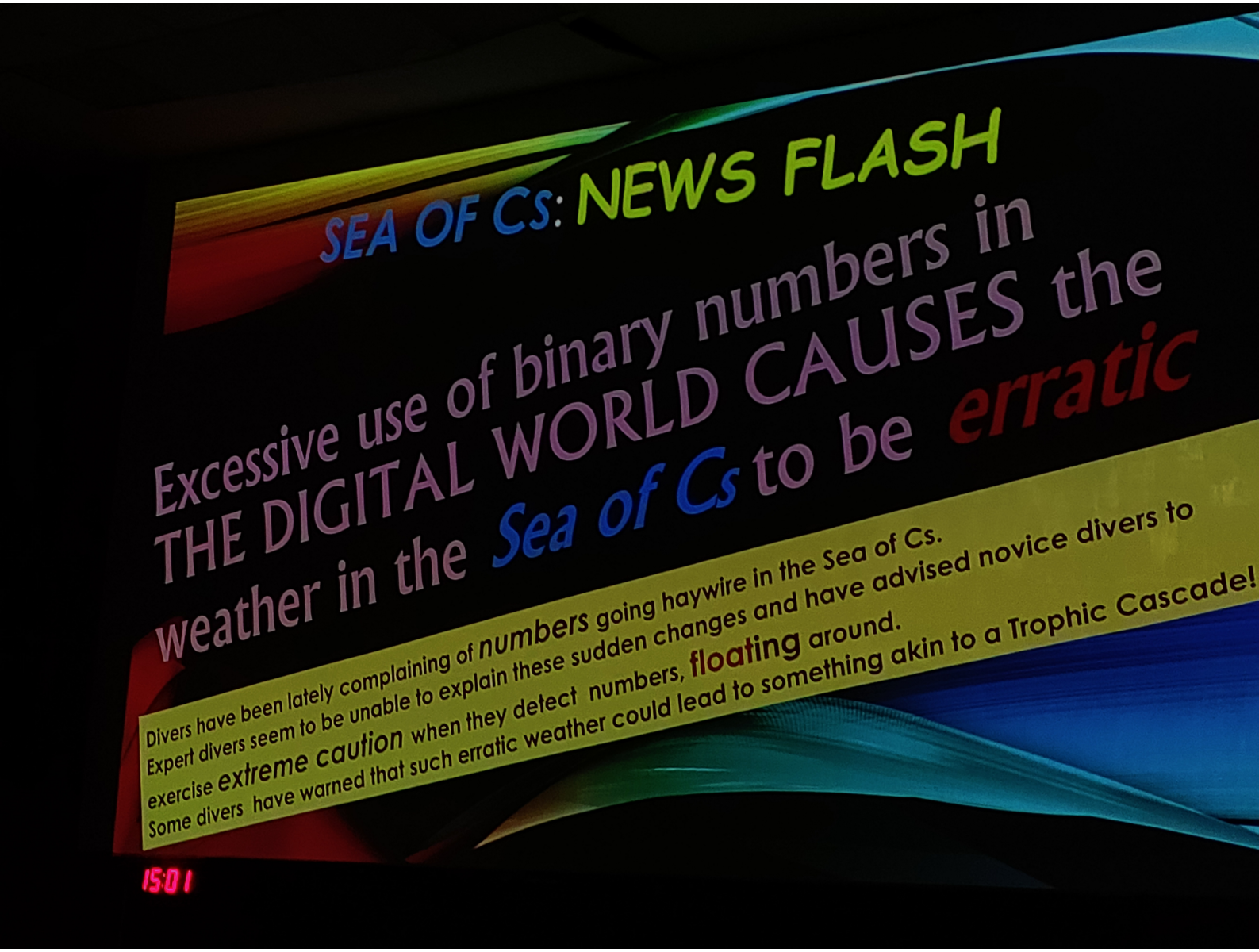
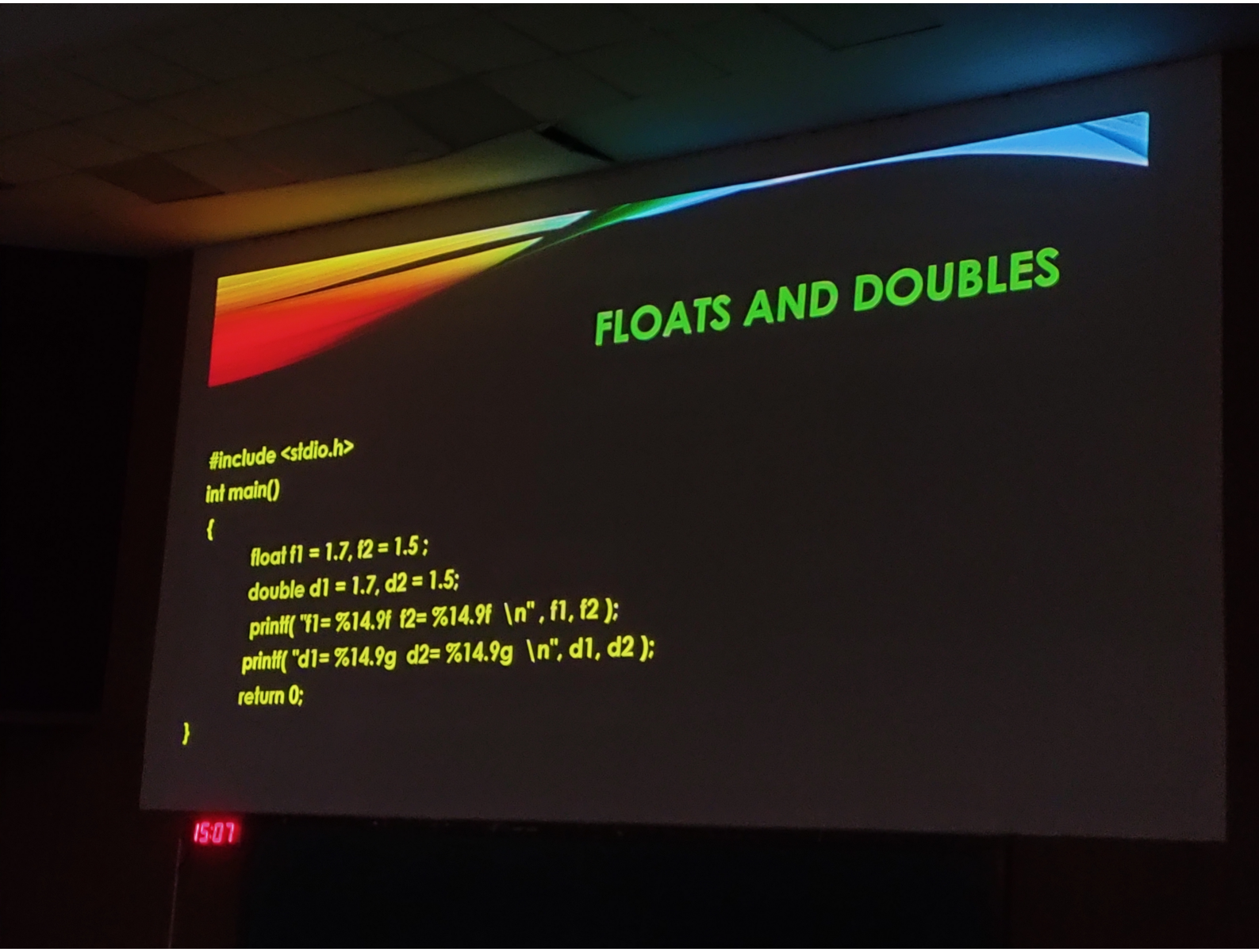

SEA OF Cs: NEWS FLASH
Excessive use of binary numbers in THE DIGITAL WORLD CAUSES the weather in the Sea of Cs to be erratic
Divers have been lately complaining of numbers going haywire in the Sea of Cs.
Expert divers seem to be unable to explain these sudden changes and have advised novice divers to exercise extreme caution when they detect numbers, floating around.
Some divers have warned that such erratic weather could lead to something akin to a Trophic Cascade!
15:01

– THE DIGITAL WORLD MAKES THE WEATHER
THE PROBLEM WITH BINARIES!
IN THE SEA OF Cs ERRATIC

- Since a DIGITAL computer uses binary numbers, often it is not possible to represent many floating point numbers accurately.
- Many decimal fractions (for instance numbers that end in 1, 2, 3, 4, 6, 7, 8, or 9 such as 0.1 or 0.2 or 0.142) cannot be accurately represented in binary.
- The closest number is thus used/stored. These conversions and rounding-offs, cause many errors.
- One way to mitigate this issue, to an extent, is to use the higher type viz. **double**.

02


FLOATS AND DOUBLES
#include <stdio.h>
int main()
{
float f1 = 1.7, f2 = 1.5 ;
double d1 = 1.7, d2 = 1.5;
printf("f1= %14.9f f2= %14.9f \n" , f1, f2);
printf("d1= %14.9g d2= %14.9g \n", d1, d2);
return 0;
}
15:07

FLOATS AND DOUBLES

```
#include <stdio.h>
int main()
{
    float f1 = 1.7, f2 = 1.5 ;
    double d1 = 1.7, d2 = 1.5;
    printf( "f1= %14.9f  f2= %14.9f  \n" , f1, f2 );
    printf( "d1= %14.9g  d2= %14.9g  \n", d1, d2 );
    return 0;
}
```

```
Output:
f1=     1.700000048   f2=     1.500000000
d1=             1.7   d2=             1.5
```

g stands for double

WHAT HAPPENED TO 2.?

```
#include <stdio.h>
int main()
{  float t;
   for( t=1.7; t<2.2; t=t+0.1 )
   {
      printf( "%4.2f\n", t );
   }
   return 0;
}
```

```
1.70
1.80
1.90
2.00
2.10
2.20
```

15:09

WHAT HAPPENED TO 2.?

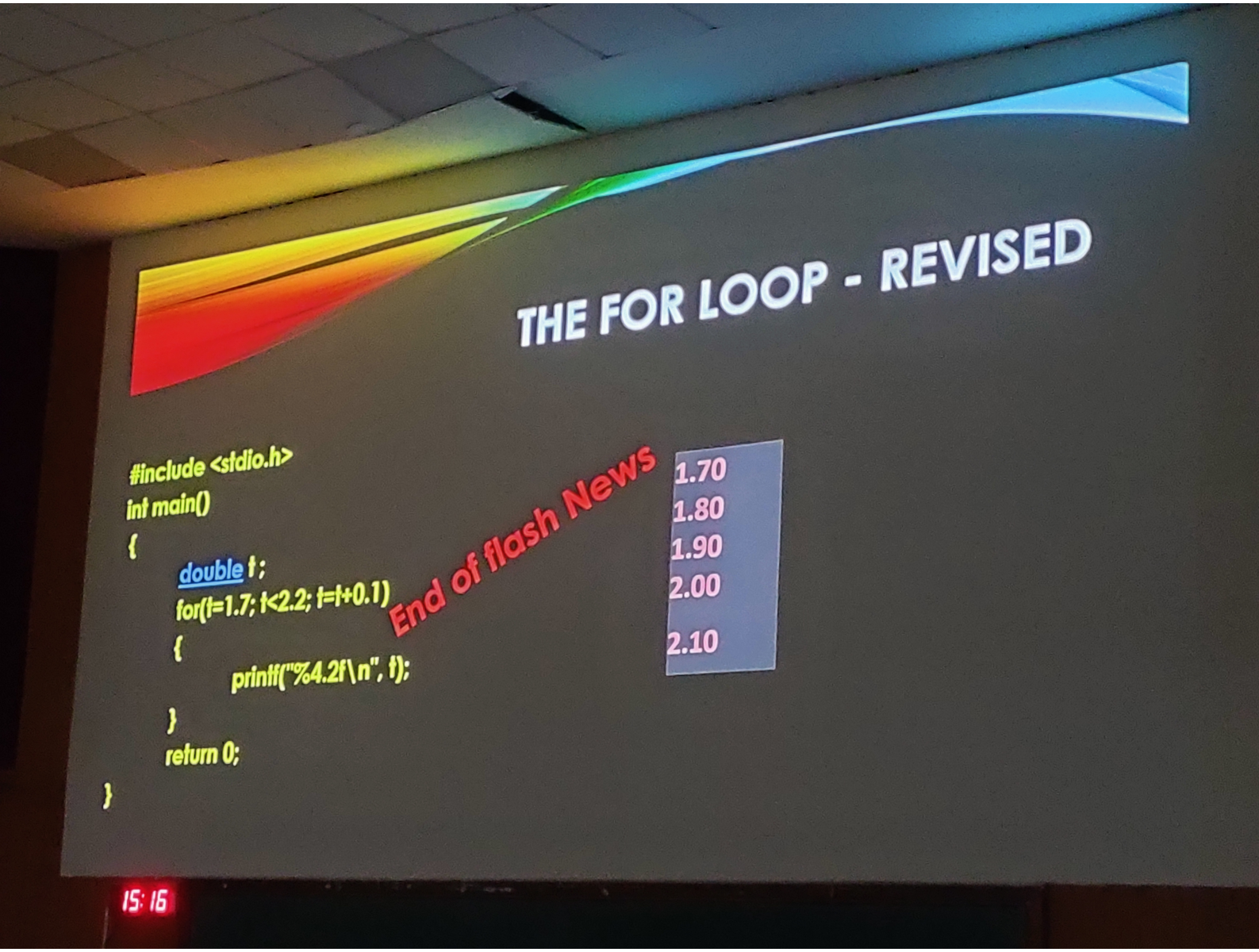
```
#include <stdio.h>
int main()
{  float t;
   for( t=1.7; t<2.2; t=t+0.1 )
   {
      printf( "%4.2f\n", t );
   }
   return 0;
}
```

```
1.70
1.80
1.90
2.00
2.10

2.20
```

```
#include <stdio.h>
int main()
{  float t;
   for( t=1.7; t<2.2; t=t+0.1 )
   {
      printf( "%14.9f\n", t );
   }
   return 0;
}
```

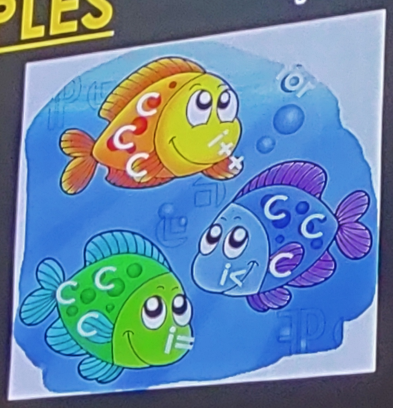
```
1.700000048
1.800000072
1.900000095
2.000000000
2.099999905

2.199999809
```

15:10


THE FOR LOOP - REVISED
#include <stdio.h>
int main()
{
double t ;
for(t=1.7; t<2.2; t=t+0.1)
{
printf("%4.2f\n", t);
}
return 0;
}
End of flash News
1.70
1.80
1.90
2.00
2.10
15: 16

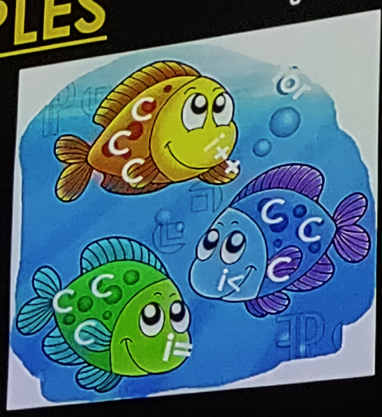
NESTED FOR LOOPS : EXAMPLES

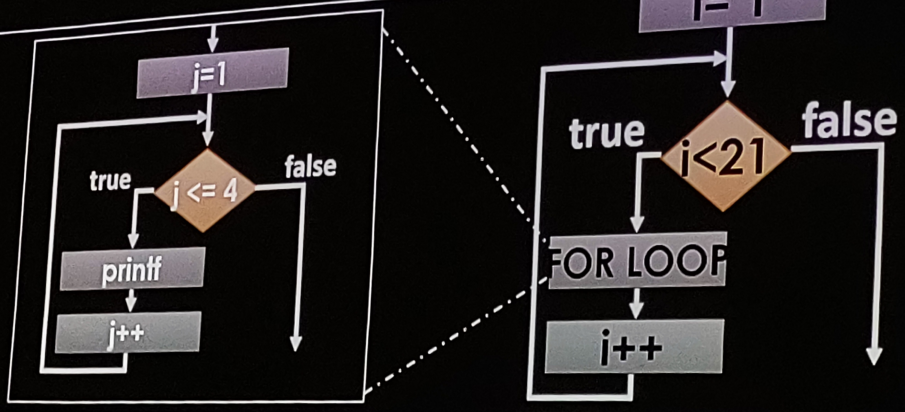

```
for(i=1; i<21; i++)
{
  for(j=1;  j<=4;  j++)
  {
    printf("%d * %d = %d\n",i,j, i*j);
  }
  printf( "\n" );
}
```



NESTED FOR LOOPS : EXAMPLES


```
for(i=1; i<21; i++)
{
  for(j=1;  j<=4;  j++)
  {
    printf("%d * %d = %d\n",i,j, i*j);
  }
  printf( "\n" );
}
```






1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
1 * 4 = 4
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
3 * 4 = 12
4 * 1 = 4
4 * 2 = 8
4 * 3 = 12
4 * 4 = 16
15:19

WHILE LOOP VERSUS FOR LOOP: MAKING A WHIRLPOOL

- Used for event driven case
- Mostly the event exits the infinite loop using the break statement

```
while( 1 )
{
    /* STARTING THE WHIRPOOL
       Loop without testing */
}
```

```
for( ; ; )
{
    /* STARTING THE WHIRPOOL
       Loop without testing */
}
```

WHILE LOOP VERSUS FOR LOOP: EXITING THE WHIRLPOOL

- Exiting an infinite loop using the **break** statement

```
while(1)
{
  scanf("%c", &c);
  if(c=='e' || c=='E')
  {
     printf("\nEntered %c, Bye!\n", c);
     break; // exit the while loop
  }
}
```

WHILE LOOP VERSUS FOR LOOP: EXITING THE WHIRLPOOL

- Exiting an infinite loop using the **break** statement

OR

```
while(1)
{
  scanf("%c", &c);
  if(c=='e' || c=='E')
  {
    printf("\nEntered %c, Bye!\n", c);
    break; //exit the while loop
  }
}
```

15:22

FOR-EVER OR INFINITE LOOP

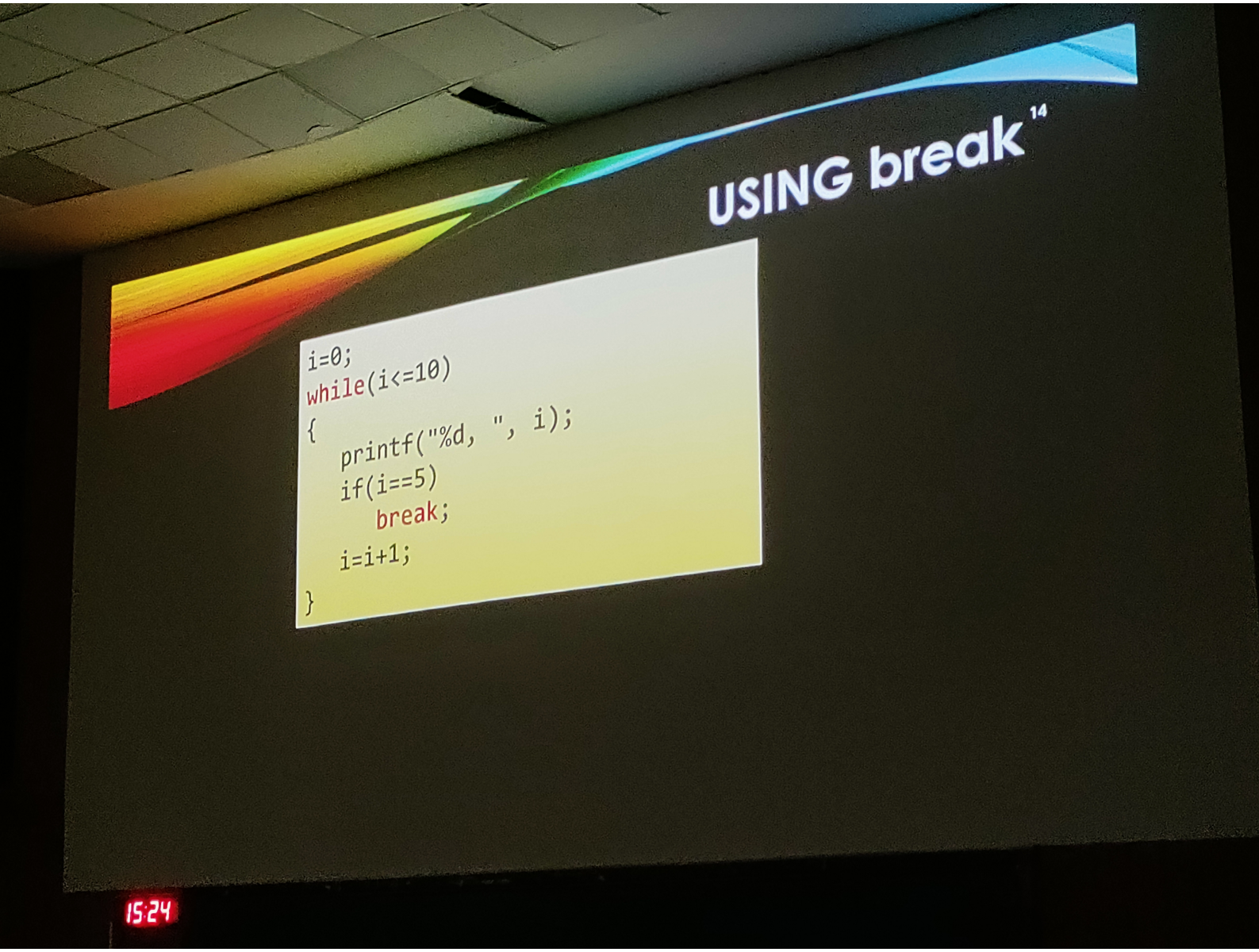
- Exiting an infinite loop using the break statement

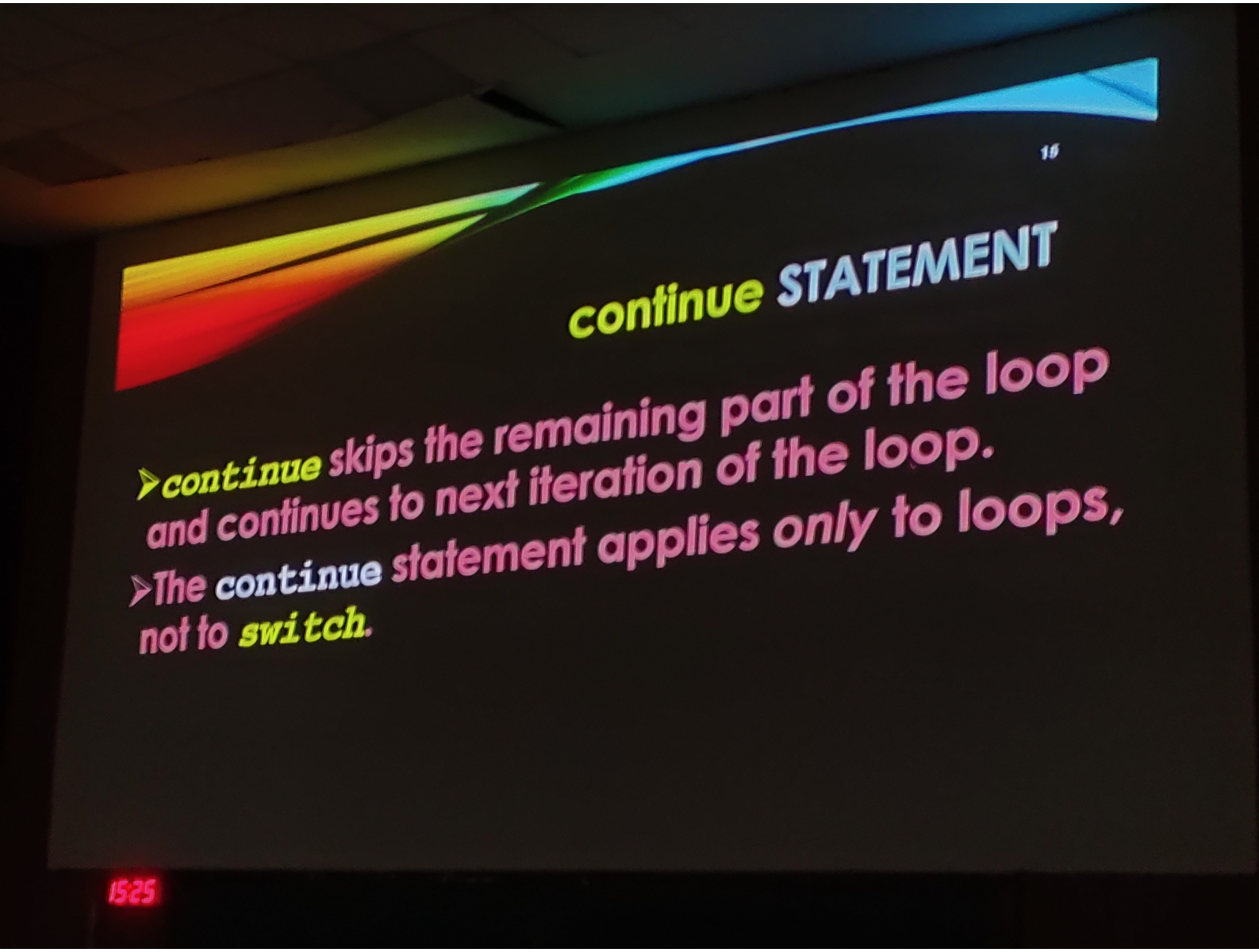
```
for(;;)
{
  scanf("%c",&c);
  if(c=='e' || c=='E')
  {
     printf("\nEntered %c, Bye!\n", c);
     break; // exit the for loop
  }
}
```

CHECK THIS...

Using break in a nested loops causes the innermost loop to be exited.

```
while(exp1)
{
   while(exp2)
   {
       statement1
       break;
   }
   statement2
}
```


USING break
14
i=0;
while(i<=10)
{
printf("%d, ", i);
if(i==5)
break;
i=i+1;
}
15:24


15
continue STATEMENT
continue skips the remaining part of the loop and continues to next iteration of the loop.
The continue statement applies only to loops, not to switch.
15:25

CONTINUING WITH continue

This reads ten numbers but prints square roots for only positive numbers.

```
for( j=0; j<10; j++ )
{
  scanf("%f", &v);
  if (v < 0)
     continue;
  else
  {
     sv = sqrt(v); /* #include<math.h> */
     printf("root is %f\n", sv);
  }
}
```

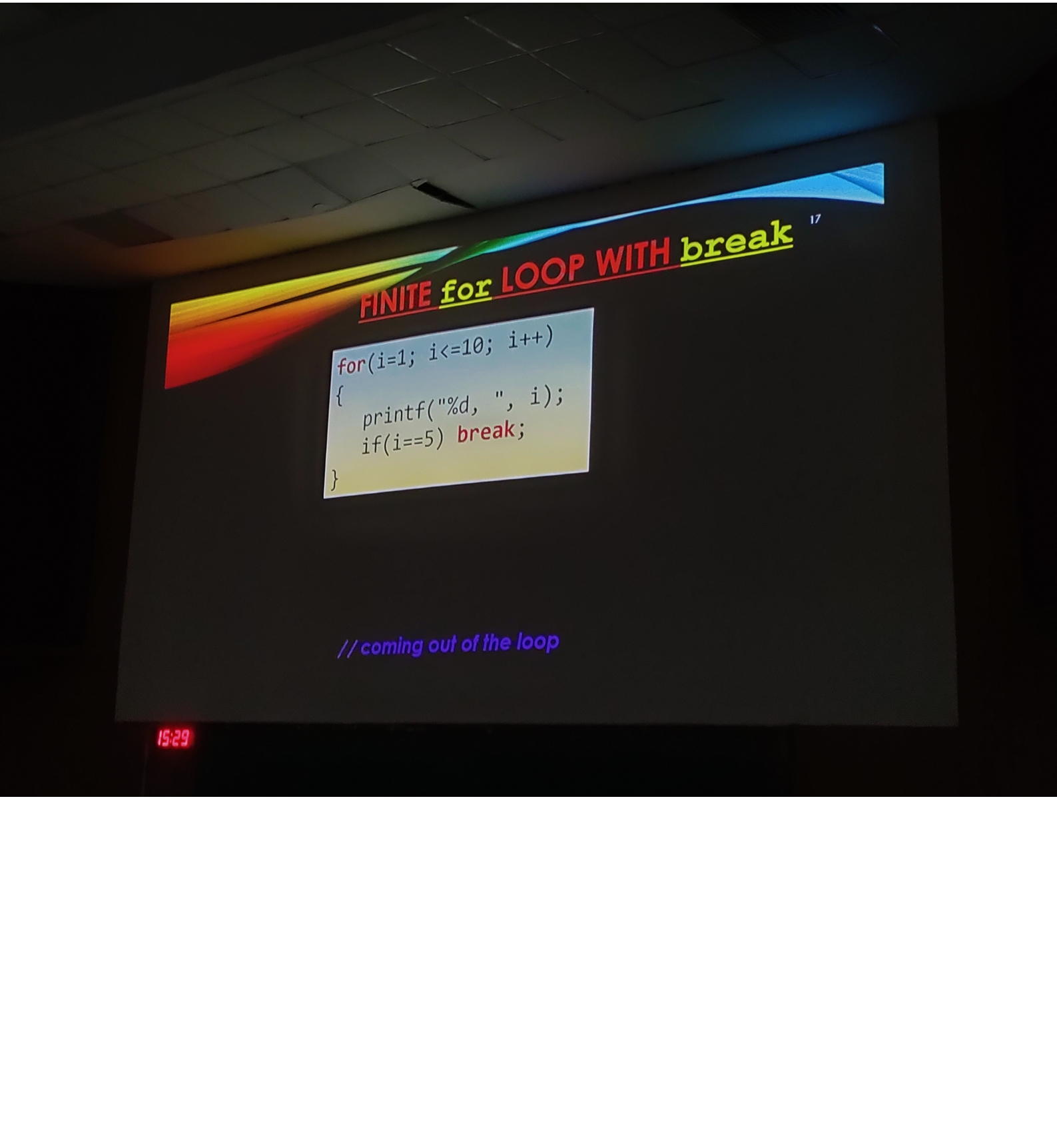
CONTINUING WITH continue

This reads ten numbers but prints square roots for only positive numbers.

```
for( j=0; j<10; j++ )
{
  scanf("%f", &v);
  if (v < 0)
     continue;
  else
  {
    sv = sqrt(v); /* #include<math.h> */
    printf("root is %f\n", sv);
  }
}
```

NB: In case of `for` loop it skips the rest of the loop, but it *does the increment step* before continuing with next iteration.

The *continue* statement applies only to loops, not to *switch*.


FINITE for LOOP WITH break
17
for(i=1; i<=10; i++)
{
printf("%d, ", i);
if(i==5) break;
}
// coming out of the loop
15:29

FINITE for LOOP WITH continue

```
for(i=1; i<=10; i++)
{
  if(i==5)
     continue;
  printf("%d\n", i);
}
```

// Skip a case if condition satisfied

for LOOP (RECAP)

The `for` statement syntax is:

```
for(expr1; expr2; expr3)
{   statements
}
```

This is equivalent to:

```
expr1;
while(expr2)
{
  statements
  expr3;
}
```

for (RECAP)...

```
for( expr1; expr2; expr3 )
        statement
```

- ✓ expr1 is the initialization
- ✓ expr2 is the test condition
- ✓ expr3 is the increment expression

for example...

Read 10 integers into an array

```
int a[10];
int j;
for(j=0; j<10; j++)
    scanf( "%d", &a[j] );
```

NB: *j* retains its value after the for loop is over.

COMMA ,

Comma can be used both as a Separator or as an operator

Separator

```
int j,k,l;
printf("%d %d %c", j,k,c);
```

Comma EXAMPLE USING for

```
/* a[10] is an array of 10 elements*/
for( j=0,k=9; j < k; j++,k-- )
{
      t = a[j];
      a[j] = a[k];
      a[k] = t;
}
```

Comma EXAMPLE USING for


```
/* a[10] is an array of 10 elements*/
for( j=0,k=9; j < k; j++,k-- )
{
      t = a[j];
      a[j] = a[k];
      a[k] = t;
}
```

This will reverse the order of elements in the array **a**

Comma EXAMPLE USING for

```
/* a[10] is an array of 10 elements*/
for( j=0,k=9; j < k; j++,k-- )
{
      t = a[j];
      a[j] = a[k];
      a[k] = t;
}
```

operator

1
2
3
4
5

5
4
3
2
1

This will reverse the order of elements in the array **a**

do-while LOOPS

Execute the loop first and then test the condition

Syntax:

```
do
{
   statements
} while( expression );
```

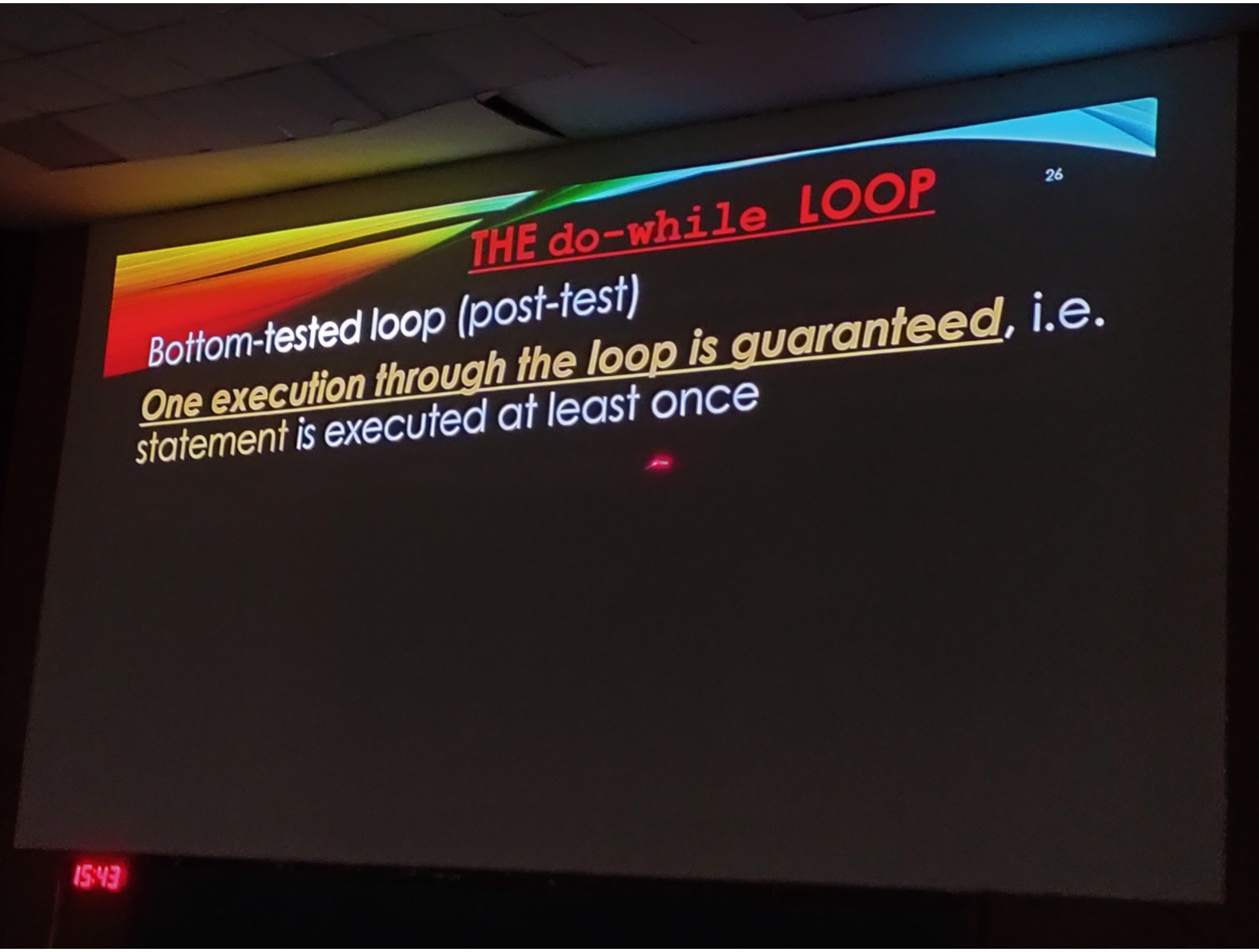
do-while LOOPS

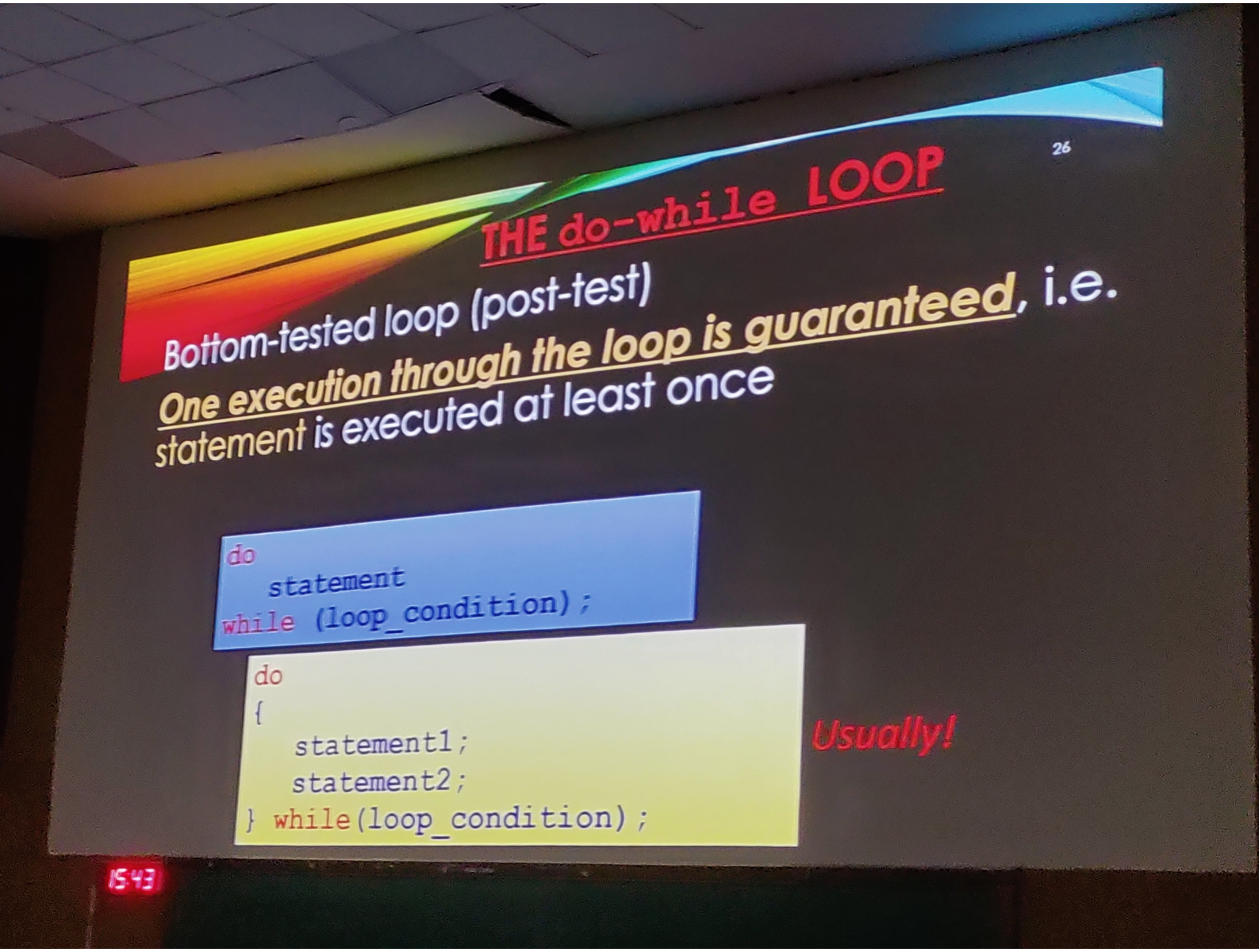
Execute the loop first and then test the condition

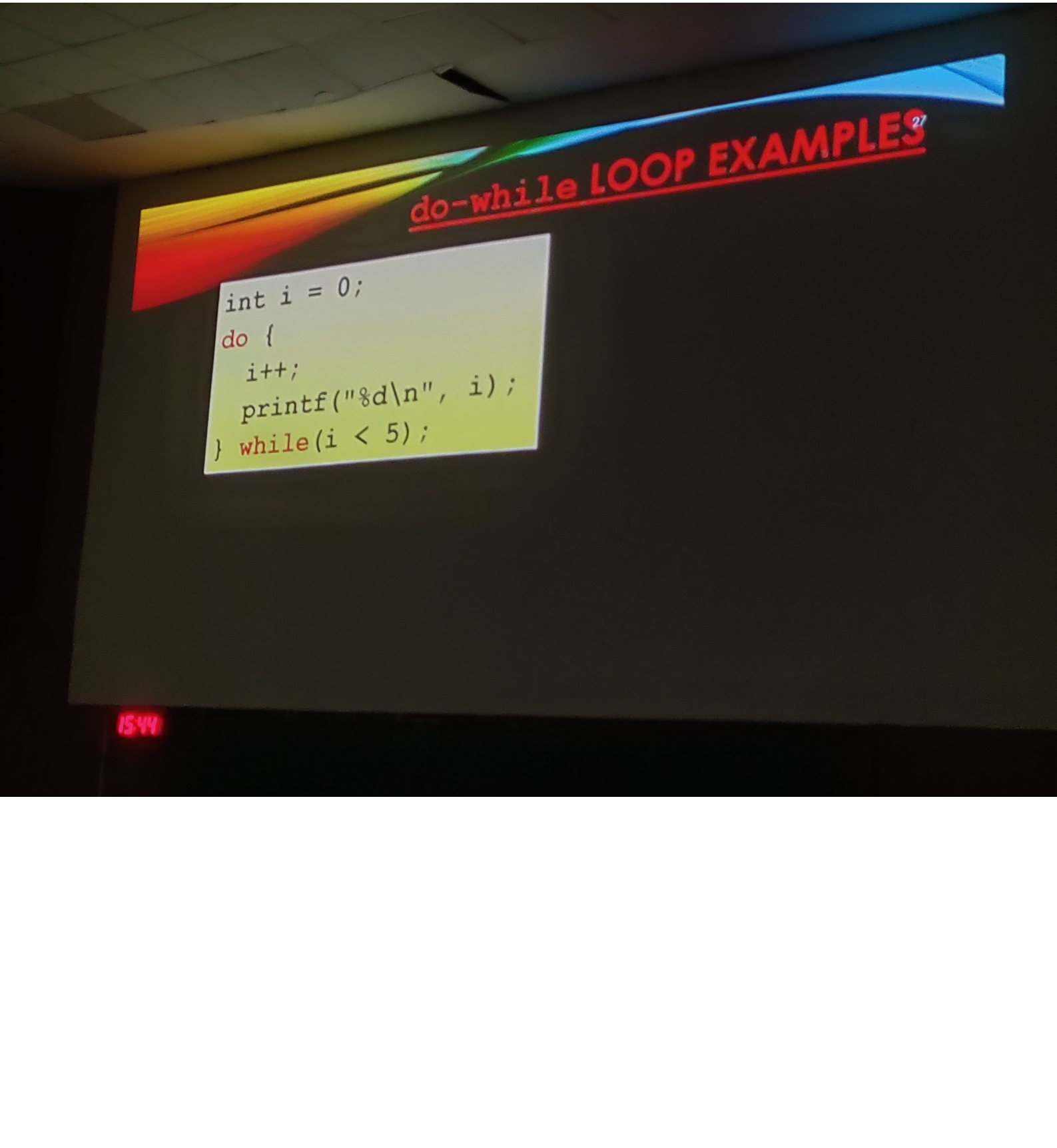
Syntax:

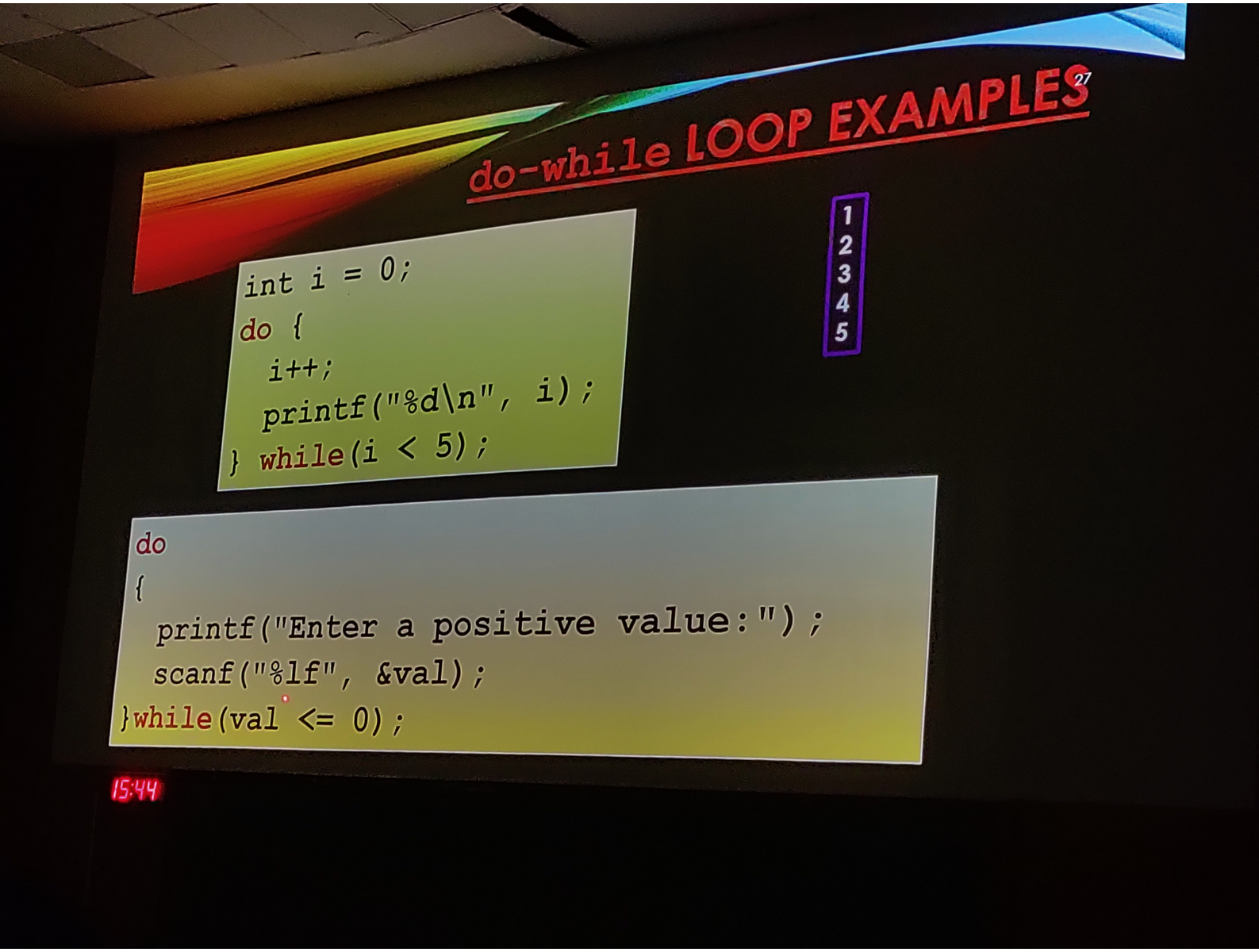
```
do
{
  statements
} while( expression );
```

The statements are executed, then expression is evaluated. If it is TRUE, the statement(s) is evaluated again and so on.


THE do-while LOOP
26
Bottom-tested loop (post-test)
One execution through the loop is guaranteed, i.e.
statement is executed at least once
15:43


26
THE do-while LOOP
Bottom-tested loop (post-test)
One execution through the loop is guaranteed, i.e.
statement is executed at least once
do
statement
while (loop_condition);
do
{
statement1;
statement2;
} while(loop_condition);
Usually!
15:43


do-while LOOP EXAMPLES
27
int i = 0;
do {
i++;
printf("%d\n", i);
} while(i < 5);
15:44


do-while LOOP EXAMPLES
27
int i = 0;
do {
i++;
printf("%d\n", i);
} while(i < 5);
1
2
3
4
5
do
{
printf("Enter a positive value:");
scanf("%lf", &val);
}while(val <= 0);
15:44

do-while example


```
char ch[128];
int j = 0;
do
{
   ch[j] = getchar();
   j++;
} while( ch[ j-1 ] != '\n');

ch[ j-1 ] = '\0';
```

do-while example


```
char ch[128];
int j = 0;
do
{
   ch[j] = getchar();
   j++;
} while( ch[ j-1 ] != '\n');

ch[ j-1 ] = '\0';
```

abdefgh

Remember that there is a ; after while(...)

do-while example


```
char ch[128];
int j = 0;
do
{
   ch[ j++ ] = getchar();
} while( ch[ j-1 ] != '\n');
ch[ j-1 ] = '\0';
```

```
ch[j] = getchar();
j++;
```

15:48

NESTED LOOPS:


```
int j, a[5], m;  char ch;
printf( "Enter 5 integers separated by return\n" );
do
{
   for(j=0; j<5; j++)
      scanf("%d", &a[j]);
   m = a[0]; j = 1;
   while( j < 5 )
   {
      if( m < a[j] )
         m = a[j];
      j++;
   }
   printf("    number is %d\n", m);
   printf( "Want to enter 5 integers again? (y/n) " );
   ch = getchar();
}while(ch == 'y' ||  ch == 'Y');
```

NESTED LOOPS:


```
int j, a[5], m;  char ch;
printf( "Enter 5 integers separated by return\n" );
do
{
   for(j=0; j<5; j++)
      scanf("%d", &a[j]);
   m = a[0]; j = 1;
   while( j < 5 )
   {
      if( m < a[j] )
         m = a[j];
      j++;
   }
   printf("    number is %d\n", m);
   printf( "Want to enter 5 integers again? (y/n) " );
   ch = getchar();
}while(ch == 'y' ||  ch == 'Y');
```

Inner loop

Inner loop

Outer loop