Chapter 5: Statements and Control Flow

Outline
Simple and Compound Statements
Expression and Null Statements
Flowcharts
Selection Statements
Sample Problem
Iteration Statements
Jump Statements
Labeled Statements
Sample Problem
Random Number Generation
Simple and Compound Statements

- A *compound statement* is a block of code enclosed with a pair of braces.
- A *block* allows a set of declarations and statements to be grouped into one syntactic unit.
- The initialization is performed for an initialization statement each time the declaration is reached in the order of execution.

Example:

```c
int i = 10;  // simple statement
{
    int i;
    i = 90;
    ...
}
printf("%d", i);
```
Expression and NULL Statements

- An expression statement contains an expression only. For example, 
  \( \text{i*7+4;} \)

- A *null statement* consisting of just a semicolon performs no operation.

  /* … */

  ;

  /* … */
Flowcharts

- Often, programmers use *flowcharts* to organize the order in which actions are to be performed.
- Common flowchart symbols are shown on the next slide.
Common Flowchart Symbols

- **Start**: Indicates the beginning of the program.
- **Input**: Represents a user input.
- **Computation**: Performs a calculation, e.g., \( p = 4 \times (t+2) \).
- **Comparison**: Checks a condition, e.g., \( t < 3 \).
- **True/False**: Determines the flow based on the comparison result.
- **Output**: Displays the result, e.g., "print \( p \)".
- **Stop**: Marks the end of the program.
Review of what we know

We covered the basics of these structures:

- if statements
- if-else statements
- for loops
- while loops
- do-while loops
- switch statements
Example Programs

• Test your if-else skills with some simple problems
Sample Problem

The system in Figure 1 (a) consists of a single body with mass \( m \) moving on a horizontal surface. An external force \( p \) acts on the body. The coefficient of kinetic friction between body and horizontal surface is \( \mu \). The freebody diagram for the system is shown in Figure 1 (b).

![System Diagram and Free Body Diagram](image)

Figure 1: The system diagram and FBD of a sample problem
The nomenclature related to the modeling of the system is listed below.

- \( m \) -- mass of the body
- \( x \) -- position of the body
- \( v \) -- velocity of the body
- \( a \) -- acceleration of the body
- \( g \) -- gravitational acceleration
- \( \mu \) -- friction coefficient
- \( f \) -- friction force
- \( N \) -- normal force

**Equation of motion:**

The equation of the motion of the system can be derived based on the Newton's second law.

\[
N = mg \quad (1) \\
f = \mu N \quad (2) \\
p - f = ma \quad (3)
\]

From equation (1), (2) and (3), the formula for calculating the acceleration of the rigid body can be derived as follows.

\[
a = \frac{p - \mu mg}{m} \quad (4)
\]
**Problem Statement:**

For the system shown in Figure 1(a), given $m = 5 \text{ kg}$, $g = 9.81 \text{ m/s}^2$, $\mu = 0.2$. The external force $p$ is expressed as a function of time $t$,

$$p(t) = 20, \quad \text{if } t \leq 3 \text{ seconds}$$
$$p(t) = 4(t-3)+20, \quad \text{if } t > 3 \text{ seconds}.$$

Calculate the force $p$ based on the time $t$ input by the user.

A flowchart illustrating the flow of control for program force.c is on the next slide. The C program of the problem is listed on the following slide.
Flowchart of Program force.c

1. Start
2. Read t
3. If t <= 3 then:
   - If false then:
     - p = 4 * (t+2)
     - Print p
   - If true then:
     - p = 20
4. Stop
C for Engineers and Scientists: An Interpretive Approach

/* File: force.c */
#include <stdio.h>

int main() {
    double p, t;

    printf("Please input time in seconds \n");
    scanf("%lf", &t);

    if(t <= 3) {
        p = 20.0;
    } else {
        p = 4*(t+2);  // simplified formula
    }

    printf("Force p = %f (N) \n", p);
    return 0;
}

Program Output:

> force.c
Please input time in seconds
4
Force p = 24.000000 (N)
Problem Statement:

For the system shown in Figure 1(a), given $m = 5 \text{ kg}$, $g = 9.81 \text{ m/s}^2$, $\mu = 0.2$. The external force $p$ is expressed as a function of time $t$,

$$p(t) = 20, \quad \text{if} \quad 0 \leq t \leq 3 \text{ seconds}$$

$$p(t) = 4(t-3)+20, \quad \text{if} \quad t > 3 \text{ seconds}.$$  

Calculate the force $p$ based on the time $t$ input by the user.

The formula for the external force $p(t)$ is changed in comparing with the previous program statement.

A flowchart illustrating the flow of control for program force2.c is on the next slide. The C program of the problem is listed on the following slide.
Flowchart of Program force2.c

1. Start
2. Read t
3. If 0 ≤ t ≤ 3, then p = 20; else go to t > 3
4. If t > 3, then p = 4(t+2); else go to print error message
5. Print p
6. Stop
Note:
- \((0 \leq t \&\& t \leq 3)\)
- \((0 \leq t \leq 3)\)
- \((0 \leq t \leq 3)\)

Program Output:

```plaintext
> force2.c
Please input time in seconds
4
Force p = 24.000000 (N)
```

```c
#include <stdio.h>

int main() {
    double p, t;

    printf("Please input time in seconds \n");
    scanf("%lf", &t);

    if(0 <= t && t <= 3){
        p = 20.0;
        printf("Force p = %f \n", p);
    }
    else if (t>3){
        p = 4*(t+2);
        printf("Force p = %f \n", p);
    }
    else {
        printf("Invalid input \n");
    }

    return 0;
}
```
Example Program

- Calculate score based upon a letter grade
- Get input from user: A, B, C, D, F
- Check for valid input
- Print Score in a 4.0 to 0.0 scale
- Use if/else-if statements
/* File: ifgrade.c */
#include <stdio.h>
int main() {
    char grade; /* grade */
    double score; /* score */
    printf("Enter a grade [A, B, C, D, F]: ");
    scanf("%c", &grade);

    if(grade == 'A') /* entered A */
        score = 4.0;
    else if(grade == 'B') /* entered B */
        score = 3.0;
    else if(grade == 'C') /* entered C */
        score = 2.0;
    else if(grade == 'D') /* entered D */
        score = 1.0;
    else if(grade == 'F') /* entered F */
        score = 0.0;
    else { /* entered any other character */
        score = -1;
        printf("Invalid grade '%c'\n", grade);
    }

    if(score != -1)
        printf("The score for the grade '%c' is %.2f\n", grade, score);

    return 0;
}

Calculate the score based on a grade.

Fill this part in
Example Program

• Test your switch statement skills with a simple problem

• Redo the grade to score program using a switch statement
Calculate the score based on a grade.

```c
/* File: switchgrade.c */
#include <stdio.h>
int main() {
    char grade;      /* grade */
    double score;    /* score */
    printf("Enter a grade [A, B, C, D, F]: ");
    scanf("%c", &grade);

    switch(grade) {
        case 'A': /* entered A */
            score = 4.0;
            break;
        case 'B': /* entered B */
            score = 3.0;
            break;
        case 'C': /* entered C */
            score = 2.0;
            break;
        case 'D': /* entered D */
            score = 1.0;
            break;
        case 'F': /* entered F */
            score = 0.0;
            break;
        default: /* entered any other character */
            score = -1;
            printf("Invalid grade '%c'\n", grade);
            break;
    }

    if(score != -1)
        printf("The score for the grade '%c' is %.2f\n", grade, score);

    return 0;
}
```

Fill this part in
Example Programs

• Test your while and for statement skills with some simple problems
A Typical Program Structure in C

A typical C program consists of four sections:

Declaration
declaration of variables.

Initialization
initialization of variables.

Processing
Data processing and computation.

Termination
Output the result.
Calculate n!

• The factorial of a given integer n is defined to be:

\[ n! = n \times (n-1) \times (n-2) \times \ldots \times 1 \]

or

\[ n! = n \times (n-1)! \]
The flowchart for program whilec.c
Example:
Calculating a factorial 5!. The factorial n! is defined as n*(n-1)!

```c
/* File: whilec.c */
#include <stdio.h>

int main() {
    /* declaration */
    unsigned int i, f, n;

    /* initialization */
    i = 1;
    f = 1;

    /* processing */
    printf("Enter a number\n");
    scanf("%u", &n);
    while (i <= n) {
        f *= i;
        i++;
    }

    /* termination */
    printf("factorial %d! = %d\n", n, f);
    return 0;
}
```

Execution and Output:

```
> whilec.c
Enter a number
5
factorial 5! = 120
```

Fill this part in
Control of Repetition

- Counter-controlled repetition
- Sentinel-controlled repetition.
Counter-controlled repetition

- Loop repeated until counter reaches a certain value.
- Definite repetition: number of repetitions is known

**Example:**

A student takes *four* courses in a quarter. Each course will be assigned a grade with the score from 0 to 4. Develop a C program to calculate the grade point average (GPA) for the quarter.
The flowchart for program gpa.c

1. Start
2. Initialize variables: NUM, count, total = 0
3. While count < NUM, repeat:
   - Read grade
   - Update total = total + grade
   - Update count = count + 1
4. Calculate GPA: GPA = total / NUM
5. Print GPA
6. Stop
Execution and Output:

> gpa.c
Enter a grade: 4
Enter a grade: 3.7
Enter a grade: 3.3
Enter a grade: 4
The GPA is: 3.750000

/* File: gpa.c */
#include <stdio.h>

#define NUM 4

int main() {
    /* declaration */
    int count;
    double grade, total, gpa;

    /* initialization */
    count = 0;
    total = 0;

    /* processing */

    /* termination */
    gpa = total/NUM;
    printf("The GPA is: \%f\n", gpa);
    return 0;
}
Sentinel-controlled repetition

- Loop repeated until the sentinel (signal) value is entered.
- Indefinite repetition: number of repetitions is unknown when the loop begins execution.

Example:

Develop a GPA calculation program that will process grades with scores in the range of [0, 4] for an arbitrary number of courses.
/* File: gpa2.c */
#include <stdio.h>

#define SENTINELNUM -1
#define HIGHESTMARK 4

int main() {
    /* declaration */
    int count;
    double grade, total, gpa;

    /* initialization */
    count = 0;
    total = 0;

    /* processing */
    printf("Enter a grade \[0, %d\] or %d to end: ",
            HIGHESTMARK, SENTINELNUM);
    scanf("%lf", &grade);

    Fill this part in
Execution and Output:

> gpa2.c
Enter a grade [0, 4] or -1 to end: 4
Enter a grade [0, 4] or -1 to end: 3.7
Enter a grade [0, 4] or -1 to end: 3.3
Enter a grade [0, 4] or -1 to end: 4
Enter a grade [0, 4] or -1 to end: 10
Invalid grade
Enter a grade [0, 4] or -1 to end: 3.7
Enter a grade [0, 4] or -1 to end: -1
The GPA is: 3.740000
A Mathematical Formula with a Single Variable

Calculate function

\[ \text{sinc}(x) = \frac{\sin(x)}{x} \]

from \(-10 \leq x \leq 10\) with a step size 5.

\(\text{sinc}(0) = \frac{\sin(0)}{0}\) is NaN inside a program. But, it can be proved that \(\sin(0)/0\) is 1 in calculus.

Generate data points \((x, y)\) for \(x\) in the range \(x_0 \leq x \leq x_f\) with step size \(x_{\text{step}}\) linearly. The number of points

\[ n = \frac{(x_f - x_0)}{x_{\text{step}}} + 1; \]

Each data point can be calculated by

\[
\text{for}(i = 0; \ i < n; \ i++) \ { }
\text{x} = x_0 + i \times x_{\text{step}}; \\
\text{y} = f(x);
\]
/* File: forsinc.c */
#include <stdio.h> /* for printf() */
#include <math.h>  /* for sin() and fabs() */
#include <float.h> /* for FLT_EPSILON */

int main() {
    double x, x0, xf, xstep, result;
    int i, n;

    printf("    x    sinc(x)\n");
    printf("    \n");
    printf("    \n");
    printf("    \n");

    x0 = -10.0;  /* initial value */
    xf = 10.0;   /* final value */
    xstep = 5.0; /* step size */
    n = (xf - x0)/xstep + 1; /* num of points */

    printf("  \n");

    for(i = 0; i < n; i++) {
        x = x0 + i * xstep;       /* value x */
        if(fabs(x) < FLT_EPSILON)
            result = 1.0;
        else
            result = sin(x)/x;
        printf("%8.4f %8.4f\n", x, result);
    }

    return 0;
}
Note:
1. $\sin(x)$ can be calculated using a function declared in header file `math.h`.
2. Use a loop control variable of int type.
3. Use format specifier "%8.4" with field width 8 and 4 digits after the decimal point.
Nested Loop

For nested loops, the inner loops must be finished before the outer loop resume iteration.
Write a program to print a multiplication table.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>6</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>8</td>
<td>12</td>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>25</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>12</td>
<td>18</td>
<td>24</td>
<td>30</td>
<td>36</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>14</td>
<td>21</td>
<td>28</td>
<td>35</td>
<td>42</td>
<td>49</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>16</td>
<td>24</td>
<td>32</td>
<td>40</td>
<td>48</td>
<td>56</td>
<td>64</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>18</td>
<td>27</td>
<td>36</td>
<td>45</td>
<td>54</td>
<td>63</td>
<td>72</td>
<td>81</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>20</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td>60</td>
<td>70</td>
<td>80</td>
<td>90</td>
<td>100</td>
</tr>
</tbody>
</table>
Example:

A program to print a multiplication table.

```c
/* File: nestedloop.c */
#include <stdio.h>

int main() {
    int i, j;
    printf("1 2 3 4 5 6 7 8 9 10\n");
    printf("--------------------------\n");
    for(i=1; i<=10; i++) {        /* outer loop */
        printf("%4d|", i);
        for(j=1; j<=i; j++) {       /* inner loop */
            printf("%4d", i*j);
        }
        printf("\n");
    }
    printf("--------------------------\n");
    return 0;
}
```

Fill this part in

```c
printf("--------------------------\n");
return 0;
```
A Mathematical Formula with Multiple Variables

Calculate function

\[ f_2(x, y) = \frac{\sin(\sqrt{x^2 + y^2})}{\sqrt{x^2 + y^2}} \]

for \( x \) from \(-10 \leq x \leq 10\) with a step size 10 and
\( y \) from \(-10 \leq y \leq 10\) with a step size of 10.
/ * File: forsinr.c */
#include <stdio.h>
#include <math.h>

int main() {
    double x, x0, xf, xstep,
            y, y0, yf, ystep, result;
    int i, j, nx, ny;

    printf("      x          y     sinr(x,y)\n");
    printf("     --------------------------\n");
    x0 = -10.0;
    xf = 10.0;
    xstep = 10.0;
    nx = (xf - x0)/xstep + 1; /* num of points for x */
    y0 = -10.0;
    yf = 10.0;
    ystep = 10.0;
    ny = (yf - y0)/ystep + 1; /* num of points for y */
```c
for (i = 0; i < nx; i++) {
    x = x0 + i * xstep;        /* calculate value for x */
    for (j = 0; j < ny; j++) {
        y = y0 + j * ystep;     /* calculate value for y */
        r = sqrt(x * x + y * y);
        result = sin(r) / r;
        printf("%10.4f %10.4f %8.4f
", x, y, result);
    }
}
return 0;
```

Fill this part in

### Output:

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>sinr(x, y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-10.0000</td>
<td>-10.0000</td>
<td>0.0707</td>
</tr>
<tr>
<td>-10.0000</td>
<td>0.0000</td>
<td>-0.0544</td>
</tr>
<tr>
<td>-10.0000</td>
<td>10.0000</td>
<td>0.0707</td>
</tr>
<tr>
<td>0.0000</td>
<td>-10.0000</td>
<td>-0.0544</td>
</tr>
<tr>
<td>0.0000</td>
<td>0.0000</td>
<td>NaN</td>
</tr>
<tr>
<td>0.0000</td>
<td>10.0000</td>
<td>-0.0544</td>
</tr>
<tr>
<td>10.0000</td>
<td>-10.0000</td>
<td>0.0707</td>
</tr>
<tr>
<td>10.0000</td>
<td>0.0000</td>
<td>-0.0544</td>
</tr>
<tr>
<td>10.0000</td>
<td>10.0000</td>
<td>0.0707</td>
</tr>
</tbody>
</table>

sin(0)/0 is NaN
**Note:**

1. Square root function `sqrt()` is declared in header file `math.h`.
2. Use nested loops for `x` and `y`. Each has its own loop control variable of `int` type.
Jump Statements

• Break Statements
  – The break statement provides an early exit from the for, while, do-while, and foreach loops as well as switch statement.
  – A break causes the innermost enclosing loop or switch to be exited immediately.

Example:

```c
int i;
for(i=0; i<5; i++) {
   if(i == 3) {
      break;
   }
   printf("%d", i);
}
```

Output:

```
0 1 2
```
• **Continue Statements**
  
  – The `continue` statement causes the next iteration of the enclosing `for`, `while`, and `do-while` loop to begin.
  
  – A continue statement should only appear in a loop body.

**Example:**

```c
int i;
for(i=0; i<5; i++) {
    if(i == 3) {
        continue;
    }
    printf("%d", i);
}
```

**Output:**

```
0 1 2 4
```
Calculating the Square Root of Number Using Newton’s Method

Based on Newton’s method for finding a root of an equation, the square root \( x = \sqrt{a} \) can be calculated by the formula

\[
x_{i+1} = \frac{1}{2} \left( x_i + \frac{a}{x_i} \right)
\]

where \( x_{i+1} \) is a function of the previous term \( x_i \). It first starts with an initial guess \( x_0 \) for a root, then calculates successive approximate roots \( x_1, x_2, \ldots, x_i, x_{i+1}, \ldots \)

A convergence criterion for numerical computation is a condition to terminate an iteration. Calculate the square root \( \sqrt{3} \) with the initial guess \( x_0 = a \) and the convergence criterion \( |x_{i+1} - x_i| < \text{FLT_EPSILON} \). The program also stops if the number of iteration is larger than 100.
Calculating square root of 3.

```c
/* File: sqrtx.c
   Calculate square root sqrt(a) for a = 3.0 using Newton's method */
#include <stdio.h>
#include <math.h>  /* for fabs() */
#include <float.h> /* for FLT_EPSILON */

#define N 100      /* the maximum number of iteration */

int main() {
    int i;
    double a, x0, x1, x2;

    a = 3.0;        /* sqrt(a) with a = 0.3 */
    x0 = a;         /* an initial guess for x0 */

    x1 = x0;        /* set x1 to x0 */

    for(i = 1; i <= N; i++) {
        x2 = (x1+a/x1)/2.0;     /* Newton's recursive formula */
        if(fabs(x2 - x1) < FLT_EPSILON)
            break;
        x1 = x2;     /* update value x1 for next iteration */
    }
    if(i < N) {     /* number of iteration is less than N */
        printf("sqrtx(%.2f) = %f\n", a, x2);
        printf("sqrt(%.2f) = %f\n", a, sqrt(a));
        printf("Number of iterations = %d\n", i);
    }
    else {          /* number of iteration equals N */
        printf("sqrtx failed to converge\n");
    }
    return 0;
}
```

Fill this part in
Output:

\[ \text{sqrtx}(3.00) = 1.732051 \]
\[ \text{sqrt}(3.00) = 1.732051 \]
Number of iterations = 5

The output indicates that only 5 iterations are needed to converge
Random Number Generation

• Header file `stdlib.h` contains functions and definitions useful for generating random numbers.

• Function `rand()` with prototype
  ```c
  int rand(void);
  ```
generates a sequence of *pseudo-random* integers in the range 0 to `RAND_MAX`.

• The function prototype for function `srand()` is as follows
  ```c
  void srand(unsigned int seed);
  ```
  Argument `seed` is an unsigned integer representing the seed to produce a new sequence of random numbers to be returned by subsequent calls to the `rand()` function.
• For example
  
  ```c
  srand(time(NULL));
  ```

  where function `time()` is defined in header file `time.h`. The `time()` function returns the current calendar time, which is converted to an `unsigned int` to seed the random number generator.

• In order to scale the random numbers directly generated by function `rand()`, the following formula may be used.

  ```c
  n = a + rand() % b
  ```

  to generate an integer in the range of `[a, a+b-1]`. Here, `a` is considered the shifting value, which is equal to the first number of the desired range. Variable `b` is the scaling factor and is equal to the width of the desired range of numbers.

**Example:**

Write a program to simulate the rolling of a six-sided die 6000 times.
/* File: srand.c */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main() {
    int face, rollcount;
    int freq1=0, freq2=0, freq3=0,
        freq4=0, freq5=0, freq6=0;

    srand(time(NULL));
    for (rollcount = 1; rollcount <= 6000; rollcount++) {
        face = 1 + rand() % 6;
        switch(face) {
        case 1:
            freq1 += 1;
            break;
        case 2:
            freq2 += 1;
            break;
        case 3:
            freq3 += 1;
            break;
        case 4:
            freq4 += 1;
            break;
        case 5:
            freq5 += 1;
            break;
        case 6:
            freq6 += 1;
            break;
        }
    }
    printf("Face   Frequency\n");
    printf("  1       %d\n", freq1);
    printf("  2       %d\n", freq2);
    printf("  3       %d\n", freq3);
    printf("  4       %d\n", freq4);
    printf("  5       %d\n", freq5);
    printf("  6       %d\n", freq6);
    return 0;
}
Some Misc. Info

• Auto indentation
• Goto – very bad
Indent a Program Using ChIDE

Click the command “Tools | Indent” to indent the program in the editing pane.
• **The goto Statements**

  A **goto** statement causes an unconditional jump to the statement prefixed by the named label in the enclosing function. A goto statement can either transfer control forward or backward within a function.

**Example:**

```c
for(/* ... */) {
    for(/* ... */) {
        /* ... */
        if(emergency)
            goto hospital;
        goto hospital;
    }
}
/* ... */
hospital:
    emergency_action();
```
Labeled Statements

– The syntax for labeled statements is as follows:

```c
labeled-statement:
    identifier : statement
    case constant-integral expr : statement
    case string-expr : statement
    default : statement
```

– A `case` or `default` label should only appear in a `switch` statement.
– Label names should be unique within a function.
– Any statement may be preceded by a prefix that declares an identifier as a label name.
Pseudocode

Pseudocode is an outline of a program written in a form that can be easily converted into real programming statements. It is used for algorithm development. There is no standard for pseudocode. We will the following structures:

while(expr) . . . endwhile

for i = 1, …, n . . . endfor

if(expr) . . . endif

else

endif

if(expr)

else if(expr2)

else

endif

endif

else

endif
Problem Statement:

Calculate $e^x$ with $x = 0.5$ using the Taylor series until the last term is less than FLT_EPSILON defined in float.h.

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \ldots$$

$$= \sum_{i=0}^{\infty} \frac{x^i}{(i)!}$$

$$e^x \approx \sum_{i=0}^{n} \frac{x^i}{(i)!}$$
Pseudocode for calculating $e^{0.5}$

```c
// File: expx.p
// Pseudocode for straightforward computation of exp(x)
// expx contains exp(x) and term is pow(x,i)/i!

declarax, expr, term
x = 0.5
expx = 1.0
term = 1.0
i = 1
while(term>FLT_EPSILON)  // continue if pow(x,i)/i! > epsilon
    // calculate factorial i!
    factorial = 1
    for j = 1, ..., i
        factorial = factorial*j
    endfor

    term = pow(x, i)/factorial
    expx = expx + term
    i = i + 1
endwhile
print expx and exp(x)
```
Modified pseudocode for calculating $e^{0.5}$ based on the recursive formula

$$\text{term}_i = \text{term}_{i-1} \frac{x}{i}$$

```c
// File: expx2.p
// Pseudocode for recursive computation of exp(x)
// expx contains exp(x) and term is pow(x,i)/i!

declare x, expr, term
x = 0.5
expx = 1.0
term = 1.0
i = 1
while(term>FLT_EPSILON)  // continue if pow(x,i)/i! > epsilon
    term = term*x/i
    expx = expx + term
    i = i+1
endwhile
print expx and exp(x)
```
Flowchart for the algorithm outlined in pseudocode exp2.p
Procedures for Algorithm Development

1. Initialize $x = 0.5$, $\exp x = 1.0$, $\text{term} = 1.0$, and $i = 1$.
2. If $\text{term} > \text{FLT EPSILON}$, continue at step 3.
   Otherwise, print $\exp x$ and $\exp(x)$, then stop.
3. Update $\text{term} = \text{term} * \frac{x}{i}$, $\exp x = \exp x + \text{term}$, and $i = i+1$.
4. Repeat step 2.
Program exp2.c based on pseudocode exp2.p

/* File: expx2.c
   Calculate the approximate value of exp(0.5) recursively */
#include <stdio.h>
#include <math.h>  /* for exp() */
#include <float.h> /* for FLT_EPSILON */

int main() {
    int i;
    double x, expx, term;

    x = 0.5;
    expx = 1.0;
    term = 1.0;
    i = 1;
    while(term>FLT_EPSILON) { /* continue if pow(x,i)/i! > epsilon */
        term *= x/(double)i;
        expx += term;
        i++;
    }
    printf("expx = %f\n", expx);
    printf("exp(x) = %f\n", exp(x));
    return 0;
}
Output:

\[ \exp x = 1.648721 \]
\[ \exp(x) = 1.648721 \]