CS161: Introduction to Computer Science I

Week 4

10/2022

What is for today?

fit@hcmus

Operator Precedence

- Increment and Decrement Operators

Repetition in Programs

- while, do while, for loops

Operator Precedence

Operator Precedence

Operator Precedence

Operator Precedence

	Operator	Description	Associativity
1	!	Logical negation (Highest)	R-L
2	()	Parens	L-R
3	*,/,%	Multiply, Divide, Modulo	L-R
4	+, -	Add, Subtract	L-R
5	>, >=, <, <=	Relational Operators	L-R
6	==, !=	Equality Operators	L-R
7	&&	AND	L-R
8	||	OR	L-R
9	=	Asignment (Lowest)	R-L
…			

Increment/Decrement Ops

fit@hcmus

There are two more operators that add or subtract 1

`++i`	means	`i = i + 1`
`--i`	means	`i = i - 1`

These are used in their prefix form

They can also be used in a postfix form:

`i++` `i--`

Although in books you will find the postfix form very common, get in the habit of using the prefix whenever possible!

Prefix Increment/Decrement

fit@hcmus

`++i` means:

1. Increment the variable by 1
2. Then, residual value is the current value of the variable
3. For example:

```
int i = 100;
cout << ++i;
```

→ Displays 101 not 100!

Postfix Increment/Decrement

fit@hcmus

`i++` means:

1. Residual value is the current value of the variable
2. Then, increment the variable by 1
3. For example:

```
int i = 100;
cout << i++;
```

→ Displays 100 not 101!

Increment/Decrement

fit@hcmus

More examples:

```
int i;
cin >> i;
i++;
++i;
cout << i;
```

input

50

100

output

Increment/Decrement

fit@hcmus

More examples:

```
int i, j;
cin >> i;
j = i++;
j = ++j;
cout << j;
```

input

50

100

output

Repetition in Programs

fit@hcmus

What if we wanted to give the user another chance to enter their input...

This would be impossible without **loops**

Algorithms that require loops look something like:

- Step 1: Receive Input
- Step 2: Echo the Input
- Step 3: Ask the user if this is correct
- Step 4: If not, <u>repeat</u> beginning at step #1

Three types of Loops

There are three ways to repeat a set of code using loops:

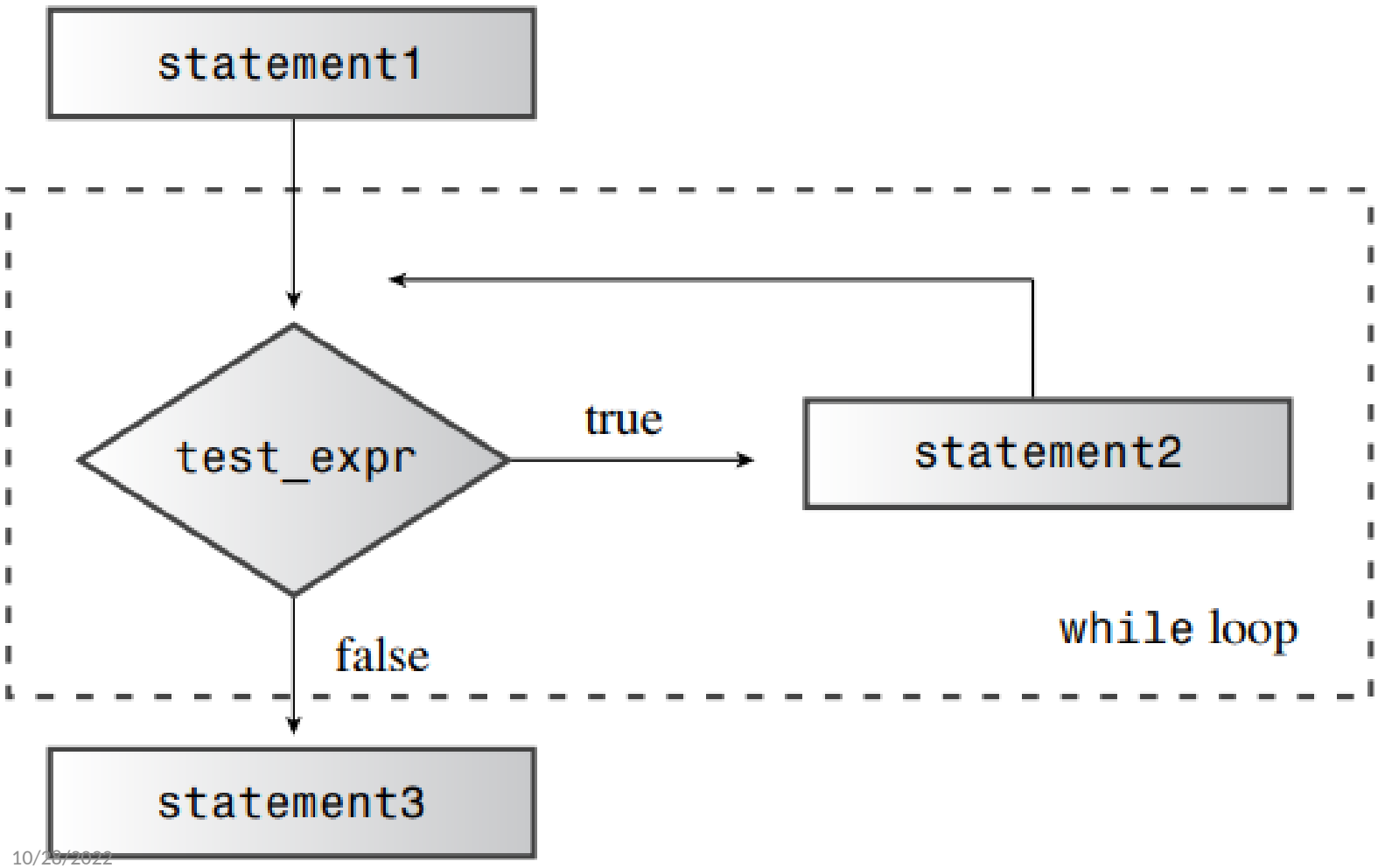
- `while` loop
- `do while` loop
- `for` loop

Each of these can perform the same operations...

- it is all in how you think about it!

....let's see....

```
statement1
while (test_expr)
        statement2
statement3
```



Using a while Loop

fit@hcmus

The `while` statement means that while an expression is true, the body of the `while` loop will be executed.

Once it is no longer true, the body will be bypassed.

The first thing that happens is that **the expression is checked, before the while loop is executed**.

THIS ORDER IS IMPORTANT TO REMEMBER!

Using a while Loop

The Syntax of the `while` loop is:

```
while (loop repetition condition)
      <body>
```

Where, the `<body>` is either one statement followed by a semicolon or a compound statement surrounded by `{}`.

```
while (logical expression)
      single statement;
```

```
while (logical expression)
{
      many statements;
}
```

Using a while Loop

fit@hcmus

Remember the body is only executed when the condition is true.

Then, after the body is executed, the condition is tested again...

- Notice, you must remember to initialize the loop control variable before you enter the while loop.
- Then, you must have some way of updating that variable inside of the body of the loop so that it can change the condition from true to false at some desired time.
- If this last step is missing, the loop will execute "forever" ... this is called an **infinite loop**.

Using a while Loop

fit@hcmus

Let's give the user a 2nd (and 3rd, 4th, 5th...) chance to enter their data using a `while` loop:

- Repeat inputting numbers until users press 'n'
- We will need a **control variable** to be used to determine when the loop is done...

Using a while Loop

fit@hcmus

/* This code confirm if the data number that was inputted by the user is correct or not, if not, ask he/she to type again */

```
int data;
char response = 'n'; //control variable
while ('n' == response) {
   cout << "Please enter your data number: ";
   cin >> data;
   cout << "We received: " << data
        << "\nIs this correct? (y/n)";
   cin >> response;
}
```

Using a while Loop

fit@hcmus

What is a drawback of the previous loop?

- The user may have entered a lower or upper case response!

One way to fix this:

- Change the logical expression to list all of the legal responses

```
while ('n' == response || 'N' == response)
{
            ...
}
```

Using a while Loop

fit@hcmus

Yet another way to fix this:

- To loop, assuming that they want to continually try again until they enter a Y or a y!
- Notice the use of AND versus OR!

```
while ('y' != response || 'Y' != response)
{
            ...
}
```

Using a while Loop

fit@hcmus

Another way to fix this:

- Use the tolower function in the ctype library:

```
#include <ctype>

while (tolower(response) != 'y')
{
      ...
}
```

Using a while Loop

fit@hcmus

Another way to fix this:

- Use the toupper function in the ctype library:

```
#include <ctype>

while (toupper(response) != 'Y')
{
      ...
}
```

Using a do while Loop

fit@hcmus

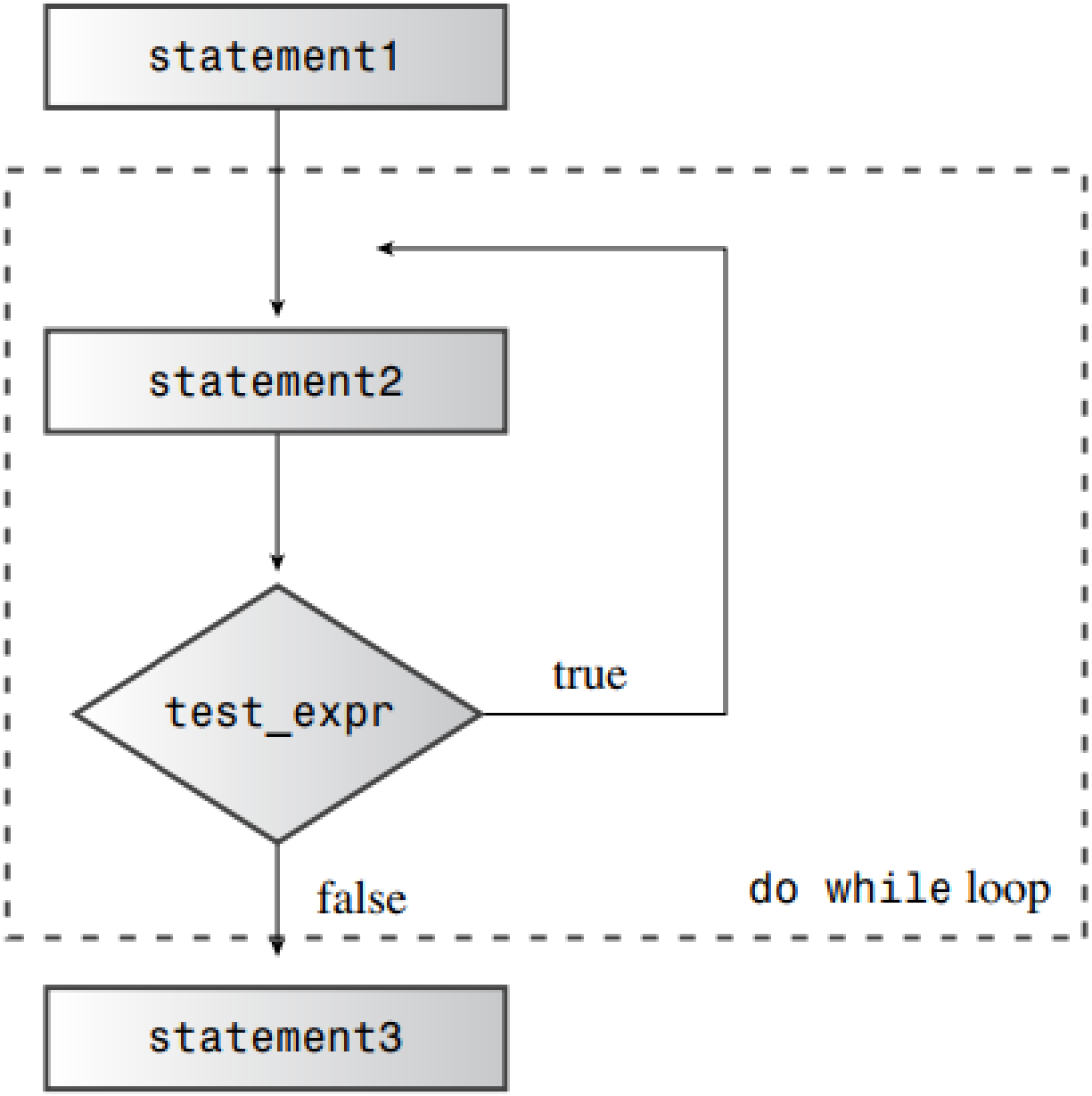
This same loop could have been rewritten using a `do while` loop instead

`do while` loops have the form: (notice semicolons!)

```
do
   single statement;
while (logical expression);
```

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

```
statement1
do
      statement2
while (test_expr);
statement3
```



Using a do while Loop

fit@hcmus

Things to notice about a do while statement:

- The body of a `do while` statement can be one statement or a compound statement surrounded by `{ }`
- Each statement in the do while loop is separated by a semicolon
- Notice the body is **always executed once**! Even if the logical expression is false the first time!

Using a do while Loop

fit@hcmus

Don't use a `do while` unless you are sure that the body of the loop should be executed at least once!

```
char response; //control variable
do {
    cout << "Please enter ... ";
    cin >> data;
    cout << "We received: " << data
         << "\nIs this correct? (y/n)";
    cin >> response;
} while ('y' != response && 'Y' != response);
```

Using a for loop

The Syntax of **for** loop is:

```
for (initialize; conditional exp; update action)
      <body>
```

Note: The body of the for loop is either one statement followed by a semicolon or a compound statement surrounded by **{ }**.

Using a for loop

fit@hcmus

The for statement will first

- INITIALIZE VARIABLE `i` to 0;
- Check the logical expression to see if it is True or False;
- if it is True the body of the loop is executed and it INCREMENTs VARIABLE `i` by 1;

  or, if it is False the loop is terminated and the statement following the body of the loop is executed.

Using a for loop

fit@hcmus

The `for` loop is commonly used to loop a certain number of times. For example, you can use it to print out the integers 0 thru 8:

```
for (int i=0; i < 9; ++i)
     cout << i << endl;
```

- `i` is called the loop control variable.
- It is most common to use variables `i, j` and `k` for control variables.
- But, mnemonic names are better!

Using a for Loop

fit@hcmus

In C++

```
for (int i=0; i < 10; ++i)
   j+=i ; //remember this is j = j+i;
```

is the same as:

```
int i = 0;
while (i < 10) {
      j +=  i;
      ++i;
}
```

Using a for Loop

fit@hcmus

We can also use a for loop to do the same loop that we have been talking about today:

```
for (char response = 'n';
     'y' != response && 'Y' != response;
     cin >> response)
{
   cout << "Please enter ... ";
   cin >> data;
   cout << "We received: " << data
        << "\nIs this correct? (y/n)";
}
```

Using a for Loop

fit@hcmus

Remember to use semicolons after each statement; however, a semicolon right after the parentheses will cause there to be a null body (i.e., nothing will be executed as long as you are inside the loop!):

```
for (int i=1; i <= 10; ++i);   //null body
     cout << "hello";     //this happens ONLY
                                 //after i is >
10.
```

Using a Loop

When using loops, check for the following conditions:

(1) Has the loop iterated one to many times? Or, one to few times?

(2) Have you properly initialized the variables used in your while or do-while logical expressions?

(3) Are you decrementing or incrementing those variables within the loop?

(4) Is there an infinite loop?

The break and continue statements

fit@hcmus

Two ways to alter the flow of control for the while, do-while, and for loops are to insert a break or continue statement.

- The **break;** statement ends the loop
- The **continue;** statement ends the current iteration of the loop body.

The break and continue statements

fit@hcmus

```
int data; //number inputted by users
int sum_even = 0; //only calculate sum of even numbers
do {
  cout << "Please enter an even number or 0 to exit: ";
  cin >> data;
  if(data %2 != 0) {
     cout << data << " is odd! Please type again! "
          <<endl;
     continue; //don't calculate if users input odd number
  }
  if(data == 0)//end loop if users input 0
     break;
  sum_even += data; //sum up if it is an even number
} while (1);
cout << "Sum of all even numbers is: "
     << sum_even << endl;
```

fit@hcmus

Any questions ???