HIGH-LEVEL PROGRAMMING I

Jump Statements

by Prasanna Ghali

Jump Statements

- Jump statements alter sequential flow of control of C/C++ programs
- 4 jump statements:
  - `return` statement
  - `break` statement
  - `continue` statement
  - `goto` statement [not covered in this course]

`return` Statement (1/2)

- General format:

  `return` *expression*$_{optional}$;

`return` Statement (2/2)

- 1st purpose is to provide *return* value for a function
  - For example, last statement of `main` function returns `int` value: `return 1;`
  - To return nothing because function returns `void`, simply say: `return;`
- 2nd purpose is to unilaterally jump out of function if mission is accomplished or if it is determined that mission cannot be accomplished because of missing information

break Statement

- Typically used in 2 scenarios:
  - To skip remainder of a `switch` structure
  - To exit early from an iteration structure
- After `break` statement executes, program execution jumps to first statement after `switch` or iteration structure

break Statement: Example 1

```
int year = 1;
if (1 == year) {
  printf("Freshman\n");
} else if (2 == year) {
  printf("Sophomore\n");
} else if (3 == year) {
  printf("Junior\n");
} else if (4 == year) {
  printf("Senior\n");
} else {
  printf("Who are you?\n");
}
```

```
switch (year) {
  case 1:
    printf("Freshman\n");
  break;
  case 2:
    printf("Sophomore\n");
  break;
  case 3:
    printf("Junior\n");
  break;
  case 4:
    printf("Senior\n");
  break;
  default:
    printf("Who are you?\n");
}
```

break Statement: Flow of Control in Loop

```
for (/* expressions */) {
  // statements
  break;
  // more statements
}
// break jumps here
// even more statements
```

break Statement: Example 2 (1/2)

- Find sum of numbers entered from standard input *until* first negative number (without break)

```
#include <stdbool.h>
int sum = 0, num;
bool isNegative = false;
while (!isNegative && scanf("%d", &num)) {
  if (num < 0) {
    isNegative = true;
  } else {
    sum += num;
  }
}
printf("Sum: %d\n", sum);
```

break Statement: Example 2 (2/2)

□ Advantage of **break** statement is that it eliminates flag variables

```c
int sum = 0, num;
while (1 == scanf(" %d", &num)) {
  if (num < 0) {
    break;
  }
  sum += num;
}
printf("Sum: %d\n", sum);
```

break Statement: Example 3

- Useful for escaping from infinite loops

```c
int i = 1;
for (;;) {
  printf("%d ", i);
  i += 1;
  if (i > 10) {
    break;
  }
}
```

```c
int i = 1;
while (1) {
  printf("%d ", i);
  i += 1;
  if (i > 10) {
    break;
  }
}
```

`continue` Statement (1/3)

- Used in `while`, `do…while`, `for` structures to skip remaining statements in loop and proceed with next iteration of loop

continue Statement (2/3)

- In **while** and **do…while** structures, ***loop condition*** test is evaluated immediately after **continue** statement

```
initial statement 1
while (loop condition) {
   statement 1
   ...
   continue;
   ...
   statement n
}
```

```
do
   statement 1
   ...
   continue;
   ...
   statement n
while (loop condition);
```

`continue` Statement (3/3)

- □ In `for` statement, execution jumps to ***update expression*** after `continue` statement and then ***loop condition*** test

```
for (initial expression; loop condition; update expression)
{
   statement 1
   ...
   continue;
   ...
   statement n
}
```

continue Statement: Flow of Control in Loop

```
for (/* expressions */) {
   // statements
   continue;
   // more statements
   // continue jumps here
}
// even more statements
```

`continue` Statement: Example

- Find sum of numbers entered from standard input *excluding* negative numbers

```c
int sum = 0, num;
while (1 == scanf("%d", &num)) {
   if (num < 0) {
      continue;
   }
   sum += num;
}
printf("Sum: %d\n", sum);
```