Week 7 Lecture 1

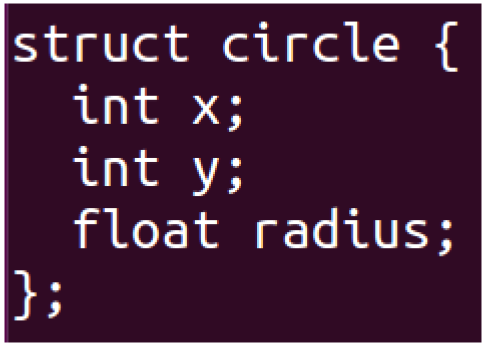
Structures;

Break and Continue

Structures

Structures are related data

- A circle can be central point and a radius.
- The central point is described by two ints called x and y.
- The radius is describe by a float called radius.

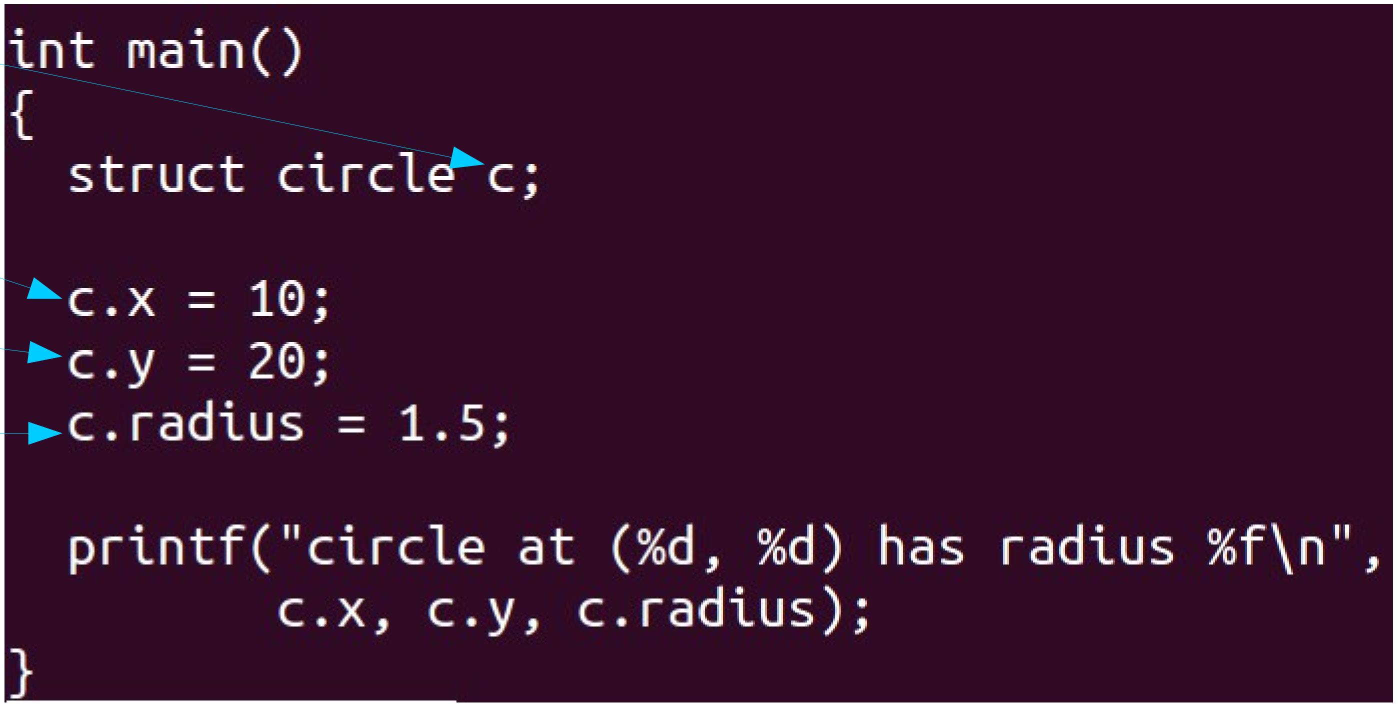


```
struct circle {
  int x;
  int y;
  float radius;
};
```

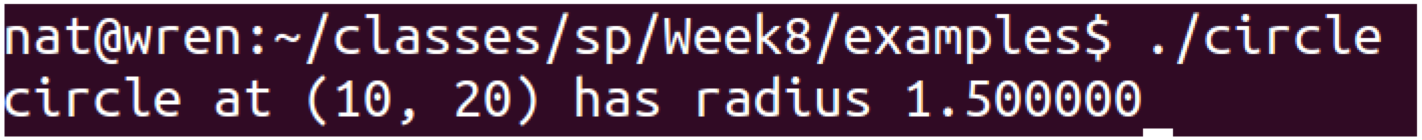
- Whenever a struct circle is created a new element is created with all three values.

Access Structure Elements using Dot Notation

- Variable c
- X
- Y
- Radius

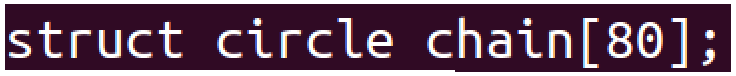


```
int main()
{
  struct circle c;

  c.x = 10;
  c.y = 20;
  c.radius = 1.5;

  printf("circle at (%d, %d) has radius %f\n",
         c.x, c.y, c.radius);
}
```



```
nat@wren:~/classes/sp/Week8/examples$ ./circle
circle at (10, 20) has radius 1.500000
```

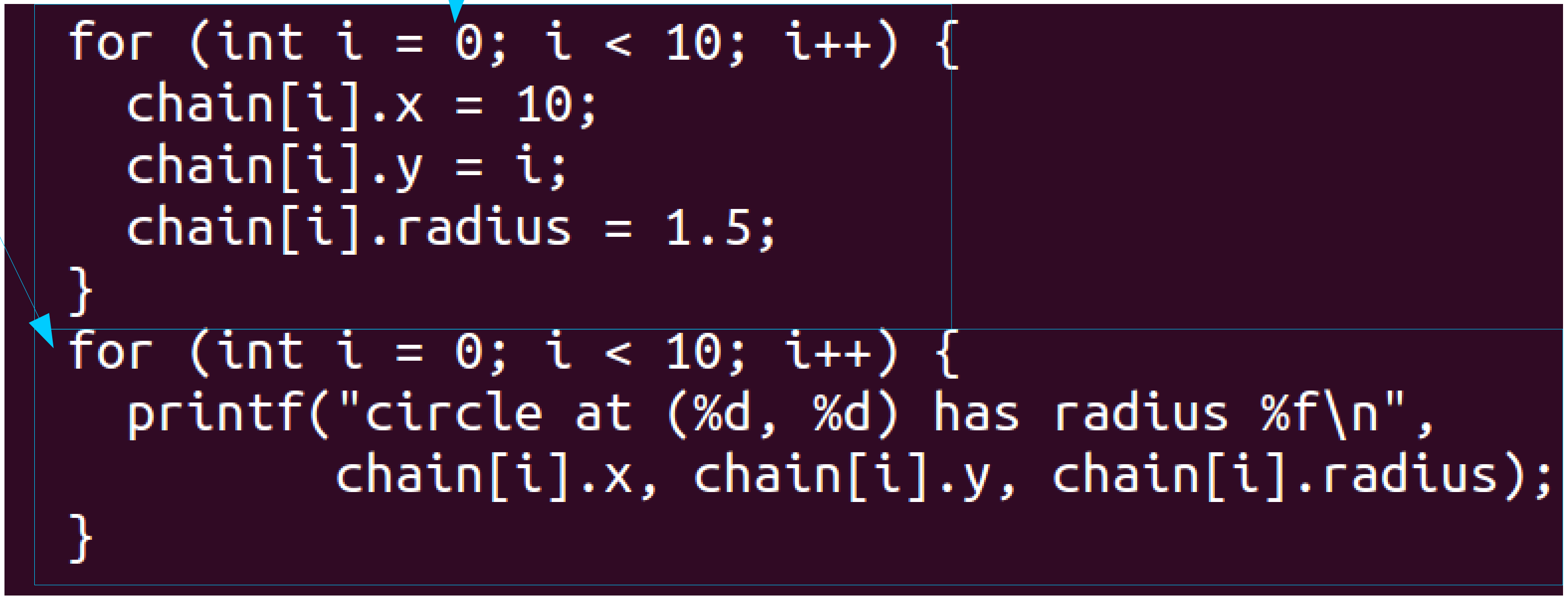
Arrays of Structures

- You define an array of structures just like you define an array of anything else
  - E.g., 

    ```
    struct circle chain[80];
    ```
  - This creates an array of 80 circles.
-

Set and Use Structure Arrays as usual

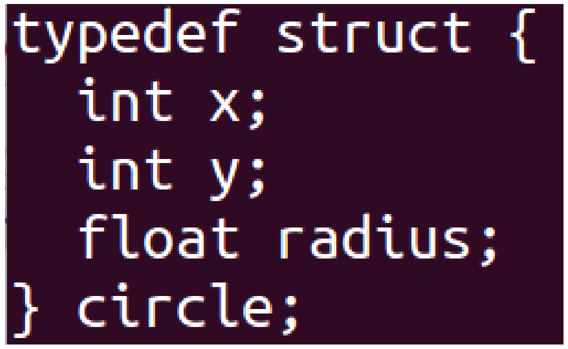
- Initialize the array
- Print the array



```
for (int i = 0; i < 10; i++) {
  chain[i].x = 10;
  chain[i].y = i;
  chain[i].radius = 1.5;
}
for (int i = 0; i < 10; i++) {
  printf("circle at (%d, %d) has radius %f\n",
         chain[i].x, chain[i].y, chain[i].radius);
}
```

Define new types with typedef

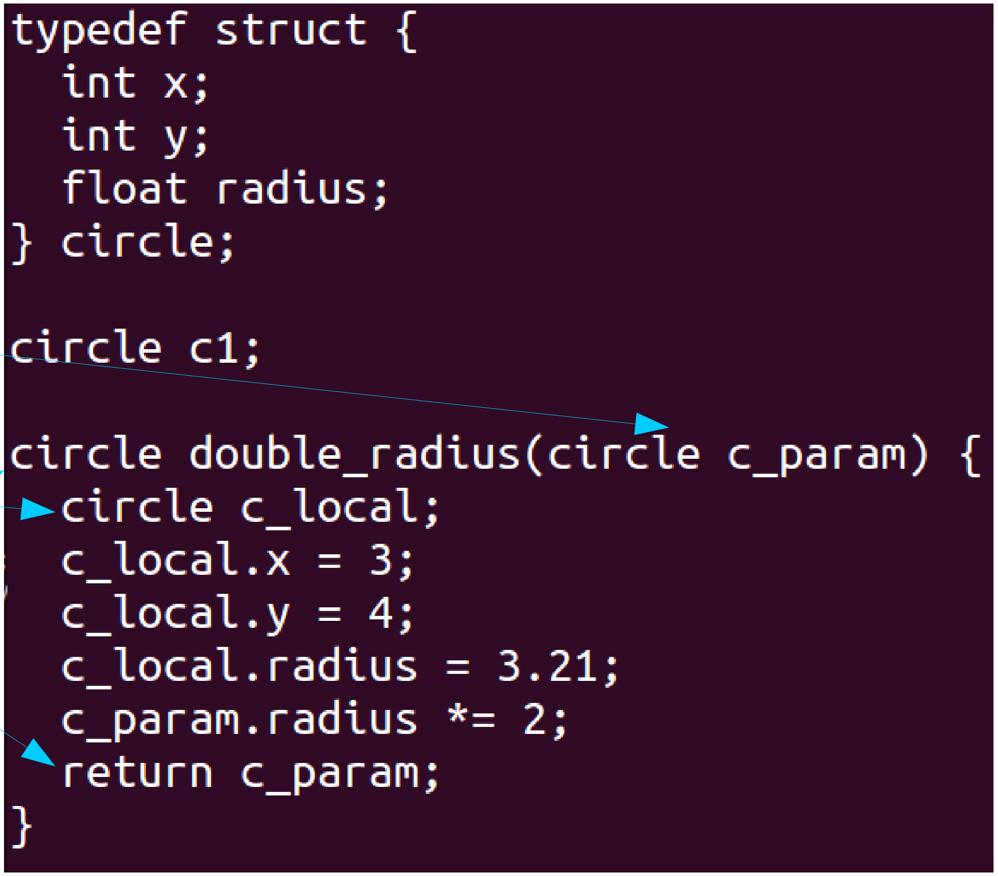
- Type definition common with struct
- Define a circle type
  - Ints x and y
  - Float radius
- Allocates no memory



```
typedef struct {
  int x;
  int y;
  float radius;
} circle;
```

Define new defined type variable

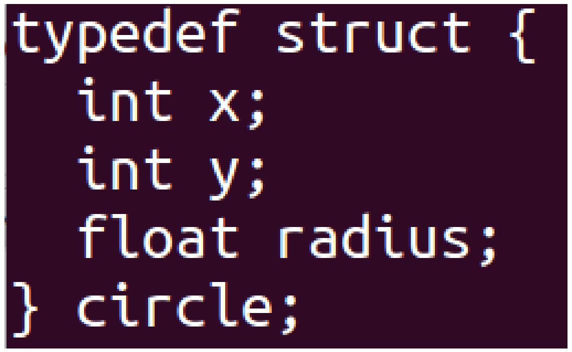
- Type definition
- Global circle
- Parameter is circle
- Local circle
- Function returns circle



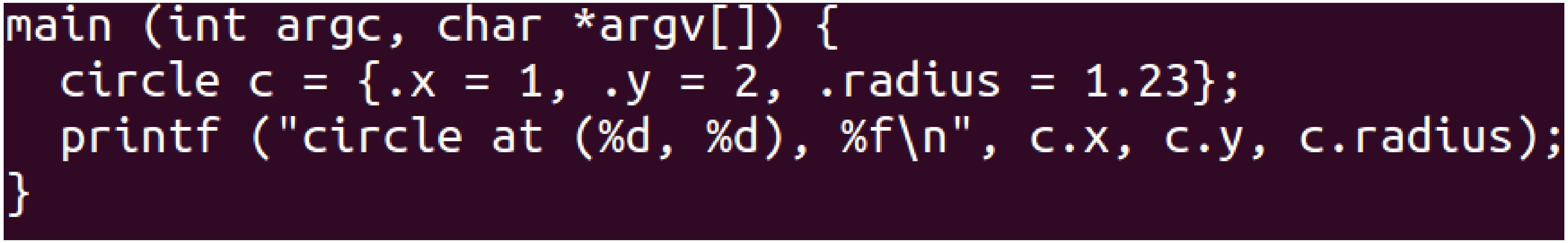
```
typedef struct {
  int x;
  int y;
  float radius;
} circle;

circle c1;

circle double_radius(circle c_param) {
  circle c_local;
  c_local.x = 3;
  c_local.y = 4;
  c_local.radius = 3.21;
  c_param.radius *= 2;
  return c_param;
}
```

Struct initialization

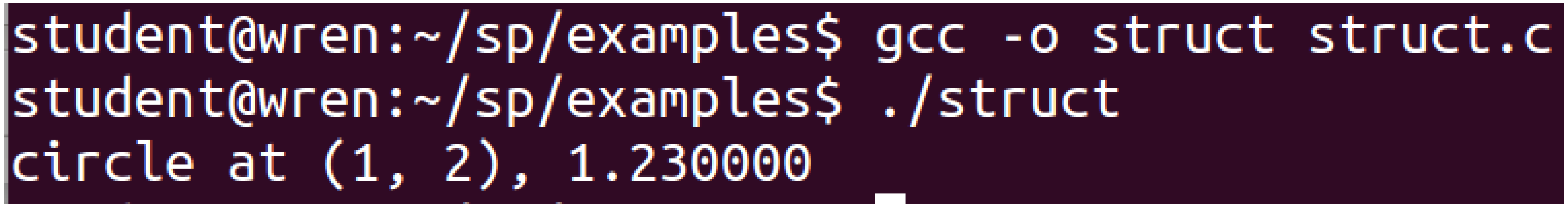
- Structs can be initialized.
- Easiest with defined type



```
typedef struct {
  int x;
  int y;
  float radius;
} circle;
```



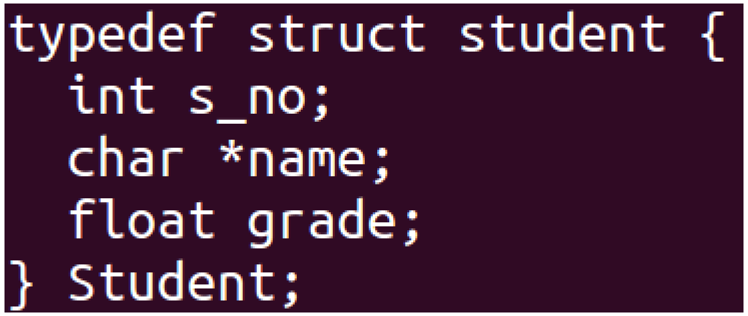
```
main (int argc, char *argv[]) {
  circle c = {.x = 1, .y = 2, .radius = 1.23};
  printf ("circle at (%d, %d), %f\n", c.x, c.y, c.radius);
}
```



```
student@wren:~/sp/examples$ gcc -o struct struct.c
student@wren:~/sp/examples$ ./struct
circle at (1, 2), 1.230000
```

Structure contain Related Data

- E.g. Student
  - Serial number
  - Name
  - Grade
- Represents a real world object
  - A particular student
- Note: name is a string, that is, an array or characters, so structures can contain arrays.

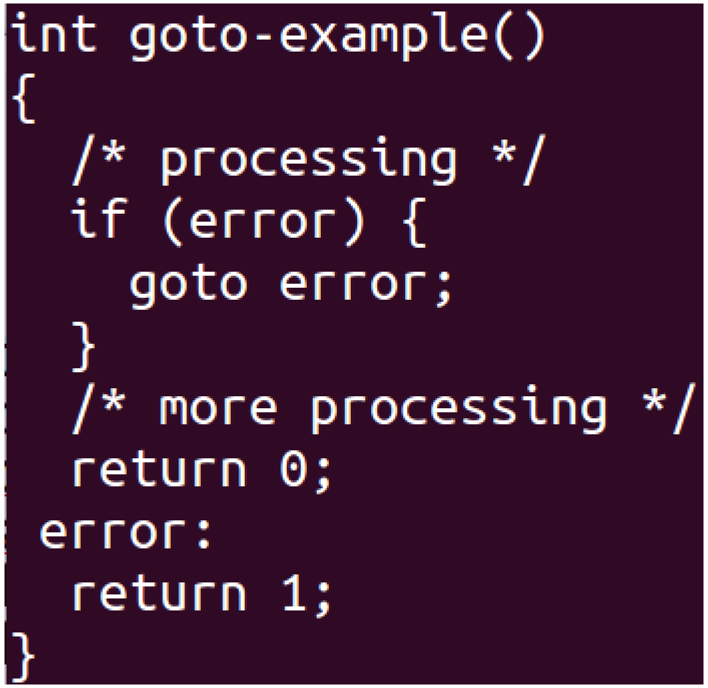


```
typedef struct student {
  int s_no;
  char *name;
  float grade;
} Student;
```

Goto, Break and Continue

Goto Statement

- You may defined labels anywhere in a program.
- A goto statement takes a label as an argument and executes the line at the label next.
- Example:



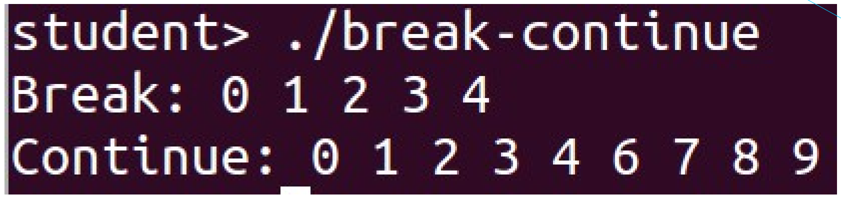
```
int goto-example()
{
  /* processing */
  if (error) {
    goto error;
  }
  /* more processing */
  return 0;
 error:
  return 1;
}
```

Break and Continue

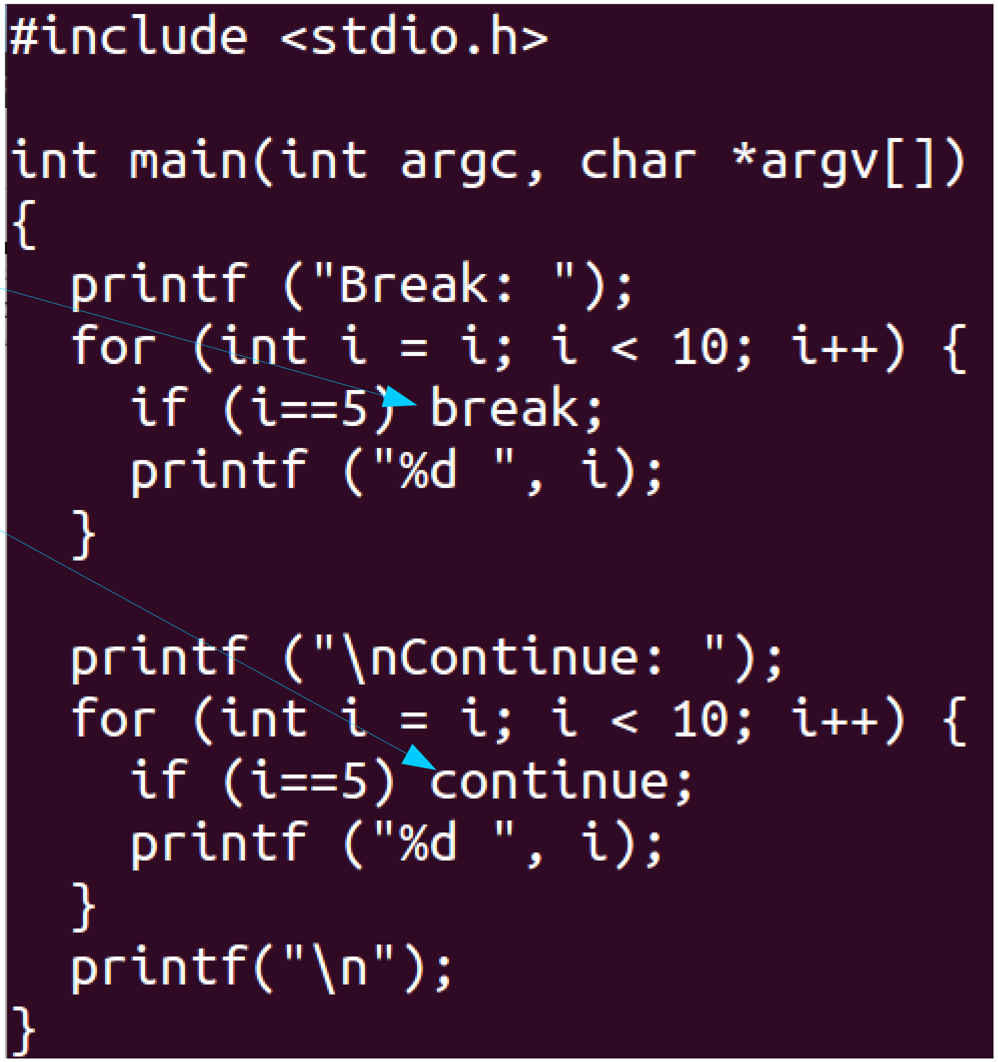
- Like goto, break and continue change the flow of control
  - Break exits the block.
    - We have see them in case statements
  - Continue exits the block, but not the loop.
    - Lets you some steps in a loop.

Break and Continue Example

- Break
- Continue
- Output:

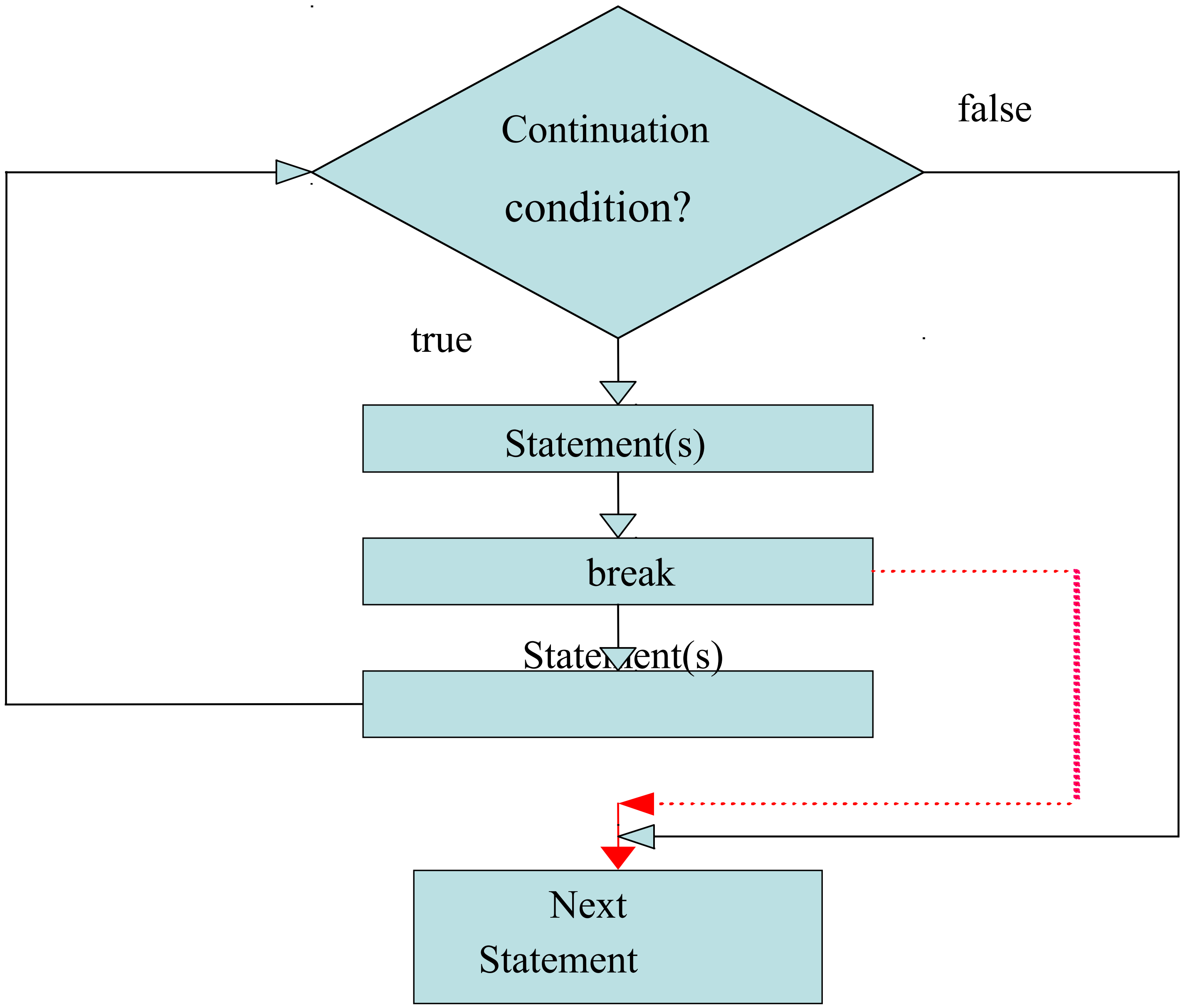


```
student> ./break-continue
Break: 0 1 2 3 4
Continue: 0 1 2 3 4 6 7 8 9
```

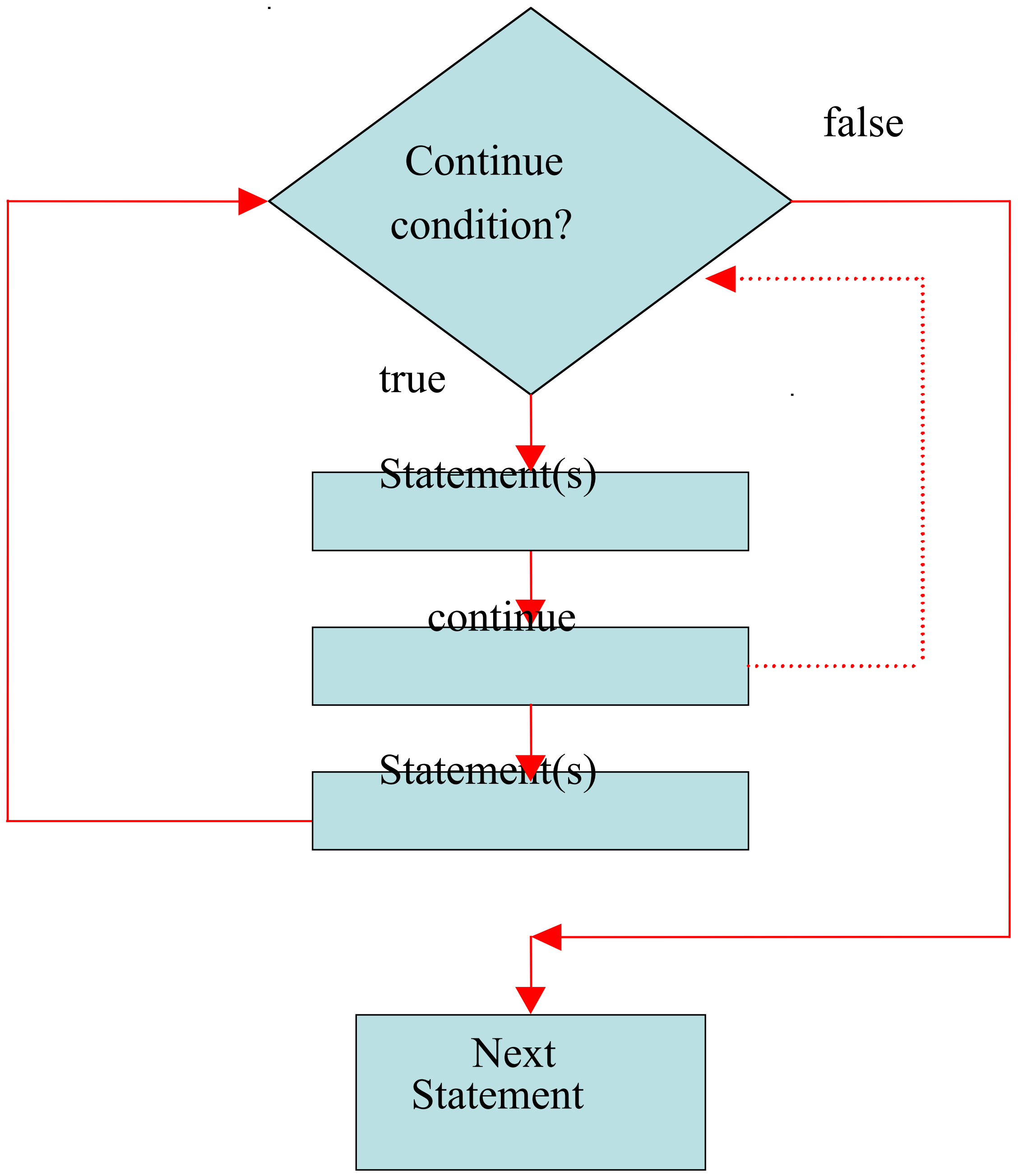


```
#include <stdio.h>

int main(int argc, char *argv[])
{
  printf ("Break: ");
  for (int i = i; i < 10; i++) {
    if (i==5) break;
    printf ("%d ", i);
  }

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

Flowchart for break.



Flowchart for continue


Continue condition?
false
true
Statement(s)
continue
Statement(s)
Next Statement