CS 201
Repetition

Debzani Deb

Overview

- Loops
  - Endfile-Controlled loop
  - Nested loop
  - Do-While loop
  - Flag-Controlled loop
- Hand Tracing the code
- Comparing Double numbers
- Debugging your code

Endfile-controlled loops

- Many library functions return helpful data as the value of the function.
- For example, `scanf` returns the number of data items it actually obtained.

```
num = scanf("%d%d%d", &n1, &n2, &n3);
```

- A data file is always terminated by an endfile character (EOF) that can be detected by the `scanf` and `fscanf` functions.
- You can write a program that processes a list of data of any length from a file without requiring a special sentinel value at the end of the data file.


```
#include <stdio.h>
int main(void) {
   FILE *inp;
   int sum = 0, score, input_status;
   inp = fopen("scores.dat", "r");
   input_status = fscanf(inp, "%d", &score);
   while (input_status != EOF)
   {
      printf("%d ", score);
      sum += score;
      input_status = fscanf(inp, "%d",   &score);
   }
   printf("\nSum of exam scores is %d\n", sum);
   fclose(inp);
   return 0;
}
```

```
/* Output */
55 33 77
Sum of exam scores is 165
```


What is the Output?

You can send EOF to `scanf` with ctrl-D (linux)

```
int i, sum=0,status=0;
status = scanf("%d",&i);
while(status!=EOF)
{
  sum+=i;
  status = scanf("%d",&i);
}
printf("%d\n", sum);
```

```
/* Input = */
2
4
6
ctl-d
```

```
/* Output = */
12
```


Nested Loops

- Usually used to work with two dimensional arrays (later).
- Nested loops consist of an outer loop with 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.

What is the Output?

```
lcv1 = 0;
sum = 0;
while (lcv1 < 100)
{
   lcv2 = 0;
   while (lcv2 < 100)
   {
      sum = sum + 1;
      lcv2++;
   }
   lcv1++;
}
printf("Sum is %d\n", sum);
```

```
/* Output = */
Sum is 10000
```


What is the Output?

```
int a=0, b=0, sum=0;
for(a=0; a<6; a+=2)
     for(b=0; b>4; b--)
          sum=sum+1;
printf("%d",sum);
```

```
/* Output = */
Sum is 0
```


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

```
do
{
   printf("Enter a letter from A through E> ");
   scanf("%c", &letter_choice);
}while (letter_choice < 'A' || letter_choice > 'E');
```


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).

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?

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

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	

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~~ 4	T	F	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~~ 4	T	F	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~~ 4	T	F	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:                    False
      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~~ 4	T	F	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:                    False
      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~~ 4	T	F	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:                   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~~ 4	T	F	F

Output: so long

```
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~~ 4	T	F	F

Output: so long

```
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~~ 4	T	F	F

Output: so long

```
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");
    }
}
```

main exits

Comparing `double` type (1)

- NEVER use if (double_variable == 0.0)
- Because of the way variables of type double are stored in memory, it makes it very difficult to compare double types for equality.
- Avoid trying to compare double variables for equality, or you may encounter unpredictable results.

Comparing `double` type (2)

- The reason is that internal representations for double variables can have a bit set that will fail a comparison to zero when the value is basically zero.
- Better to do the following:
  `if ( fabs(double_variable) < some_epsilon )`
- You can read more about this
  www.cygnus-software.com/papers/comparingfloats/comparingfloats.htm

What is the Output?

```
y = sqrt(2.0); // what if I compare with my calculator
if(y == 1.4142135)
  printf("test y==1.4142135 is true\n");
else
  printf("test y==1.4142135 is false\n);
printf("ACTUAL VALUE OF SQRT(2.0) IS : %f\n", y);
if(fabs(y-1.4142135)<0.0001)
  printf("test fabs(y-1.4142135)<0.0001 is true\n");
else
  printf("test fabs(y-1.4142135)<0.0001 is false\n");
```

```
/* Output */
test y==1.4142135 is false
ACTUAL VALUE OF SQRT(2.0) IS : 1.414214
test fabs(y-1.4142135)<0.0001 is true
```


Debugging your code

- Use a source level debugger as part of your IDE.
- Use a command line debugger like `gdb`.
- Add `printf` statements to trace execution.

Using Debugger Programs

- A debugger program can help you debug a C program.
- The debugger program lets you execute your program one statement at a time.
- Through **single-step execution**, you can trace your program's execution and observe the effect of each C statement on variables you select.
- If your program is very long, you can separate your program into segments by setting breakpoints at selected statements.
- The debugger executes all statements upto the breakpoint and then you can examine the variables and values to make sure your program is working.

Debugging with `printf`

- Use several `printf` statements to output the values of your variables to make sure they have the correct value in them as your program executes.
- It can be very handy to print out the value of your loop control variable to make sure you are incrementing it and will not enter an infinite loop.

∞ Loops

- It's hard to distinguish between a complex calculation loop and an infinite loop without debugging statements.