



CS10003: **Programming & Data Structures**

**Dept. of Computer Science & Engineering
Indian Institute of Technology Kharagpur**

Autumn 2020



Iterations and Loops – contd.

An Example

```
int main() {  
    int fact, i;  
    fact = 1; i = 1;  
    while ( i<10 ) { /* run loop –break when fact >100 */  
        fact = fact * i;  
        if ( fact > 100 ) {  
            printf ("Factorial of %d above 100", i);  
            break; /* break out of the while loop */  
        }  
        ++i;  
    }  
    return 0;  
}
```

Test if a number is prime or not

```
int main() {  
    int n, i=2;  
    scanf ("%d", &n);  
    while (i < n) {  
        if (n % i == 0) {  
            printf ("%d is not a prime \n", n);  
            break;  
        }  
        ++i;  
    }  
    if (i == n) printf ("%d is a prime \n", n);  
    return 0;  
}
```

More efficient??

```
int main() {  
    int n, i = 2, flag = 0;  
    double limit;  
    scanf ("%d", &n);  
    limit = sqrt(n);  
    while (i <= limit) {  
        if (n % i == 0) {  
            printf ("%d is not a prime \n", n);  
            flag = 1; break;  
        }  
        i = i + 1;  
    }  
    if (flag == 0) printf ("%d is a prime \n", n);  
    return 0;  
}
```

continue Statement

continue

Skips the remaining statements in the body of a while, for or do/while structure

Proceeds with the next iteration of the loop

while and do/while

Loop-continuation test is evaluated immediately after the **continue** statement is executed

for structure

Increment expression is executed, then the loop-continuation test is evaluated.

expr3 is evaluated, then *expr2* is evaluated.

```
while (expr)  
      statement;  
  
do {  
      statements;  
} while (expr);
```

```
for ( expr1; expr2; expr3)  
      statement;
```

An Example with **break** and **continue**

```
int main() {  
    int fact = 1, i = 1;  
    while (1) {  
        fact = fact * i;  
        ++i;  
        if ( i <=10 )  
            continue; /*not done yet! Go to loop for next iteration*/  
        break;  
    }  
    return 0;  
}
```

The diagram illustrates the control flow of the provided C code. It starts with the declaration of variables `fact` and `i`. A blue arrow points from the opening brace of the `while` loop to its closing brace, indicating the loop's scope. Inside the loop, the expression `fact = fact * i;` is followed by an increment of `i`. Then, an `if` statement checks if `i` is less than or equal to 10. If true, a `continue` statement is reached, indicated by a blue arrow pointing to the label `/*not done yet! Go to loop for next iteration*/`. After the `continue` statement, a `break` statement exits the loop, shown by a black arrow pointing to the closing brace of the loop. Finally, the function returns 0.

Some Loop Pitfalls

```
while (sum <= NUM) :  
    sum = sum+2;
```

```
for (i=0; i<=NUM; ++i):  
    sum = sum+i;
```

```
for (i=1; i!=10; i=i+2)  
    sum = sum+i;
```

```
double x;  
for (x=0.0; x<2.0; x=x+0.2)  
    printf("%0.18f\n", x);
```

Nested Loops: Printing a 2-D Figure

How would you print the following diagram?

* * * * *

* * * * *

* * * * *

repeat 3 times

print a row of 5 *'s

repeat 5 times
print *

Nested Loops

```
const int ROWS = 3;  
const int COLS = 5;  
...  
row = 1;  
while (row <= ROWS) {  
    /*print a row of 5 '*'s*/  
    ...  
    ++row;  
}
```

```
row = 1;  
while (row <= ROWS) {  
    /* print a row of 5 '*'s */  
    col = 1;  
    while (col <= COLS) {  
        printf ("* ");  
        col++;  
    }  
    printf("\n");  
    ++row;  
}
```

outer
loop

inner
loop

2-D Figure: with **for** loop

Print

* * * * *

* * * * *

* * * * *

```
const int ROWS = 3;  
const int COLS = 5;  
....  
for (row=1; row<=ROWS; ++row) {  
    for (col=1; col<=COLS; ++col) {  
        printf("* ");  
    }  
    printf("\n");  
}
```

Another 2-D Figure

Print

```
*  
* *  
* * *  
* * * *  
* * * * *
```

```
const int ROWS = 5;  
....  
int row, col;  
for (row=1; row<=ROWS; ++row) {  
    for (col=1; col<=row; ++col) {  
        printf("* ");  
    }  
    printf("\n");  
}
```

Yet Another One

Print

* * * * *

* * * * *

* * * *

* *

*

```
const int ROWS = 5;  
....  
int row, col;  
for (row=0; row<ROWS; ++row) {  
    for (col=1; col<=row; ++col)  
        printf(" ");  
    for (col=1; col<=ROWS-row; ++col)  
        printf("* ");  
    printf ("\n");  
}
```

break and continue with nested loops

For nested loops, **break** and **continue** are matched with the nearest loops (**for**, **while**, **do-while**)

Example:

```
while (i < n) {  
    for (k=1; k < m; ++k) {  
        if (k % i == 0) break;      ←→  
    }  
    i = i + 1;      ←→ Breaks here  
}
```

Example

```
int main()
{
    int low, high, desired, i, flag = 0;
    scanf("%d %d %d", &low, &high, &desired);
    i = low;
    while (i < high) {
        for (j = i+1; j <= high; ++j) {
            if (j % i == desired) {
                flag = 1;
                break;
            }
        }
        if (flag == 1) break;
        i = i + 1;
    }
    return 0;
}
```

The diagram illustrates the control flow of the C code. It starts at the top left with the function definition. An arrow points down to the first brace of the outer while loop. Another arrow points down to the first brace of the inner for loop. From the inner for loop's brace, an arrow points right to the 'break;' statement inside the if block. A red annotation 'Breaks here' is placed to the right of this 'break;' statement. An arrow then points back up to the outer while loop's brace. From there, another arrow points right to the 'break;' statement at the end of the if block. A second red annotation 'Breaks here' is placed to the right of this 'break;' statement. Finally, an arrow points down to the 'return 0;' statement at the bottom.

The comma operator

- Separates expressions
- Syntax

expr-1, expr-2, ...,expr-n

where, **expr-1, expr-2,...** are all expressions

- Is itself an expression, which evaluates to the value of the last expression in the sequence
- Since all but last expression values are discarded, not of much general use
- But useful in for loops, by using side effects of the expressions

Example

- We can give several expressions separated by commas in place of **expr1** and **expr3** in a for loop to do multiple assignments for example

```
for (fact=1, i=1; i<=10; ++ i)  
    fact = fact * i;
```

```
for (sum=0, i=1; i<=N; ++i)  
    sum = sum + i * i;
```

Homework

Compute the following function given a value of x
with the accuracy of 10^{-6} :

$$f(x) = 1 - x^2/2! + x^4/4! - x^6/6! + \dots$$

You should not use any math library functions
or C function to calculate factorial.

Computing standard deviation

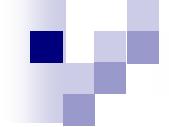
The Steps

1. Read N
2. Read X_i
3. Compute Mean
4. Compute Standard Deviation

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2}$$

$$\mu = \frac{1}{N} \sum_{i=1}^N x_i$$

The Problem



Thank You!