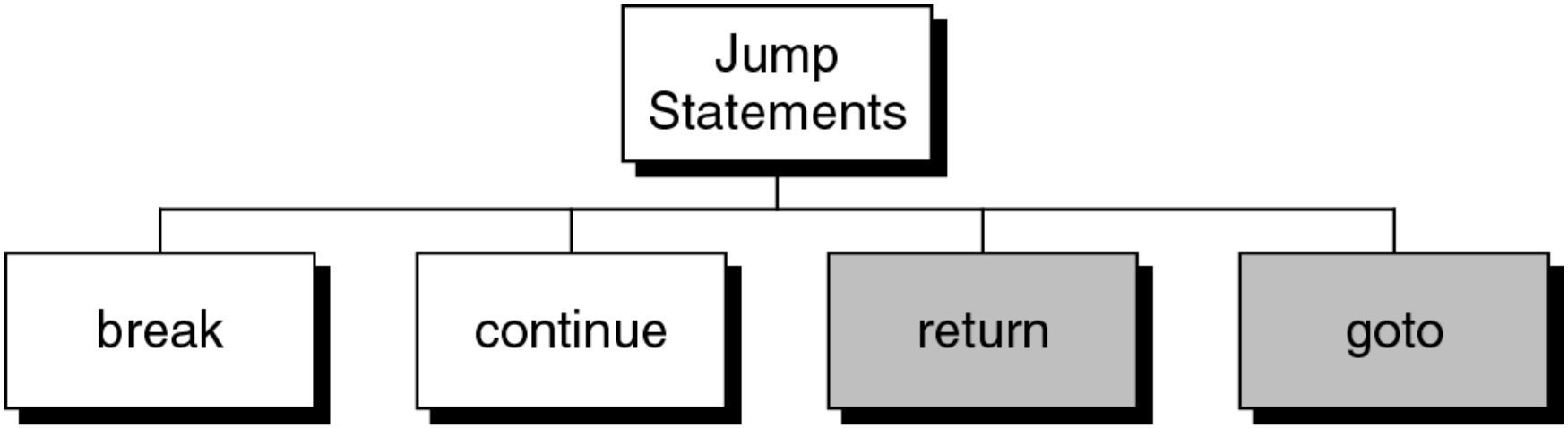
CSE101-Lec 7
Jump statements

break , continue, goto, return

Jump statements

- You have learn that, the repetition of a loop is controlled by the loop condition.
- C provides another way to control the loop, by using **jump statements.**
- There are four jump statements:



`break` statement

- `break` is a keyword.
- `break` allows the programmer to **terminate** the loop.
- A `break` statement causes control to transfer to the first statement after the loop or block.
- The `break` statement can be used in nested loops. If we use `break` in the innermost loop then the control of the program is terminated only from the innermost loop.

break statement

```
##include<stdio.h>
int main()
{
 int n;
 for (n=10; n>0; n=n-1){
  if (n<8)
   break;
  printf("%d ", n);
 } //end for
}
```

Program to show use of break statement.

```
10 9 8
```

Q1

What will be the output of the following C code?

```
#include<stdio.h>
int main()
{
int i;
for(i=1;i<=5;i++)
{
if(i==2||i==3)
continue;
printf("\nHello");
}
return 0;
}
```

A. Hello will be printed 3 times

B. Hello will be printed 5 times

C. Hello will be printed for one time

D. Nothing will be displayed

Q2

What will be the output of the following C code?

```
#include<stdio.h>
int main()
{
int i;
for(i=1;i<=5;i++)
{
   printf("\nHello");
   break;
}
return 0;
}
```

A. Hello will be printed 4 times
B. Hello will be printed 5 times
C. Hello will be printed for one time
D. Nothing will be displayed

Q3

```
//What will be the output of following
code?
#include<stdio.h>
int main()
{
int i=0;
while(1)
{
   i++;
   if(i==2)
   continue;
   else if(i==5)
   break;
   printf("%d ",i);
}
return 0;
}
```

A. 1 3 4

B. 2 5

C. 1 3

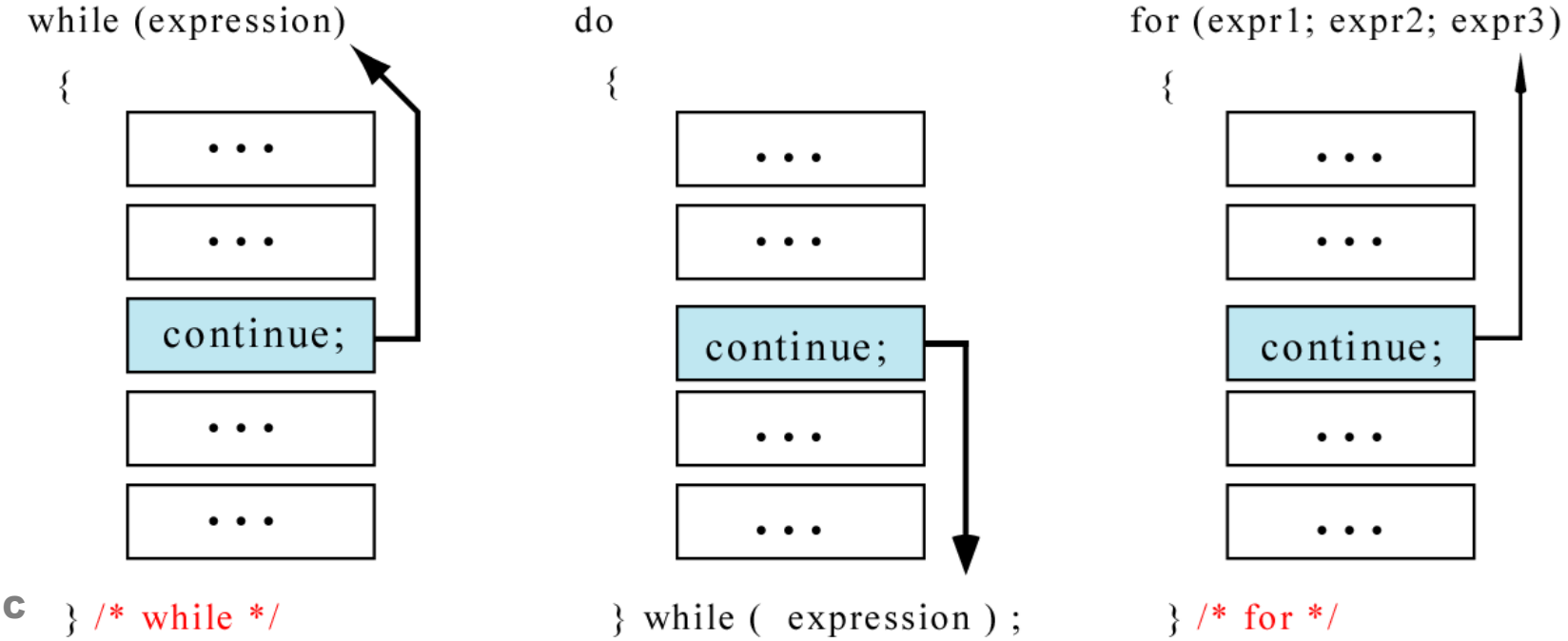
D. Nothing will be displayed

`continue` statement

- `continue` statement is exactly opposite to `break`.
- `continue` statement is used for continuing the next iteration of the loop statements
- When it occurs in the loop, it does not terminate, but skips the statements after this statement

`continue` statement

- In `while` and `do…while` loops, the continue statement transfers the control to the loop condition.
- In `for` loop, the continue statement transfers the control to the updating part.



continue statement

```
#include<stdio.h>
int main()
{
 int n;
 for (n=10; n>0; n=n-1){
  if (n%2==1)
     continue;
   printf("%d ", n);
 }
}
```

```
10 8 6 4 2
```

Program to show the use of continue statement in for loop

continue statement

```
#include<stdio.h>
int main()
{
 int n = 10;
 while(n>0){
  printf("%d", n);
  if (n%2==1)
    continue;
  n = n -1;
 }
return 0;
}
```

For n=9, loop goes to infinite execution

Program to show the use of continue statement in for loop

```
10 9 9 9 9 9 ……
```

The loop then prints number 9 over and over again. It never stops.

Q1

What will be the output of the following C code?

```
#include <stdio.h>
int main()
{
    int i = 0;
    do
    {
        i++;
        if (i == 2)
            continue;
            printf("In while loop ");
    } while (i < 2);
    printf("%d\n", i);
    return 0;
}
```

A. In while loop 2
B. In while loop In while loop 3
C. In while loop 3
D. Infinite loop

Q2

```
#include<stdio.h>
int main()
{
int i=1;
while(i<=5)
{
    i++;
    continue;
    printf("\nC");
}
return 0;
}
```

A. C will be printed 5 times
B. C will be printed 0 times
C. C will be printed 1 time
D. None of these

Q3

What will be the output of following code?

```
#include<stdio.h>
int main()
{
int i;
for(i=1;i<=5;i++)
{
  printf("%d ",i);
  continue;
}
return 0;
}
```

A. 1 2 3 4 5

B. 1

C. 1 2

D. Nothing will be displayed

Q4

```
//What will be the output of following code?
#include<stdio.h>
int main()
{
int i=0;
for(;;)
{
   i++;
   if(i%3==0)
      continue;
   else if(i%5==0)
      break;
   printf("%d ",i);

}
return 0;
}
```

A. 1 2

B. 1 2 3 4 5

C. 1 2 4

D. Nothing will be displayed

break vs continue

BASIS FOR COMPARISON	BREAK	CONTINUE
Task	It terminates the execution of remaining iteration of the loop.	It terminates only the current iteration of the loop(or it is used to skip the current iteration and the statements following the continue statement)
Control after break/continue	'break' resumes the control of the program to the end of loop enclosing that 'break'.	'continue' resumes the control of the program to the next iteration of that loop enclosing 'continue'.
Causes	It causes early termination of loop.	It causes early execution of the next iteration.
Continuation	'break' stops the continuation of loop.	'continue' do not stops the continuation of loop, it only stops the current iteration.
Other uses	'break' can be used with 'switch', 'label'.	'continue' can not be executed with 'switch' and 'labels'.

goto

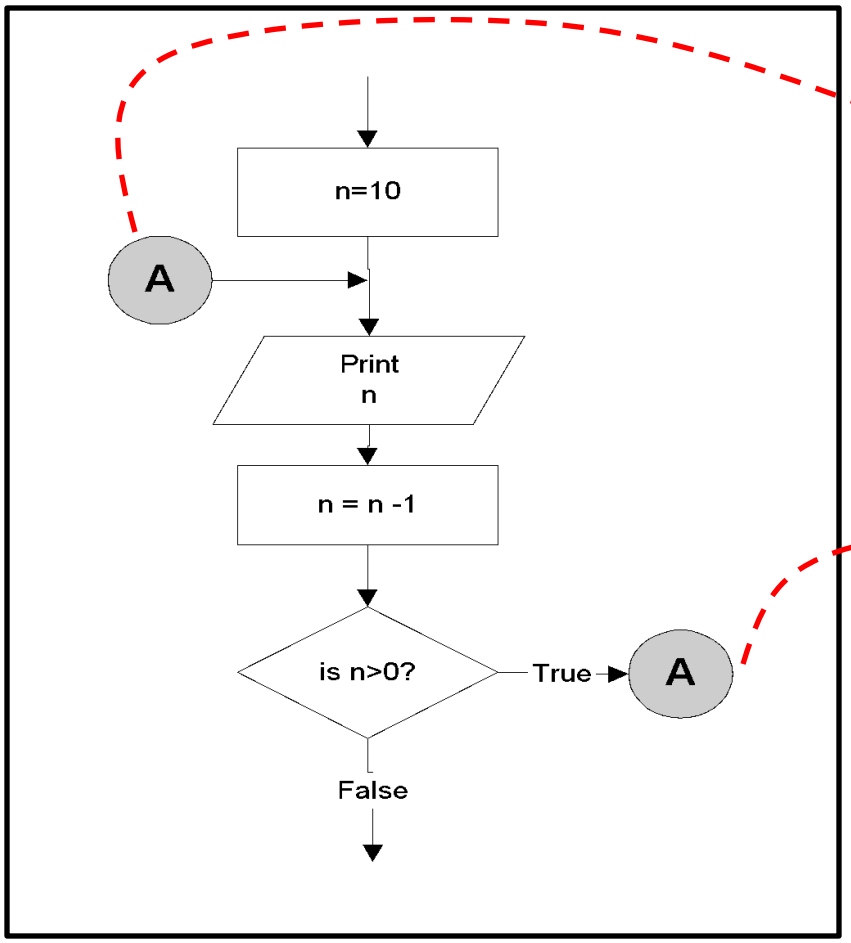
- **Unconditionally transfer control.**
- `goto` may be used for transferring control from one place to another.
- The syntax is:

```
goto identifier;
```

Control is unconditionally transferred to the location of a local label specified by *identifier*. For example,

```
Again:
...
goto Again;
```

goto statement




```
n=10;

A:
   printf("%d ", n);
   n = n -1;

   if (n>0)
         goto A;
```

```
Output:

10 9 8 7 6 5 4 3 2 1
```

Program to show goto statement.

```
#include<stdio.h>
int main()
{
 int x;
 printf("enter a number: ");
 scanf("%d",&x);
 if(x%2==0)
        goto even;
 else
        goto odd;
 even:
   printf(" %d is even", x);
   return;
 odd:
   printf("%d is odd", x);
   return 0;
}
```

```
enter a number: 18
18 is even
```

`return` statement

- **Exits the function.**
- return exits immediately from the currently executing function to the calling routine, optionally returning a value. The syntax is:
- return [*expression*];
- For example,

```
int sqr (int x){
    return (x*x);
 }
```