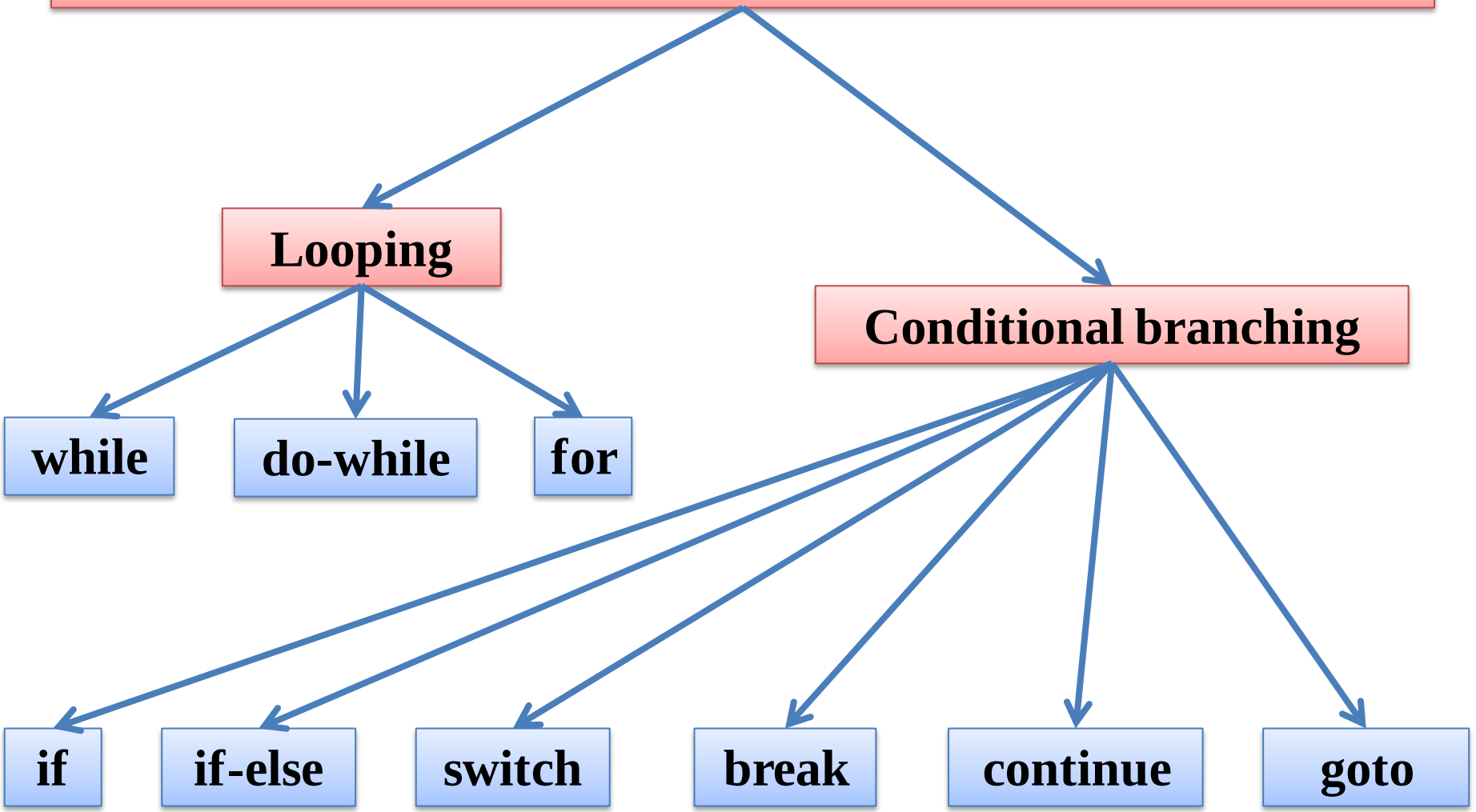
Introduction to Scientific and Engineering Computation (BIL 104E)

Lecture 6

Controlling Program Flow

Controlling Program Flow

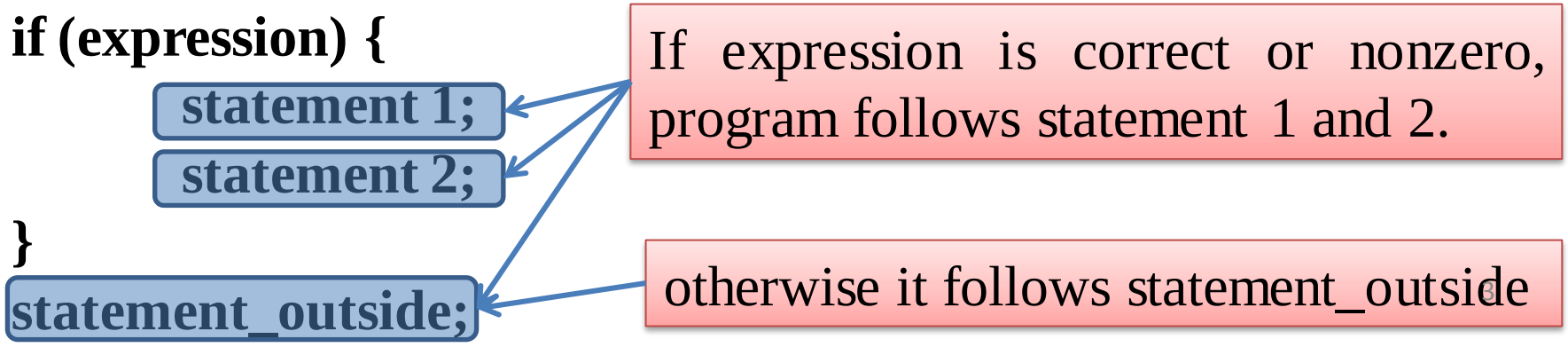


Controlling Program Flow: if

An important task of a program is to instruct the computer to **branch** (that is, jump) to different portions of the code and work on different jobs whenever the specified conditions are met.

The **if** statement is the most popular **conditional branching statement**.

It can be used to evaluate the conditions as well as to make the decision whether the block of code controlled by the statement is going to be executed.



Controlling Program Flow: if

The general form of the if statement is

```
if (expression) {
        statement1;
        statement2;
        .
        .
}
```

If there is only one statement inside the block, the braces can be omitted.

Here expression is the conditional criterion.

If expression evaluates to a nonzero value, the statements inside the braces ({ and }), such as **statement1** and **statement2**, are executed.

If expression evaluates to a value of zero, the statements are skipped.

Controlling Program Flow: if

Using the if Statement in Decision Making

```
1:  /* 10L01.c  Using the if statement */
2:  #include <stdio.h>
3:
4:  main()
5:  {
6:     int i;
7:
8:     printf("Integers that can be divided by both 2 and 3\n");
9:     printf("(within the range of 0 to 100):\n");
10:    for (i=0; i<=100; i++){
11:       if ((i%2 == 0) && (i%3 == 0))
12:          printf("   %d\n", i);
13:   }
14:   return 0;
15: }
```

Controlling Program Flow: if

Computer Screen

```
Integers that can be divided by both 2 and 3
(within the range of 0 to 100):
   0
   6
   12
   18
   24
   30
   36
   42
   48
   54
   60
   66
   72
   78
   84
   90
   96
```

Controlling Program Flow: if - else

In the **if** statement, when the conditional expression evaluates to a **nonzero value**, the computer will jump to the statements controlled by the **if** statement and execute them right away.

If the expression evaluates to a **value of zero**, the computer will ignore those statements controlled by the **if** statement.

If –else statement is used to execute an alternate set of statements when the conditional expression of the **if** statement evaluates to **logically false**.

Controlling Program Flow: if - else

The general form of the if statement is

```
if (expression) {
        statement1;
        statement2;
        .
}
else {
        statement_A;
        statement_B;
        .
}
```

If expression is correct or nonzero, program follows statements in the **if part**.

otherwise it follows statements in the **else part**.

If expression evaluates to a nonzero value, the statements controlled by if, including statement1 and statement2, are executed.

However, if expression evaluates to a value of zero, statement_A and statement_B following the else keyword are executed instead.

Controlling Program Flow: if - else

Using the if-else Statement

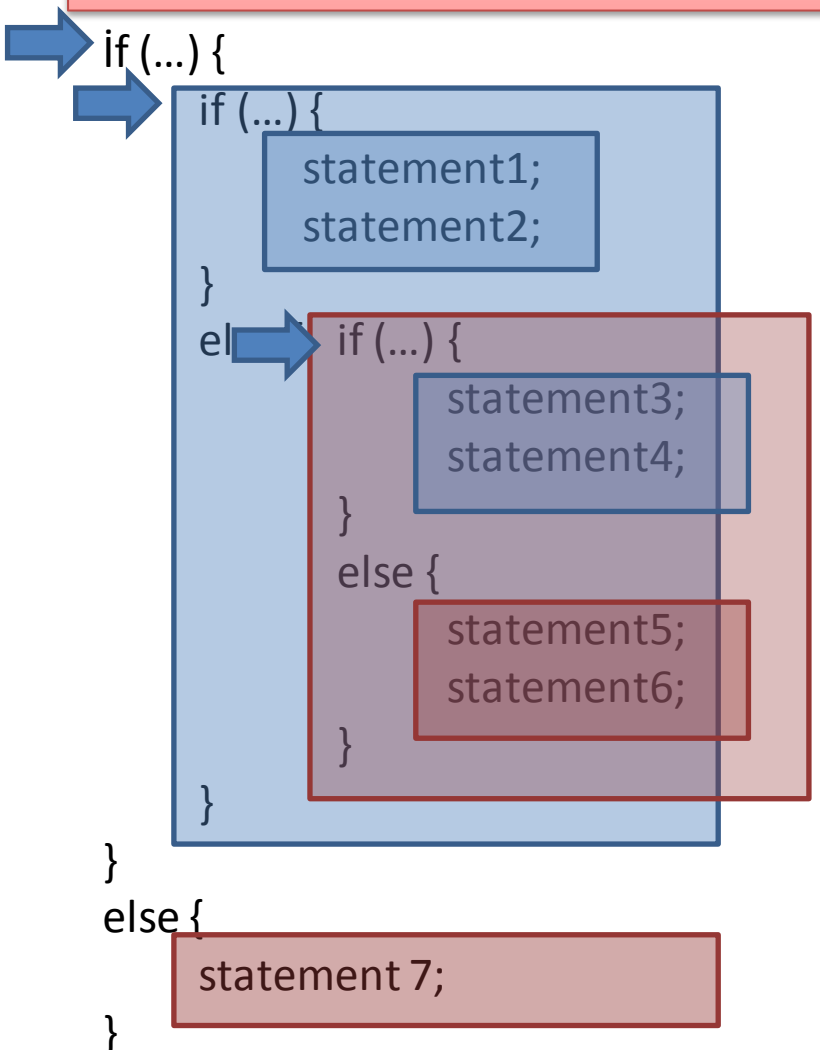
```
1:  /* 10L02.c  Using the if-else statement */
2:  #include <stdio.h>
3:
4:  main()
5:  {
6:     int i;
7:
8:     printf("Even Number   Odd Number\n");
9:     for (i=0; i<10; i++)
10:       if (i%2 == 0)
11:          printf("%d", i);
12:       else
13:          printf("%14d\n", i);
14:
15:    return 0;
16: }
```

Computer Screen

```
Even Number   Odd Number
0             1
2             3
4             5
6             7
8             9
```

Controlling Program Flow: nested if

if statement enables a program to make one decision.
In many cases, a program has to make a series of related decisions.
For this purpose, you can use **nested if** statements.



Controlling Program Flow: Nested if

Using Nested `if` Statements

```
1:  /* 10L03.c  Using nested if statements */
2:  #include <stdio.h>
3:
4:  main()
5:  {
6:     int i;
7:
8:     for (i=-5; i<=5; i++){
9:        if (i > 0)
10:           if (i%2 == 0)
11:              printf("%d is an even number.\n", i);
12:           else
13:              printf("%d is an odd number.\n", i);
14:       else if (i == 0)
15:           printf("The number is zero.\n");
16:       else
17:           printf("Negative number: %d\n", i);
18:    }19:    return 0;
20: }
```

Controlling Program Flow: Nested if

Computer Screen

```
Negative number: -5
Negative number: -4
Negative number: -3
Negative number: -2
Negative number: -1
The number is zero.
1 is an odd number.
2 is an even number.
3 is an odd number.
4 is an even number.
5 is an odd number.
```

Controlling Program Flow: switch

The **nested if** statements can become very complex if there are many decisions that need to be made.

Sometimes, a programmer will have problems just keeping track of a series of complex **nested if** statement

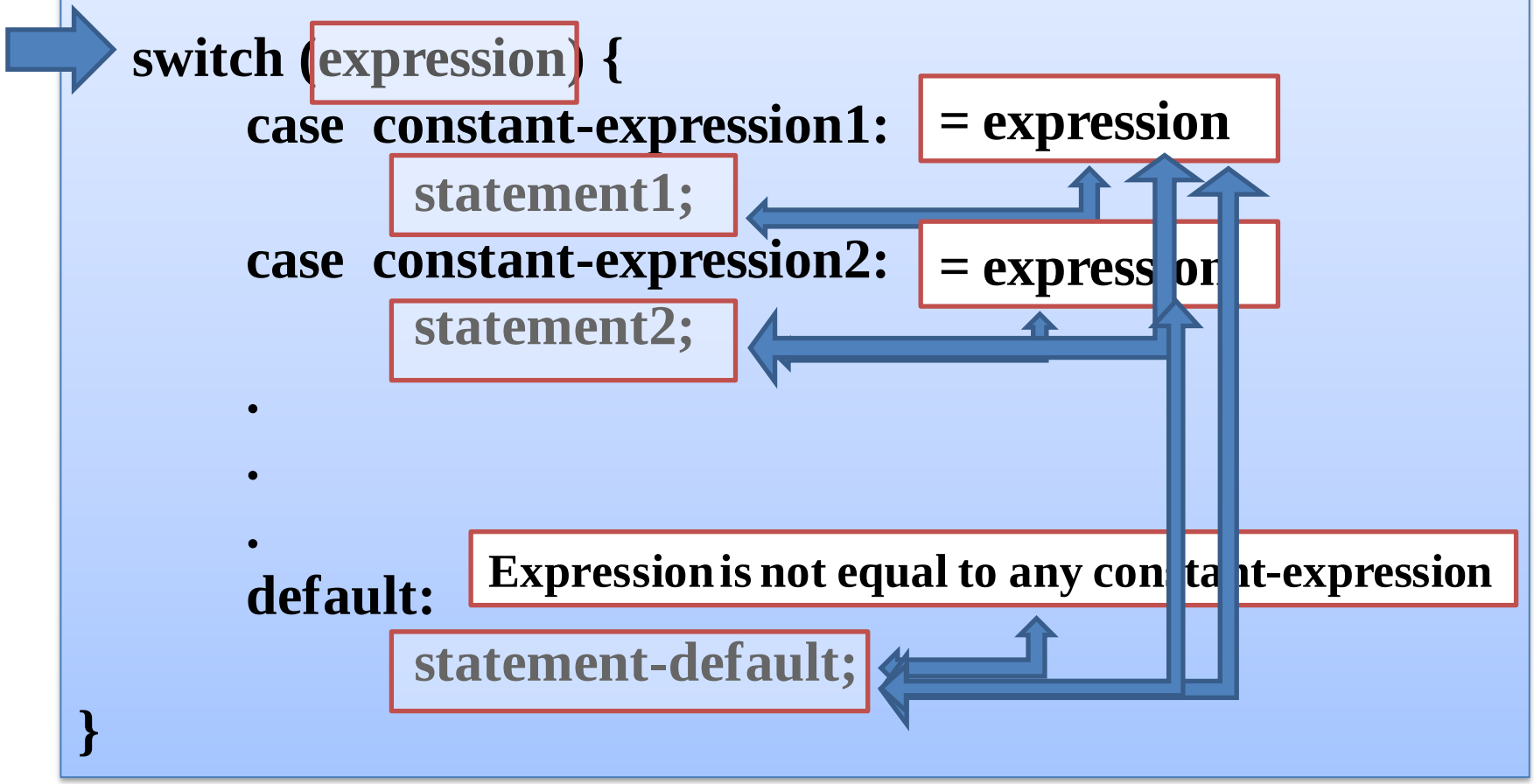
The switch statement can be used to make unlimited decisions or choices based on the value of a **conditional expression** and specified **cases**.

Label points to a place to jump when you want to depart from the normal topdown flow of execution.

Label is a unique identifier followed by a colon -not a semicolon

Controlling Program Flow: switch

The general form of the switch statement is



Controlling Program Flow: switch

Using the switch Statement

```
1:  /* 10L04.c  Using the switch statement */
2:  #include <stdio.h>
3:
4:  main()
5:  {
6:     int day;
7:
8:     printf("Please enter a single digit for a day\n");
9:     printf("(within the range of 1 to 3):\n");
10:    day = getchar();
11:    switch (day){
12:       case '1':
13:          printf("Day 1\n");
14:       case '2':
15:          printf("Day 2\n");
16:       case '3':
17:          printf("Day 3\n");
18:       default:
19:          ;
20:    }
21:    return 0;
22: }
```

```
Please enter a single digit for a day
(within the range of 1 to 3):
3
Day 3
```

```
(within the range of 1 to 3):
1
Day 1
Day 2
Day 3
```

Controlling Program Flow: break

The **break** statement can be used at the end of the statement to exit the **switch** entirely after each case **label**.

The **break** statement simply exits the **switch** and resumes execution after the end of the **switch** statement block.

The **break** statement can be used to exit an loop.

The **break** statement can be added by according to specified condition. After condition is realized, the loop is terminated and program resumes execution after the end of the loop block.

You can also use the **break** statement to break an infinite loop.

Controlling Program Flow: break

Adding the break Statement

```
1:  /* 10L05.c  Adding the break statement */
2:  #include <stdio.h>
3:
4:  main()
5:  {
6:     int day;
7:
8:     printf("Please enter a single digit for a day\n");
9:     printf("(within the range of 1 to 7):\n");
10:    day = getchar();
11:    switch (day){
12:       case '1':
13:          printf("Day 1 is Sunday.\n");
14:          break;
15:       case '2':
16:          printf("Day 2 is Monday.\n");
17:          break;
18:       case '3':
19:          printf("Day 3 is Tuesday.\n");
```

Controlling Program Flow: break

```
20:            break;
21:         case '4':
22:            printf("Day 4 is Wednesday.\n");
23:            break;
24:         case '5':
25:            printf("Day 5 is Thursday.\n");
26:            break;
27:         case '6':
28:            printf("Day 6 is Friday.\n");
29:            break;
30:         case '7':
31:            printf("Day 7 is Saturday.\n");
32:            break;
33:         default:
34:            printf("The digit is not within the range of 1 to 7.\n");
35:            break;
36:     }
37:     return 0;
38: }
```

```
Please enter a single digit for a day
(within the range of 1 to 7):
1
Day 1 is Sunday.
```

Controlling Program Flow: Breaking an Infinite Loop

```
for ( ; ; ) {
        statement1;
        statement2;
        .
        .
        .
}

while {
        statement1;
        statement2;
        .
        .
        .
}
```

If you use nothing instead of conditional expression of the **loop**, the **loop** always continues and is named **infinite loop**.

Controlling Program Flow: Breaking an Infinite Loop

Breaking an Infinite Loop

```
1:  /* 10L06.c: Breaking an infinite loop */
2:  #include <stdio.h>
3:
4:  main()
5:  {
6:     int c;
7:
8:     printf("Enter a character:\n(enter x to exit)\n");
9:     while  {
10:       c = getc(stdin);
11:       if (c == 'x')
12:          break;
13:    }
14:    printf("Break the infinite while loop. Bye!\n");
15:    return 0;
16: }
```

```
Enter a character:
(enter x to exit)
H
I
x
Break the infinite while loop. Bye!
```

Controlling Program Flow: continue

The **continue** statement causes execution to jump to the bottom of the loop immediately.

Therefore, some statements after continue statement are skipped. Iteration of this loop is stopped when program comes across with **continue** statement and new iteration is started.

Controlling Program Flow: continue

Using the continue Statement

```
1:  /* 10L07.c: Using the continue statement */
2:  #include <stdio.h>
3:
4:  main()
5:  {
6:     int i, sum;
7:
8:     sum = 0;
9:     for (i=1; i<8; i++){
10:       if ((i==3) || (i==5))
11:          continue;
12:       sum += i;
13:    }
14:    printf("The sum of 1, 2, 4, 6, and 7 is: %d\n", sum);
15:    return 0;
16: }
```

```
The sum of 1, 2, 4, 6, and 7 is: 20
```

Controlling Program Flow: goto

The following is the general form of the goto statement:

```
labelname:
        statement1;
        statement2;
        .
        .
        .
goto labelname;
```

labelname is a label name that tells the **goto** statement where to jump.

I do not recommend that you use the goto statement. Because this keyword takes program anywhere you want and reduces readibility of C programs.