Lecture Notes for Lecture 4 of CS 5001 (Foundations of CS) for the Fall, 2018 session at the Northeastern University Silicon Valley Campus.

Decision Making, Conditional Processing, and Repetition

Philip Gust,
Clinical Instructor
Department of Computer Science

Lecture 3 Review

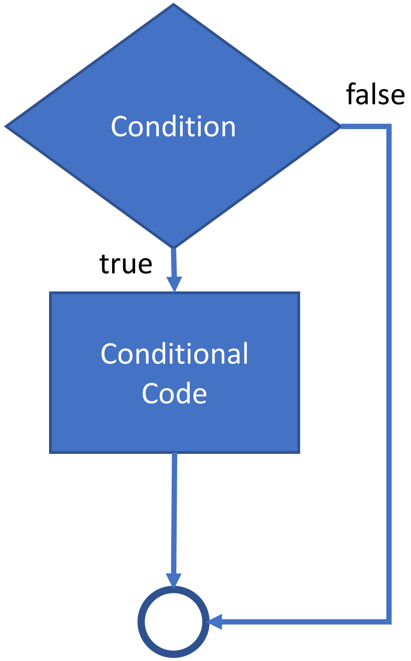
- A C function consists of
    - a documentation block with purpose, parameters, and return type descriptions
    - a prototype including return type and formal parameter declarations
    - a function body that implement the business logic of the function
- Formal parameters are local variables set when the function is called.
- Function then performs its operations and returns a return type value.
- A function with *void* return type returns no value, and with *void* parameter list accepts no arguments.
- Functions can access parameters and other locally declared variables, and global variables that are shared among functions.
- Functional decomposition divides a program among functions that encapsulate business logic and are easier to design and maintain.
- C runtime initializes the program, calls main function, and terminates the program when main function returns.

Decision Making

- Program logic often depends on evaluating conditions during a computation and performing different sequences of operations depending on the results of the evaluation.
- C language provides control statements that use boolean values to determine sequence of statements to perform.
- Boolean values are often the result evaluating expressions involving comparison and logical operators.
- Types of control statements:
  - Conditional
  - Choice
  - Repeated

Conditional Processing

- The simplest conditional control statement uses a boolean value to determine whether to execute statements if the value is *true*. In C, this is know as an “if” statement.



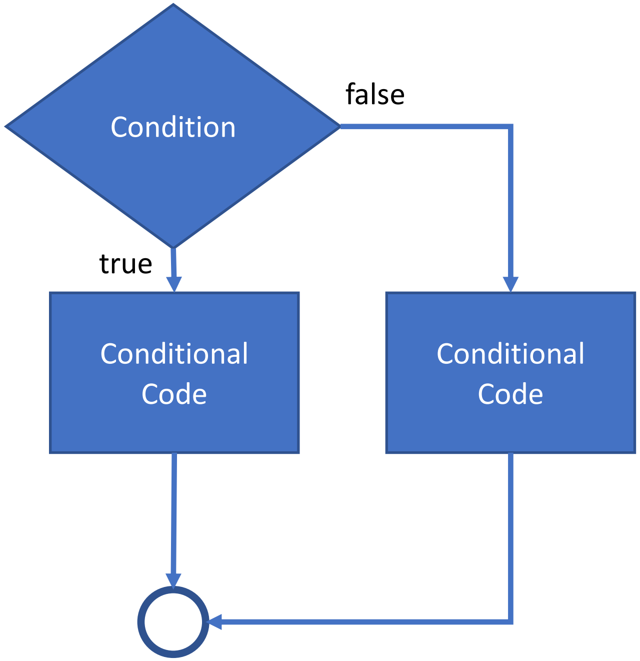
```
if (boolean-condition) {
    conditional statements
}
```

Example: add surtax to water bill if usage exceeds limit.

```
if (waterUsage > usageLimit) {
    waterBill += surtax*waterBill;
}
```

Conditional Processing

- The conditional control statement can include optional statements to execute if the boolean value is false. In C, this is know as an “if-then-else” statement.



```
If (*boolean-condition*) {
   *conditional statements*
} else {
   conditional statements
}
```

Example: add surtax to water bill if usage exceeds usage limit, else apply discount for conservation.

```
If (waterUsage > usageLimit) {
   waterBill += surtax*waterBill;
} else {
   waterBill -= discount*waterBill;
}
```

Conditional Processing

- Example: Compute water bill with adjustments

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

/**  15% surtax for excess use */
const float surtax = 0.15;

/** 10% discount for conservation */
const float discount = 0.10;

/** Maximum usage triggers surtax */
const float usageLimit = 125.0;
```

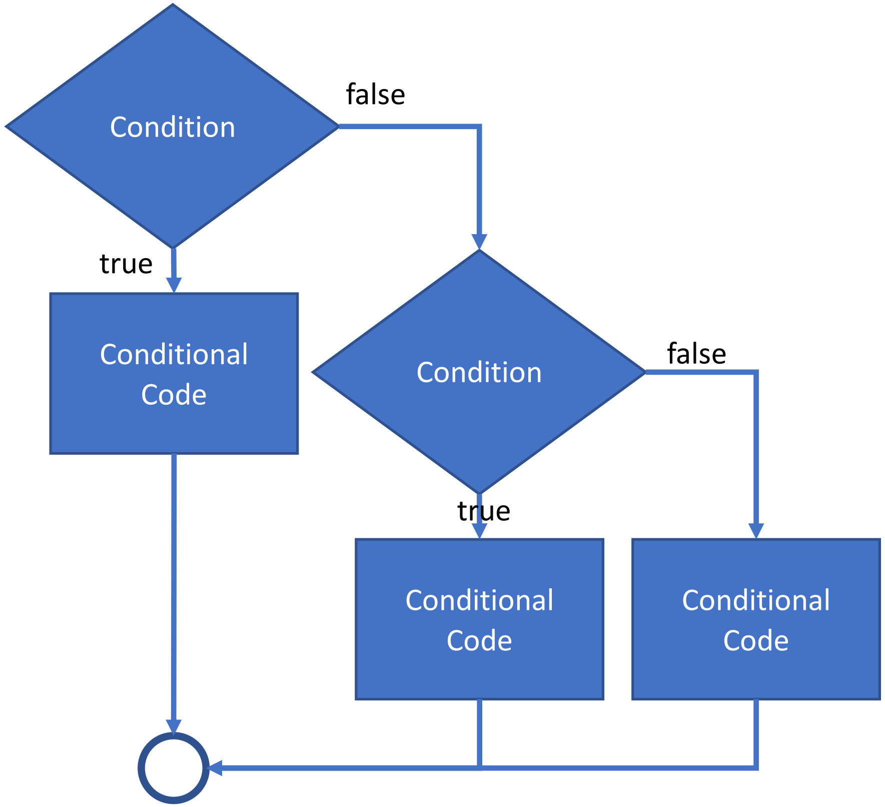
Conditional Processing

- Example: Compute water bill with adjustments

```
/**
 * Adjust water bill based on water usage.
 * @param waterBill original water bill
 * @param usage water usage
 * @ return adjusted water bill based on usage limit
 */
float adjustWaterBill(float waterBill, float usage) {
    if (usage > usageLimit) {  // apply surtax if exceeded limit
        waterBill += surtax * waterBill;
    } else {  // apply discount for conservation
        waterBill -= discount * waterbill;
    }
    return waterBill;
}
```

Conditional Processing

- Control statements can contain other control statements. One style is the cascaded control statement. In C this is known as an “if-then-else-if” statement



```
If (boolean-condition) {
   conditional statements
} else if (boolean-condition) {
   conditional statements
} else {  // optional
   conditional statements
}
```

Example: add surtax to water bill if usage exceeds usage limit, else apply discount for usage below conservation limit

```
If (waterUsage > usageLimit) {
   waterBill += surtax*waterBill;
} else if (waterUsage < conservationLimit) {
   waterBill -= discount*waterBill;
} else {
   // no change to bill
}
```

Conditional Processing

- Example: Compute water bill with adjustments

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

/**  15% surtax for excess use */
const float surtax = 0.15;

/** 10% discount for conservation */
const float discount = 0.10;

/** Maximum usage triggers surtax */
const float usageLimit = 125.0;

/** Maximum usage for discount */
const float conservationLimit = 85.0;
```

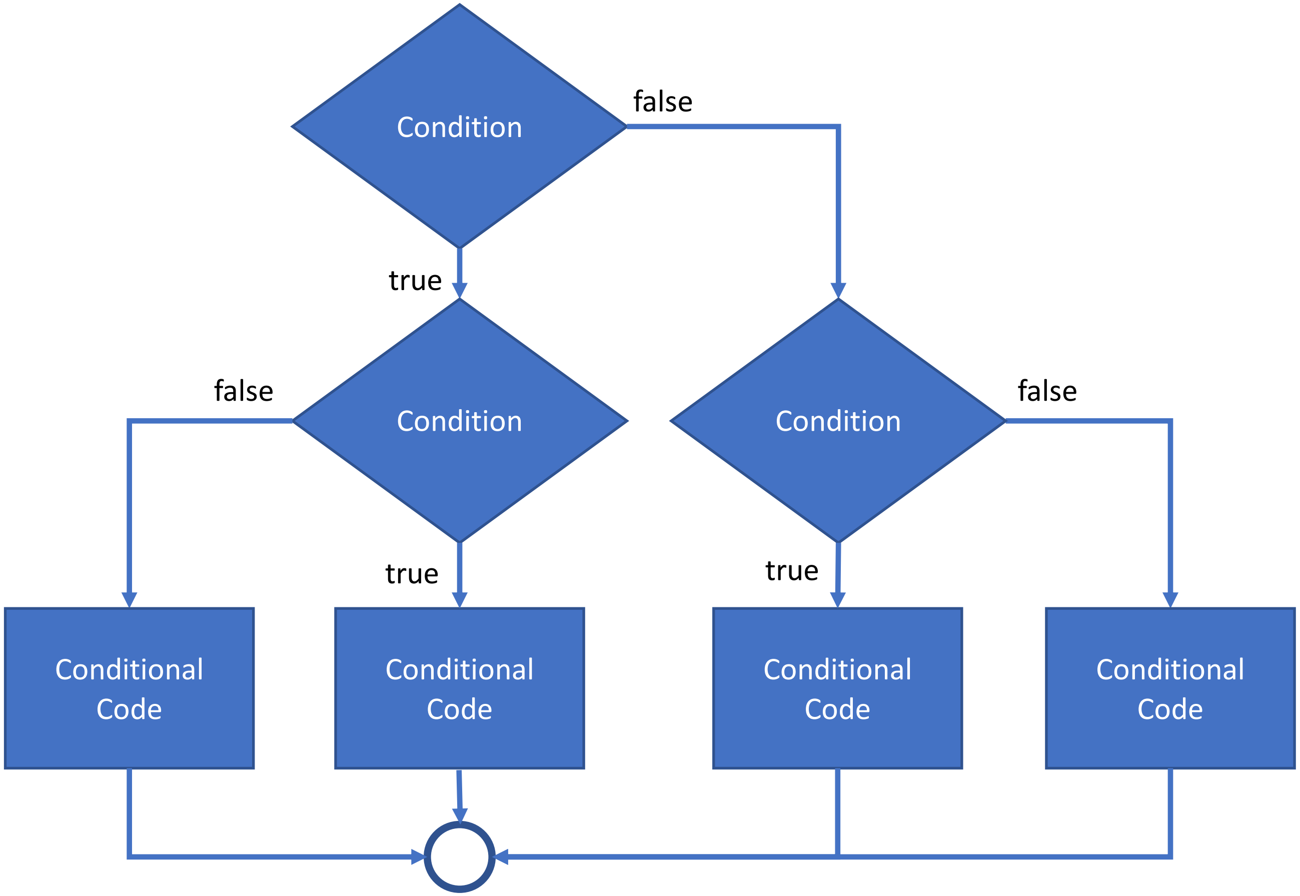
Conditional Processing

- Example: Compute water bill with adjustments

```
/**
 * Adjust water bill based on water usage.
 * @param waterBill original water bill
 * @param usage water usage
 * @ return adjusted water bill based on usage limit
 */
float adjustWaterBill(float waterBill, float usage) {
     if (usage > usageLimit) {  // apply surtax if exceeded limit
             waterBill += surtax * waterBill;
     } else if (usage < conservationLimit) {  // apply discount for conservation
             waterBill -= discount * waterbill;
     }
     return waterBill;
}
```

Conditional Processing

- Another nested control statement is when a second decision is needed on both sides. This is known as a “decision tree.”



Conditional Processing

- Example: If water bill is delinquent, over-use penalties double and conservation discounts are halved.

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

/**  15% surtax for excess use */
const float surtax = 0.15;

/** 10% discount for conservation */
const float discount = 0.10;

/** Maximum usage triggers surtax */
const float usageLimit = 125.0;

/** Maximum usage for discount */
const float conservationLimits= 85.0;
```

Conditional Processing

- Example: If water bill is delinquent, over-use penalties double and conservation discounts are halved.

```
/**
 * Adjust water bill based on water usage.
 * @param waterBill original water bill
 * @param usage water usage
 * @param delinquent true if bill is delinquent
 * @ return adjusted water bill based on usage limit
 */
float adjustWaterBill(float waterBill, float usage, bool delinquent) {
    if (usage > usageLimit) {  // apply surtax if exceeded limit
        if ( delinquent) {
            waterBill += 2 * surtax * waterBill;
        } else {
            waterBill += surtax * waterBill;
        }
    }
```

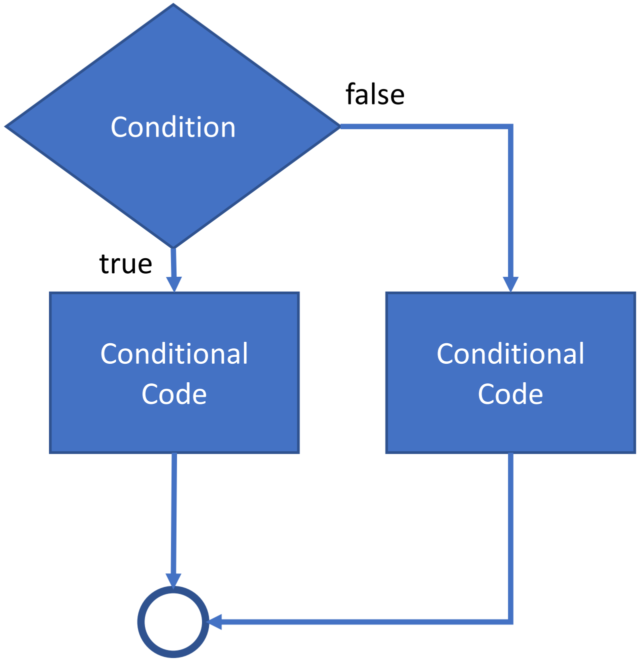
Conditional Processing

- Example: If water bill is delinquent, over-use penalties double and conservation discounts are halved.

```
    else if (usage < conservationLimit) {  // apply discount for conservation
        if (delinquent) {
            waterBill -= discount/2 * waterBill;
        } else {
            waterBill -= discount * waterBill;
        }
    } else {
        if (delinquent) {   // apply surtax if only late
            waterBill += 2 * surtax * waterBill;
        }
    }
    return waterBill;
}
```

Conditional Processing

- Conditional expression has the same role as conditional statement, but can be used in an expression. C language supports conditional expressions



(*boolean-condition*) ? expr : expr

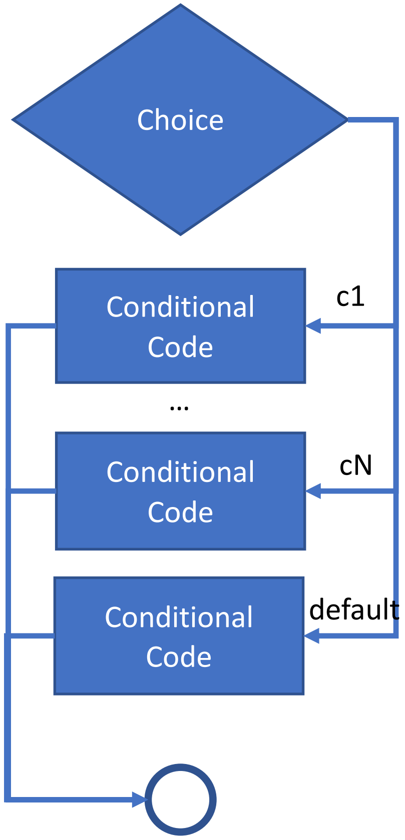
Both expr must be of conforming types.

Example: implement min(x,y) function:.

(x < y) ? x : y

Conditional Processing

- The choice control statement uses an integral value to select a sequences of statements. In C, this is known as a “switch” statement, and switch selectors must be integral constants.



```
switch (integral-value) {
case c1:
   conditional statements
   break;
case c2:
   conditional statements
   break;
…
case cN:
   conditional statements
   break;
default:
   conditional statements
   break;
}
```

Conditional Processing

- Example: Perform specified arithmetic operation

```
/**
 * Perform arithmetic operation: +, -, *, /.
 * @param v1  the first operand
 * @param v2 the second operand
 * @param op the operator
 * @ return the arithmetic result or NAN if unknown operator
 */
float math(float v1, float v2, char op) {
    float result;
    switch (op) {
    case ‘+’:    // addition
          result = v1 + v2;
          break;
```

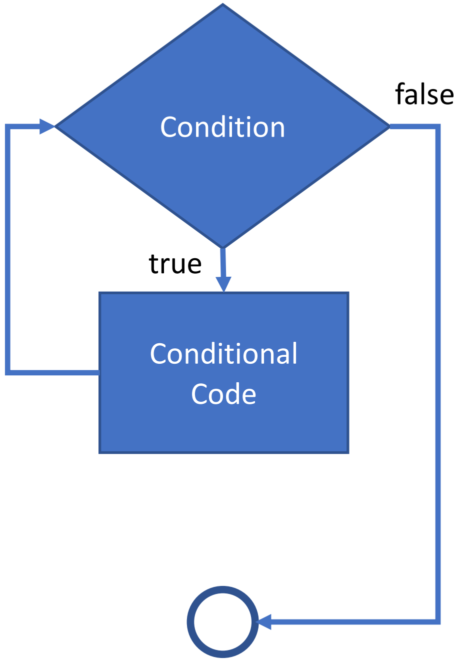
Conditional Processing

- Example: Perform specified arithmetic operation

```
        case ‘-’:    // subtraction
                result = v1 - v2;
                break;
        case ‘*’:    // multiplication
                result = v1 * v2;
                break;
        case ‘/’:    // division
                result = v1 / v2;
                break;
        default:    // unknown operator
                result = NAN;
        }
        return result;
    }
```

Repetition

- The repeated control statement repeats the statement body while a boolean condition remains true. In C repeated control statements include for, while, and do-while.



Repeated control structures

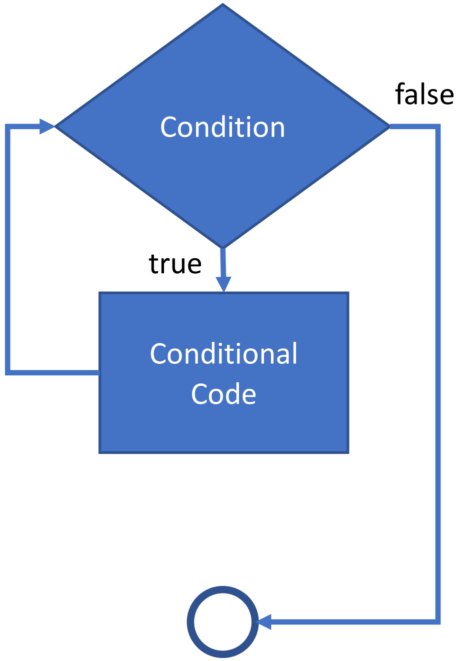
```
while(boolean-condition) {
    conditional statements
}

for (init; boolean-condition; re-init) {
    conditional statements
}

do {
    conditional statements
} while(boolean-condition);
```

Repetition

- Anatomy of while loop



```
while (boolean-condition) {
        conditional-statements;
}
```

- initialization of loop variables and condition done before loop is executed
- *boolean-condition* is evaluated at the beginning of each iteration; conditional code run if condition is true
- *conditional code* is responsible for terminating loop based by changing state of boolean-condition

Repetition

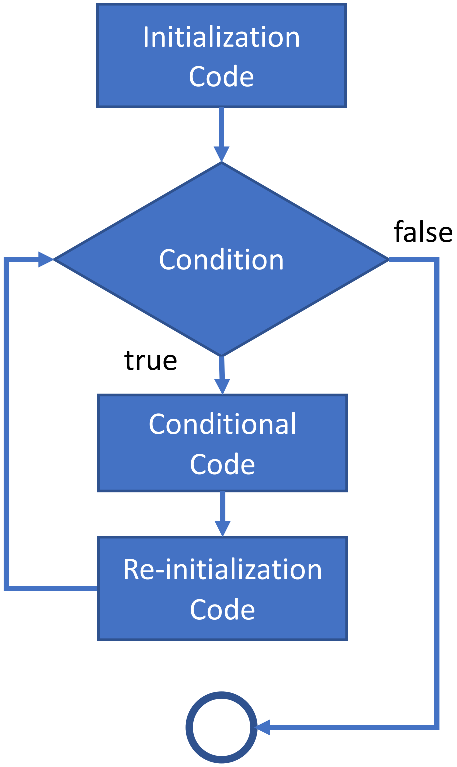
- Example: Echo characters from input to output

```
#include <stdio.h>

/**
 * Echo characters from input to output
 */
int main(void) {
    int ch = getchar();
    while (ch != EOF) {
        putchar(ch);
        ch = getchar();
    }
}
```

Repetition

- Anatomy of for loop



for (*init*; *boolean-condition*; *re-init*) {
        conditional-statements;
}

- *initialization* code executed only once; declare and initialize loop control variables
- *boolean-condition* is evaluated at the beginning of each iteration; conditional code run if condition is true
- *re-initialization* code is performed after conditional code runs, but before *boolean-condition* is re-evaluated.

Repetition

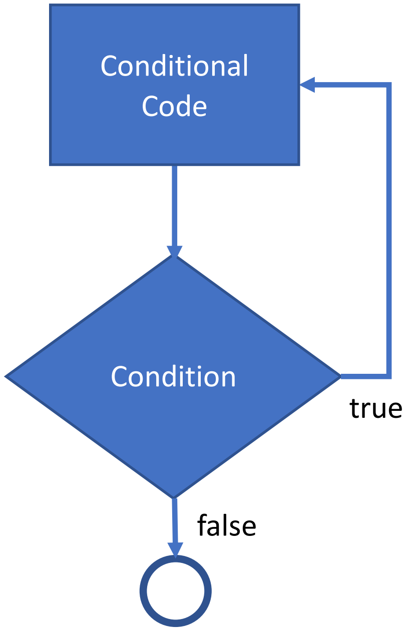
- Example: Printing lower-case letters and their alphabets

```
#include <stdio.h>

/**
 * Print lower-case characters of the alphabet
 */
int main(void) {
    // loop over lower case letters
    for (char ch = 'a'; ch <= 'z'; ch++) {
            printf("'%c':  code: %x\n", ch, ch);
    }
}
```

Repetition

- Anatomy of do-while loop



do {
        conditional-statements;
} while (*boolean-condition*);

- initialization of loop variables and condition done before loop is executed
- conditional code run once before *boolean-condition* evaluated at the end of each iteration; conditional repeats if condition is true
- *conditional code* is responsible for terminating loop based by changing state of boolean-condition

Repetition

- Example: Square root by successive approximation

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

/**
 * Uses Babylonian or Hero’s method to approximate the non-negative
 * real square root of a non-negative real number.
 *
 * The idea is that if x is an overestimate to the square root of a non-negative
 * real number s then (s/x) is an underestimate, so the average of these two
 * numbers may reasonably be expected to provide a better approximation.
 * @param n the number to operate on
 * @param epsilon the precision required
 * @return the square root to the specified precision
 */
```

Repetition

- Example: Square root by successive approximation

```
double mySqrt(double n, double epsilon) {
    if (n < 0.0) {      // special case square root of negative number
        return NAN;
    }
    if (n == 0.0) {    // special case 0 to avoid divide by 0
        return 0.0;
    }
    double guess = n;     // guess value of n as sqrt of n initially
    double lastGuess;
    do {
        lastGuess = guess;    // save last guess and make new one
        guess = (lastGuess + n/lastGuess)/2.0;    // new guess bounds from above
    } while (lastGuess - guess > epsilon);       // done if guesses converged
    return guess;
}
```

Repetition

- Example: Square root by successive approximation

```
/**
 * This function tests square root function
 */
int main(void) {
    double n, sqrtN, epsilon = 1.0e-6;
    n = 0.0;     sqrtN = mySqrt(n, epsilon);     printf("sqrt(%lf): %lf\n", n, sqrtN);
    n = 0.5;     sqrtN = mySqrt(n, epsilon);     printf("sqrt(%lf): %lf\n", n, sqrtN);
    n = 1.0;     sqrtN = mySqrt(n, epsilon);     printf("sqrt(%lf): %lf\n", n, sqrtN);
    n = 2.0;     sqrtN = mySqrt(n, epsilon);     printf("sqrt(%lf): %lf\n", n, sqrtN);
    n = 4.0;     sqrtN = mySqrt(n, epsilon);     printf("sqrt(%lf): %lf\n", n, sqrtN);
    n = 125348.0;   sqrtN = mySqrt(n, epsilon);     printf("sqrt(%lf): %lf\n", n, sqrtN);
}
```