);

int main()
{
     int x, y;
     printf( "Enter two integers, and I will tell you\n" );
     printf( "the relationships they satisfy:\n" );
     scanf( "%d%d", &x, &y );
     relationship(x,y);
     return 0;
}

void relationship (int num1, int num2)
{
     if ( num1 == num2 ) printf( "%d is equal to %d\n", num1, num2 );
     if ( num1 != num2 )  printf( "%d is not equal to %d\n", num1, num2 );
     if ( num1 < num2 )   printf( "%d is less than %d\n", num1, num2 );
     if ( num1 > num2 )   printf( "%d is greater than %d\n", num1, num2 );
     if ( num1 <= num2 ) printf( "%d is less than or equal to %d\n", num1, num2 );
     if ( num1 >= num2 ) printf( "%d is greater than or equal to %d\n", num1, num2 );
     return;
}
```

if Statement

- if statements can also be nested:

Example:

```
if (a < 50) /*outer if statement */
{
        ++count;
        sum += a;
        if (b > a) /*inner if statement */
                b = 0;
}
```

Review Problem 1

Write a C program which finds the maximum of three integers which are read from standard input.

```
/* Finding the maximum of three integers */
#include <stdio.h>

int maximum( int x, int y, int z ); /* function prototype */

int main()
{
    int number1, number2, number3;

    printf( "Enter three integers: " );
    scanf( "%d%d%d", &number1, &number2, &number3 );
    printf( "Maximum is: %d\n", maximum( number1, number2, number3 ) );
    return 0;
}

int maximum( int x, int y, int z )
{
    int max = x;      /* assume x is largest */

    if ( y > max )    /* if y is larger than max, assign y to max */
            max = y;
    if ( z > max )    /* if z is larger than max, assign z to max */
            max = z;
    return max;       /* max is largest value */
}
```

if/else Statement

- The if/else statement allows to execute one set of statements if a condition is true and a different set if the condition is false.

- **General Form:**

```
if (condition)
  statement1;
else
  statement2;
```

```
if ( grade >= 60 )
     printf( "Passed.\n" );
else {
     printf( "Failed.\n" );
     printf( "You must take this course again.\n" );
}
```

- Statements 1 and 2 can also be an empty statement, which is a semicolon.

```
if (a < b)
        ;
else
        count++;
```

is equivalent to

```
if (a  >= b)
           count++;
```

Review Problem 2

Write a C program which finds the maximum of two integers which are read from standard input.

```c
/* Finding the larger of two integers */
#include <stdio.h>

int larger_of( int x, int y); /* function prototype */

int main()
{
     int number1, number2, larger_number;

     puts( "Enter two different integer numbers: " );
     scanf( "%d%d", &number1, &number2 );
     larger_number= larger_of( number1, number2);
     printf( "Larger value is: %d\n",larger_number);
     return 0;
}

int larger_of( int x, int y)
{
     if ( x > y )
          return x;
     else
          return y;
}
```

if/else Statement

- **Example (nested if/else):**

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

is equivalent to

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

if/else Statement

Indentation is just for style and it does not change interpretation:

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

is equivalent to

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

Use blocking to change interpretation:

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

if/else Statement

- In general do not use equality operator when comparing floating-point numbers. For example, instead of comparing denominator to zero, the absolute value of the denominator can be compared with a very small value.

- **Example:**

```
if (fabs(denominator) < 0.0001)
            printf("Denominator of x is equal to zero\n");
      else
      {
            fraction = numerator/denominator;
            printf("fraction = %f \n",fraction);
      }
```

else-if Statement

- If's and else's can be used to construct logic that branches one of several ways and then rejoins, a common programming structure, in this way:

```
if (...)
        {...}
else if (...)
        {...}
else if(...)
        {...}
else
        {...}
```

- The conditions are tested in order, and exactly one block is executed; either the first one whose if is satisfied, or the one for the last else. When this block is finished, the next statement executed is the one after the last else. If no action is to be taken for the ``default'' case, omit the last else.

```
/* else – if example */

#include<stdio.h>

int main( )
{
     int c;
     c=getchar( );

     if( ('A'<=c && c<='Z') || ('a'<=c && c<='z') )
              printf("You entered a letter\n");

     else if( '0'<=c && c<='9' )
              printf("You entered a digit\n");

     else
              printf("You entered a non-alphanumeric character\n");

     return 0;
}
```

Ternary conditional operator (?:)

- Takes three arguments (condition, value if true, value if false)
- **Example**:

```
if ( grade >= 60 )
        printf( "Passed\n");
else
        printf( "Failed\n");
```

is equivalent to:

```
grade >= 60  ?  printf( "Passed\n” )    :  printf( "Failed\n” ) ;
```

OR

```
printf( "%s\n", grade >= 60  ?  "Passed"  :  "Failed" ) ;
```

Switch-Case

- The switch statement can be used to make unlimited decisions or choices based on the value of a conditional expression and specified cases. The general form:

```
switch (expression)
{
case expression1: statement1;
case expression2: statement2;
. . .
default: statement-default;
}
```

```
char ch;
ch=getch();
switch (ch) {
     case('1'): printf("You pressed 1");
     case('2'):
     case('3'): printf("You pressed 1 or 2 or 3");
                break;
     case('4'): printf("You pressed 4");
     default:   printf("You did not press 1, 2, 3 or 4");
 }
```

```
#include<stdio.h>

int main( )
{
     int day;
     printf("Enter a number between 1 and 7 and I will tell you the day it
     corresponds.\n");
     scanf("%d",&day);

     switch (day){
               case 1:  printf("Sunday\n");    break;
               case 2:  printf("Monday\n");    break;
               case 3:  printf("Tuesday\n");   break;
               case 4:  printf("Wednesday\n"); break;
               case 5:  printf("Thursday\n"); break;
               case 6:  printf("Friday\n");      break;
               case 7:  printf("Saturday\n");  break;
               default:  printf("You made an invalid entry!\n"); break;
     }

     return 0;
}
```

```
/* What happens if we do not use breaks??? */

#include<stdio.h>

int main( )
{
    int day;
    printf("Enter a number between 1 and 7 and I will tell you the day it corresponds and
    the remaining days of the week.\n");
    scanf("%d",&day);

    switch (day){
            case 1:  printf("Sunday\n");
            case 2:  printf("Monday\n");
            case 3:  printf("Tuesday\n");
            case 4:  printf("Wednesday\n");
            case 5:  printf("Thursday\n");
            case 6:  printf("Friday\n");
            case 7:  printf("Saturday\n");  break;
            default:  printf("You made an invalid entry!\n");
    }

    return 0;
}
```