# ICS103 Programming in C

# Lecture 7: Repetition Structures

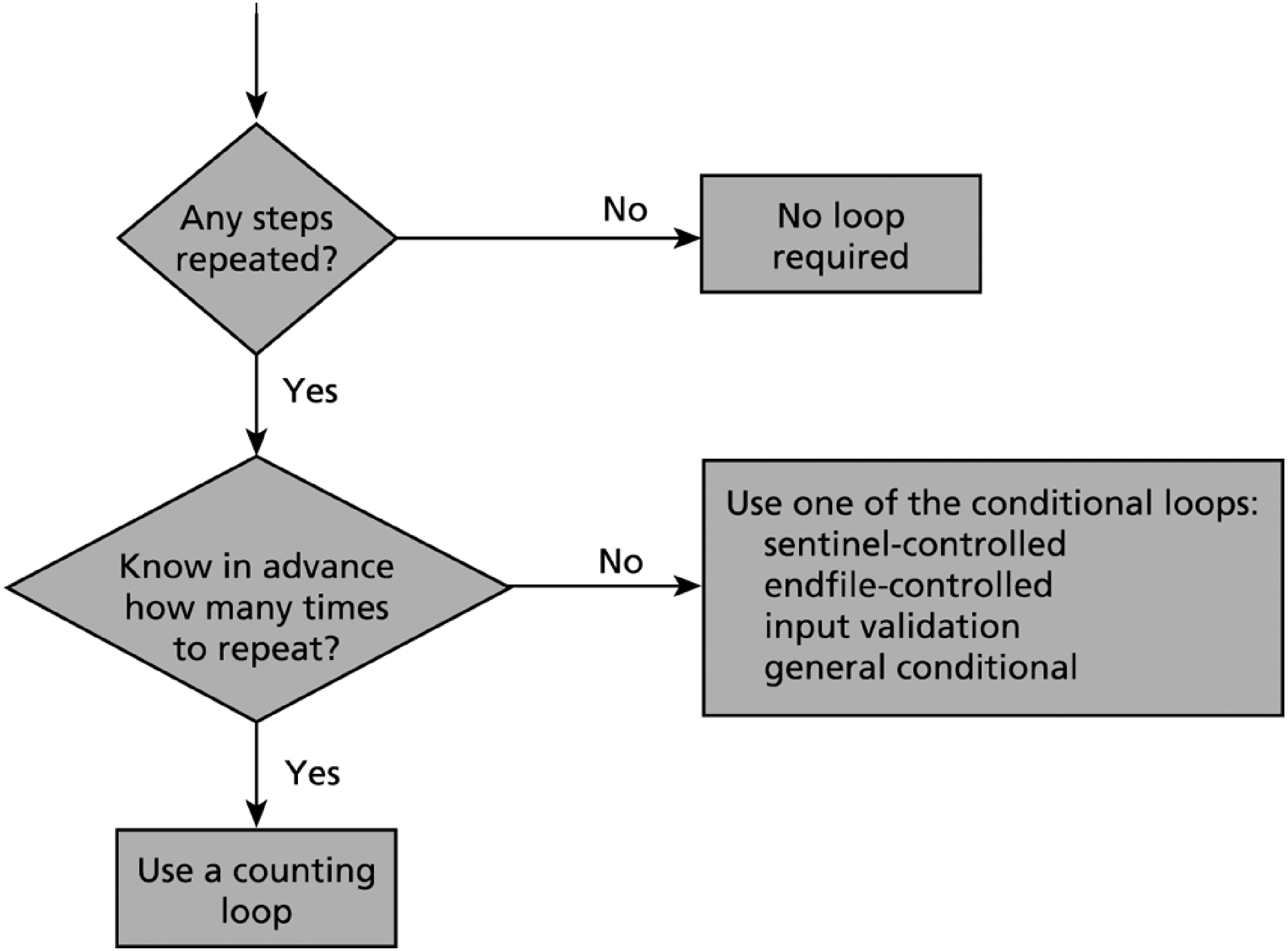
# Overview

- Repetition in Programs
- Counting Loops
  - Using while statement
    - Compound assignment operators
  - Using for statement
    - Increment and Decrement Operators
- Conditional Loops
  - sentinel-Controlled loops
  - Nested loop
  - Do-While loop
  - Flag-Controlled loop
- Hand Tracing the code
- Comparing Double numbers
- Debugging your code

# Repetition in Programs

- We have learned how to write code that chooses between multiple alternatives.
- It is also useful to be able to write code that repeats an action.
- Writing out a solution to a specific case of problem can be helpful in preparing you to define an algorithm to solve the same problem in general.
- After you solve the specific case, you need to determine whether loops will be required in the general algorithm and if so which loop structure to choose from.

# Flow Diagram of Loop Choice Process



# Counting Loops

- The loop shown below in pseudo code is called a counter-controlled loop (or counting loop) because its repetition is managed by a loop control variable whose value represents a count.

```
Set loop control variable to an initial value of 0
While loop control variable < final value
   ... //Do something multiple times
   Increase loop control variable by 1.
```

- We use a counter-controlled loop when we can determine prior to loop execution exactly how many loop repetitions will be needed to solve the problem.

# The While Statement

- This slide shows a program fragment that computes and displays the gross pay for seven employees. The loop body is the compound statements (those between { and }).
- The **loop repetition condition** controls the while loop.

```
count_emp = 0;
while (count_emp < 7)
{
  printf("Hours> ");
  scanf("%d",&hours);
  printf("Rate> ");
  scanf("%lf",&rate);
  pay = hours * rate;
  printf("Pay is $%6.2f\n", pay);
  count_emp = count_emp + 1;
}
printf("\nAll employees processed\n");
```

**loop repetition condition**

# While Statement

- General form:

```
While (loop repetition condition)
{
   //Steps to perform.  These should eventually
   //result in condition being false
}
```

- Syntax of the while Statement:
  - Initialization. i.e. `count_emp = 0;`
  - Testing. i.e. `count_emp < 7`
  - Updating i.e. `count_emp = count_emp + 1;`
- The above steps must be followed for every while loop.
- If any of these are skipped it may produce an **infinite loop**

# General While Loops

- In the above example we had `count_emp < 7`, but we may have more or less than 7 employees.
- To make our program fragment more general we should use a `printf/scanf` to get the number of employees and store it is `num_emp`.
- Now we can have `count_emp < num_emp` and our code is more general.

# Computing Sum

- If we want to compute $\sum_{i=1}^{100} i$, we need to go 1+2+3+...+100
- We can use a while loop.

```
/* computes the sum: 1 + 2 + 3 + ....+ 100 */
#include <stdio.h>

int main(void) {
    int sum =0, i = 1;

    while (i <= 100) {
        sum = sum + i;
        i = i + 1;
    }
    printf("Sum is %d\n", sum);
    return 0;
}
```

# Compound Assignment Operators

- Several times we have seen:

  `variable = variable <operator> expression;`

  Example: `sum = sum + i;`
- where `<operator>` is a C operator
- This occurs so often, C gives us short cuts.
- Instead of writing `x = x +1` we can write:

  `x += 1.`
- W can use -=, *=, /=, and %= in the same way.

# The For Statement

- A better way to construct a counting loop is to use the **for** statement.
- C provides the **for** statement as another form for implementing loops.
- As before we need to
  - **Initialize** the loop control variable
  - **Test** the loop repetition condition
  - **Update** the loop control variable.
- An important feature of the for statement in C is that it supplies a designated place for each of these three components.
- An example of the for statement is shown in the next slide.

# For Example

- To compute the sum of 1 to 100:

```
int sum = 0;
int i;
for (i = 1; i <= 100; i++)
{
   sum = sum + i;
}
```

- Note: `i++` is the same as `i = i + 1` and as `i += 1`.

# General Form of For statement

```
for (initialize; test; update)
{
  //Steps to perform each iteration
}
```

- First, the initialization expression is executed.
- Then, the loop repetition condition is tested.
- If the condition is true, the statement enclosed in { } are executed.
- After that the update expression is evaluated.
- Then the loop repetition condition is retested.
- The statement is repeated as long as the condition is true.
- For loop can be used to count up or down by any interval.

# Program Style

- For clarity, it can be useful to place each expression of the *for* heading on a separate line.
- If all three expressions are very short, we will place them together on one line, like we did in the example.
- The body of the for loop is indented just as the if statement.

# Increment and Decrement Operators

- The counting loops that we have seen have all included assignment expressions of the form

  ```
  counter = counter + 1
  ```

  or

  ```
  counter++
  ```

  or

  ```
  counter += 1
  ```

- This will add 1 to the variable counter. If we use a - instead of a +, it will subtract 1 from the variable counter.
- Be careful about using the ++ or -- options.

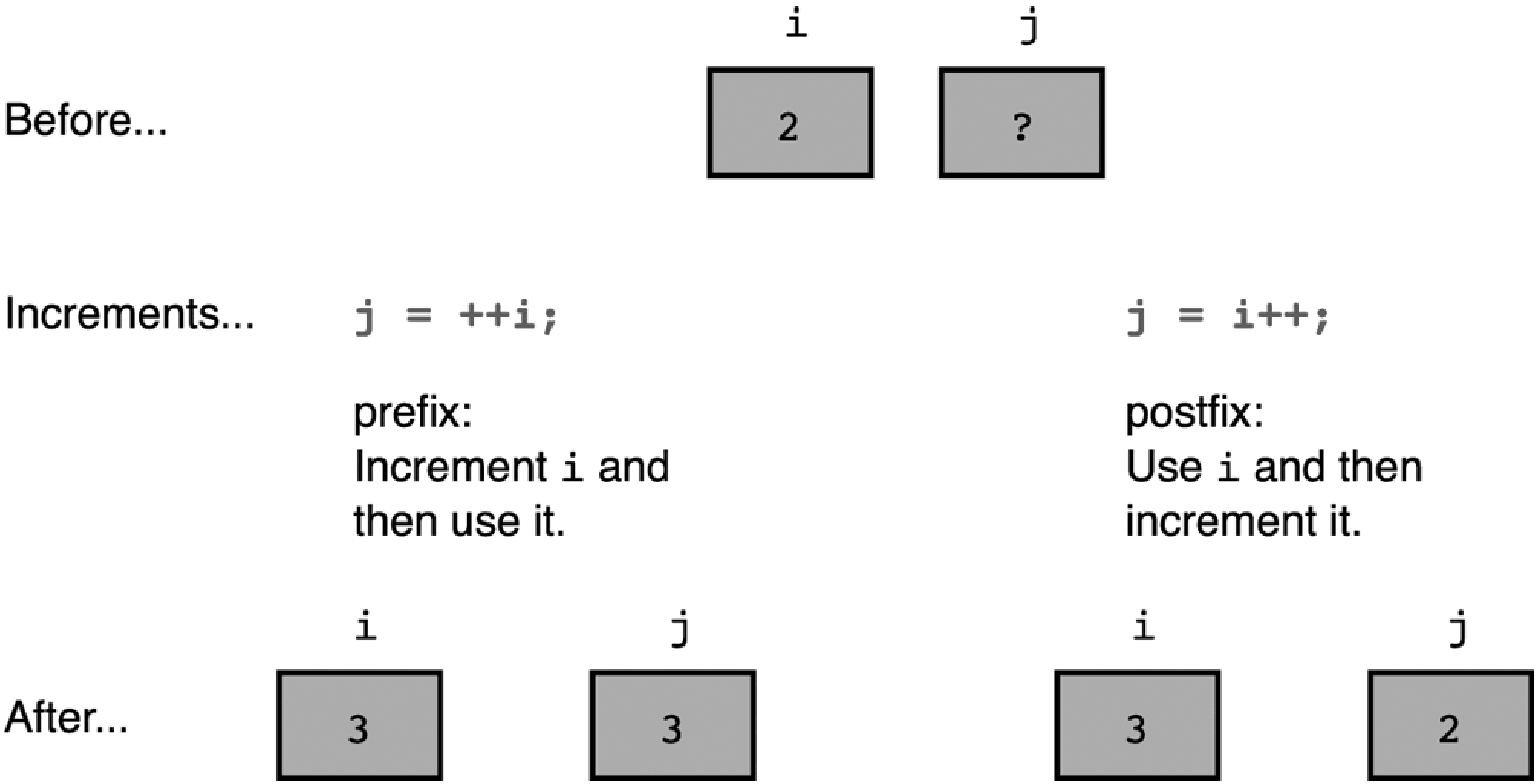
# Increment and Decrement Other Than 1

- Instead of adding just 1, we can use `sum = sum + x` or `sum += x`
- Both of these will take the value of `sum` and add `x` to it and then assign the new value to `sum`.
- We can also use `temp = temp -x` or `temp -= x`
- Both of these will take the value of `temp` and subtract `x` from it and then assign the new value to `temp`.

# Prefix and Postfix Increment/Decrement

- The values of the expression in which the ++ operator is used depends on the position of the operator.
- When the ++ operator is placed immediately in front of its operand (prefix increment, Ex: `++x`), the value of the expression is the variable’s value after incrementing.
- When the ++ operator is placed immediately after the operand (postfix increment , Ex: `x++`), the value of the expression is the value of the variable before it is incremented.

# Comparison of Prefix and Postfix Increments



```
j = ++i;
```

prefix:
Increment i and
then use it.

```
j = i++;
```

postfix:
Use i and then
increment it.

# More on prefix and postfix operator

- If n = 4, what will be the output of the following?

```
printf(“%3d”, --n);
printf(“%3d”, n);
```

```
  3  3
```

```
printf(“%3d”, n--);
printf(“%3d”, n);
```

```
  4  3
```

# Conditional Loops

- In many programming situations, we will not be able to determine the exact number of loop repetitions before loop execution begins.
- Below is an example where we do not know how many times our program will repeat.

# Example

- We need a program that prompts the user for a `value` and multiplies it by the value of the variable `temp`. It then stores the result in `temp`. It keeps doing this until the user enters a 0.
- The outline of the program would be as follows:

```
assign temp the value of 1
prompt the user for a value
while value does not equal 0
  assign temp the value of temp times value
  prompt the user for a value
output the value of temp
```

# Program Fragment

```
temp = 1;
printf("Enter a value, 0 will stop the program> ");
scanf("%d",&value);                         Initialization
while(value != 0) {  <-------------  Testing
   temp = temp * value;

   printf("Enter a value, 0 will stop the program>");
   scanf("%d",&value);                      Update
}
printf("The product is %d", temp);
```

- It is very common for loops to have identical initialization and update steps while performing input operations where the number of input values is not known in advance.

# Sentinel Controlled Loops

- Many programs with loops input one or more additional data items each time the loop body is repeated.
- Often we don’t know how many data items the loop should process when it begins execution.
- We must find some way to signal the program to stop reading and processing new data.
- One way to do this is to instruct the user to enter a unique data value, called a **sentinel** value, after the last data item.
- The loop repetition condition tests each data item and causes loop exit when the sentinel value is read.
- This is what we did in the previous example: use the value 0 to stop the loop.

# Sentinel-Controlled while Loop

```
/* Compute the sum of a list of exam scores. */

#include <stdio.h>
#define SENTINEL -99

int main(void)  {
        int sum = 0,   /* sum of scores input so far              */
            score;     /* current score                  */
        printf("Enter first score (or %d to quit)> ", SENTINEL);
       scanf("%d", &score );
while   (score != SENTINEL) {
             sum += score;
            printf("Enter next score (%d to quit)> ", SENTINEL);
           scanf("%d", &score);
        }
        printf("\nSum of exam scores is %d\n", sum);
        system("pause");
        return (0);
}
```

# Sentinel Controlled for loop

- Because the for statement combines the initialization, test, and update in once place, some programmers prefer to use it to implement sentinel-controlled loops.

```
printf("Enter first score (or %d to quit)> ", sentinel);
for( scanf("%d",&score);
     score != sentinel;
     scanf("%d",&score))
{
   sum += score;
   printf("Enter next score (%d to quit)> ", sentinel);
}
```

# Nested Loops

- Usually used to work with two dimensional arrays (later).
- Nested loops consist of an outer loop with one or more inner loops.
- Each time the outer loop is repeated, the inner loops are reentered
  - Their loop control expressions are reevaluated
  - All required iterations are performed again.

# Example: Bald eagle sightings for a year

```
/* Tally by month the bald eagle sightings for the year. Each month's
 * sightings are terminated by the sentinel zero. */

#include <stdio.h>
#define SENTINEL   0
#define NUM_MONTHS 12

int
main(void)
{
      int month,     /* number of month being processed          */
      mem_sight, /* one member's sightings for this month    */
      sightings;  /* total sightings so far for this month     */

      printf("BALD EAGLE SIGHTINGS\n");
      for (month = 1;
               month <= NUM_MONTHS;
               ++month) {
               sightings = 0;
               for (scanf("%d", &mem_sight);
                              mem_sight != SENTINEL;
                              scanf("%d", &mem_sight)) {
                              sightings += mem_sight;
               }
               printf("  month %2d: %2d\n", month, sightings);
      }
      return (0);
}
```

```
/*Sample Input data */
2 1 4 3 0
1 2 0
0
5 4 1 0
. . .
```

```
/*sample output */

BALD EAGLE SIGHTINGS
  month  1: 10
  month  2:  3
  month  3:  0
  month  4: 10
  . . .
```

# What is the Output?

```
/*
 * Illustrates a pair of nested counting loops
 */

#include <stdio.h>

Int  main(void)
{
      int i, j;   /* loop control variables */
      printf("          I    J\n");
      for  (i = 1;  i < 4;  ++i)   {
         printf("Outer  %6d\n", i);
         for  (j = 0;  j < i;  ++j)   {
            printf("  Inner%9d\n", j);
         }   /* end of inner loop */
      }   /*  end of outer loop */

      return (0);
}
```

```
//output:
           I   J
Outer      1
  Inner        0
Outer      2
  Inner        0
  Inner        1
Outer      3
  Inner        0
  Inner        1
  Inner        2
```

# Do While statement

- Both the for statement and the while statement evaluate the loop condition before the first execution of the loop body.
- In most cases, this pretest is desirable and prevents the loop from executing when there may be no data items to process
- There are some situations, generally involving interactive input, when we know that a loop must execute at least one time.

# Do-While Example

```
#include <stdio.h>
#define KMS_PER_MILE 1.609

/* converts miles to kilometers - repeateadly */
int main(void) {
        double kms,
             miles;
        char res;  //for user response [y/n]

        do {
          printf("Enter the distance in miles> ");
          scanf("%lf", &miles);
          kms = KMS_PER_MILE * miles;
          printf("That equals %f kilometers. \n", kms);
          printf("\nDo you wish to try again [y/n]? ");
          getchar(); //skips the new line character.
          scanf("%c", &res);
        } while (res == 'Y' || res == 'y');

        system("PAUSE");
        return (0);
}
```

# Do-While Example

```
/* Gets an integer input value in the range from n_min to n_max. */
#include <stdio.h>
int main(void) {
     int  n_min, n_max; /* minimum and maximum values                    */
     int  inval,  /* data value which user enters                 */
         status;  /* status value returned by scanf               */
     char skip_ch;                /* character to skip            */
     int  error;                  /* error flag for bad input   */
     printf("Enter minimum and maximum valid values> ");
     scanf("%d%d", &n_min, &n_max);
     do {
        printf("Enter an integer in the range from %d to %d inclusive> ",
              n_min, n_max);
        status = scanf("%d", &inval);
        if (status == 1) {
             error = 0;
        } else {
             error = 1;
             scanf("%c", &skip_ch);
             printf("\nInvalid character>>%c>> Skipping rest of line.\n",
                   skip_ch);
             do {
                scanf("%c", &skip_ch);
             }  while (skip_ch != '\n') ;
        }
      }  while (error || inval < n_min || inval > n_max) ;
      /* Rest of Processing */
}
```

***scanf* returns how many values it actually read from user**

# Flag Controlled Loops

- Sometimes a loop repetition condition becomes so complex that placing the full expression in its usual spot is awkward.
- In many cases, the condition may be simplified by using a **flag**.

```
while (flag)
{
    ….
}
```

- A **flag** is a type `int` variable used to represent whether or not a certain event has occurred.
- A flag has one of two values: 1 (true) and 0 (false).

# Flag Controlled Example

```
/* Gets a valid fraction */
int main(void) {
    int  num, den;       /* numerator, denominator of fraction                  */
    char slash;          /* character between numerator and  denominator */
    int  status;         /* status code returned by scanf indicating number of valid values obtained   */
    int  error;          /* flag indicating whether or not an error has been detected in current input */
    char discard;        /* unprocessed character from input line               */

    do {
      /* No errors detected yet                          */
      error = 0;

      /* Get a fraction from the user                                     */
      printf("Enter a common fraction as two integers separated by ");
      printf("a slash\nand press <enter>\n> ");
      status = scanf("%d%c%d", &num, &slash, &den);
```

# Flag Controlled Example

```
 /* Validate the fraction */
       if (status < 3) {
           error = 1;
           printf("Input invalid-please read directions carefully\n");
       } else if (slash != '/') {
           error = 1;
           printf("Input invalid-separate numerator and denominator");
           printf(" by a slash (/)\n");
       } else if (den <= 0) {
           error = 1;
           printf("Input invalid-denominator must be positive\n");
       }

       /* Discard extra input characters          */
       do {
          scanf("%c", &discard);
       } while (discard != '\n');
    } while (error);

    /* Finish processing of fraction - code omitted                          */
}
```

# Hand Tracing the Code

- A critical step in program design is to verify that an algorithm or C statement is correct before you spend extensive time coding or debugging it.
- Often a few extra minutes spent in verifying the correctness of an algorithm saves hours of coding and testing time.
- A **hand trace** or **desk check** is a careful, step-by-step simulation on paper of how the computer executes the algorithm or statement.
- The results of this simulation should show the effect of each step’s execution using data that is relatively easy to process by hand.

# Hand Trace

Given this code,
what is the output?

```
int main ( void )
{
   int a=3, b=4, c=5, d=6,total=2;
   if( a<b  &&  c>d )
     total = 3;
   else
     total = 4;
    switch (total) {
      case 2:
         printf(“hello\n”);
         break;
      case 3:
         printf(“good-bye\n”);
         break;
      default:
         printf(“so long\n”);
      }
}
```

# Hand Trace

Given this code,
what is the output?

| a | b | c | d | total | a<b | c>d | && |
|---|---|---|---|---|---|---|---|
|  |  |  |  |  |  |  |  |

**Output:**

```
int main ( void )
{
   int a=3, b=4, c=5, d=6,total=2;
   if( a<b  &&  c>d )
     total = 3;
   else
     total = 4;
    switch (total) {
      case 2:
         printf(“hello\n”);
         break;
      case 3:
         printf(“good-bye\n”);
         break;
      default:
         printf(“so long\n”);
      }
}
```

# Hand Trace

Given this code,
what is the output?

```
int main ( void )
{
  int a=3, b=4, c=5, d=6,total=2;
  if( a<b  &&  c>d )
    total = 3;
  else
    total = 4;
   switch (total) {
     case 2:
        printf(“hello\n”);
        break;
     case 3:
        printf(“good-bye\n”);
        break;
     default:
        printf(“so long\n”);
     }
}
```

| a | b | c | d | total | a<b | c>d | && |
|---|---|---|---|---|---|---|---|
| 3 | 4 | 5 | 6 | 2 | | | |

**Output:**

# Hand Trace

Given this code,
what is the output?

| a | b | c | d | total | a<b | c>d | && |
|---|---|---|---|---|---|---|---|
| 3 | 4 | 5 | 6 | 2 | T | F | |

**Output:**

```
int main ( void )
{
   int a=3, b=4, c=5, d=6,total=2;
   if( a<b  &&  c>d )
     total = 3;
   else
     total = 4;
    switch (total) {
      case 2:
         printf("hello\n");
         break;
      case 3:
         printf("good-bye\n");
         break;
      default:
         printf("so long\n");
      }
}
```

# Hand Trace

Given this code,
what is the output?

| a | b | c | d | total | a<b | c>d | && |
|---|---|---|---|---|---|---|---|
| 3 | 4 | 5 | 6 | ~~2~~ | T | F | F |
|  |  |  |  | 4 |  |  |  |

```
int main ( void )
{
   int a=3, b=4, c=5, d=6,total=2;
   if( a<b  &&  c>d )
     total = 3;
   else
     total = 4;
    switch (total) {
     case 2:
        printf(“hello\n”);
        break;
     case 3:
        printf(“good-bye\n”);
        break;
     default:
        printf(“so long\n”);
     }
}
```

**Output:**

# Hand Trace

Given this code,
what is the output?

| a | b | c | d | total | a<b | c>d | && |
|---|---|---|---|---|---|---|---|
| 3 | 4 | 5 | 6 | ~~2~~ | T | F | F |
|  |  |  |  | 4 |  |  |  |

**Output:**

```
int main ( void )
{
   int a=3, b=4, c=5, d=6,total=2;
   if( a<b  &&  c>d )
     total = 3;
   else
     total = 4;
    switch (total) {
      case 2:
         printf(“hello\n”);
         break;
      case 3:
         printf(“good-bye\n”);
         break;
      default:
         printf(“so long\n”);
      }
}
```

# Hand Trace

Given this code,
what is the output?

```
int main ( void )
{
   int a=3, b=4, c=5, d=6,total=2;
   if( a<b  &&  c>d )
     total = 3;
   else
     total = 4;
    switch (total) {
      case 2:                          False
         printf(“hello\n”);
         break;
      case 3:
         printf(“good-bye\n”);
         break;
      default:
         printf(“so long\n”);
      }
}
```

| a | b | c | d | total | a<b | c>d | && |
|---|---|---|---|---|---|---|---|
| 3 | 4 | 5 | 6 | ~~2~~ | T | F | F |
|  |  |  |  | 4 |  |  |  |

**Output:**

# Hand Trace

Given this code,
what is the output?

| a | b | c | d | total | a<b | c>d | && |
|---|---|---|---|---|---|---|---|
| 3 | 4 | 5 | 6 | ~~2~~ | T | F | F |
| | | | | 4 | | | |

**Output:**

```
int main ( void )
{
   int a=3, b=4, c=5, d=6,total=2;
   if( a<b  &&  c>d )
     total = 3;
   else
     total = 4;
    switch (total) {
      case 2:
         printf("hello\n");
         break;
      case 3:
         printf("good-bye\n");
         break;
      default:
         printf("so long\n");
      }
}
```

False

# Hand Trace

Given this code,
what is the output?

| a | b | c | d | total | a<b | c>d | && |
|---|---|---|---|---|---|---|---|
| 3 | 4 | 5 | 6 | ~~2~~ | T | F | F |
| | | | | 4 | | | |

**Output:**

```
int main ( void )
{
   int a=3, b=4, c=5, d=6,total=2;
   if( a<b  &&  c>d )
     total = 3;
   else
     total = 4;
    switch (total) {
      case 2:
         printf(“hello\n”);
         break;
      case 3:
         printf(“good-bye\n”);
         break;
      default:                         True
         printf(“so long\n”);
      }
}
```

# Hand Trace

Given this code,
what is the output?

| a | b | c | d | total | a<b | c>d | && |
|---|---|---|---|---|---|---|---|
| 3 | 4 | 5 | 6 | ~~2~~ | T | F | F |
|  |  |  |  | 4 |  |  |  |

```
int main ( void )
{
   int a=3, b=4, c=5, d=6,total=2;
   if( a<b  &&  c>d )
     total = 3;
   else
     total = 4;
    switch (total) {
      case 2:
         printf(“hello\n”);
         break;
      case 3:
         printf(“good-bye\n”);
         break;
      default:
         printf(“so long\n”);
      }
}
```

**Output:** **so long**

# Hand Trace

Given this code,
what is the output?

| a | b | c | d | total | a<b | c>d | && |
|---|---|---|---|---|---|---|---|
| 3 | 4 | 5 | 6 | ~~2~~ | T | F | F |
| | | | | 4 | | | |

```
int main ( void )
{
   int a=3, b=4, c=5, d=6,total=2;
   if( a<b  &&  c>d )
     total = 3;
   else
     total = 4;
    switch (total) {
      case 2:
         printf(“hello\n”);
         break;
      case 3:
         printf(“good-bye\n”);
         break;
      default:
         printf(“so long\n”);
      }
}
```

**Output: so long**

# Hand Trace

Given this code,
what is the output?

```
int main ( void )
{
   int a=3, b=4, c=5, d=6,total=2;
   if( a<b  &&  c>d )
     total = 3;
   else
     total = 4;
    switch (total) {
      case 2:
         printf(“hello\n”);
         break;
      case 3:
         printf(“good-bye\n”);
         break;
      default:
         printf(“so long\n”);
      }
}
```

| a | b | c | d | total | a<b | c>d | && |
|---|---|---|---|---|---|---|---|
| 3 | 4 | 5 | 6 | ~~2~~ | T | F | F |
|  |  |  |  | 4 |  |  |  |

**Output:** **so long**

**main exits**