

Different Loop constructs - and related ...

- **while, for, do while.**
- Why have different constructs?
- Can you replace one by another?
Yes, with careful modifications
- Each of the construct is natural for different problems.
- `break`, `continue` to handle control flow within the runs of the loop.

Computing 2^n

Accept n from the user. Print 2^n on the standard output.

```
#include<stdio.h>

int main() {
    int n;
    int value = 2;

    printf("Enter a positive integer:\t");
    scanf("%d", &n);

    while (____) {
        value *= 2;
        -----;
    }
    printf("%d\n", value);
    return 0;
}
```

Computing 2^n

Accept n from the user. Print 2^n on the standard output.

Computing 2^n

Accept n from the user. Print 2^n on the standard output.

```
#include<stdio.h>

int main() {
    int n;
    int value = 2;

    printf("Enter a positive integer:\t");
    scanf("%d", &n);

    while (n > 1) {
        value *= 2;
        n--;
    }
    printf("%d\n", value);
    return 0;
}
```

Stepping back : Stages of Program Design ...

To solve a problem using a computer program :

Stepping back : Stages of Program Design ...

To solve a problem using a computer program :

- Need to develop an idea of the sequence of statements and the flow of the control through the statements to achieve a given task.

Stepping back : Stages of Program Design ...

To solve a problem using a computer program :

- Need to develop an idea of the sequence of statements and the flow of the control through the statements to achieve a given task.
- **Algorithm** : The description of the idea - *a step-by-step procedure to solve the problem.*

Stepping back : Stages of Program Design ...

To solve a problem using a computer program :

- Need to develop an idea of the sequence of statements and the flow of the control through the statements to achieve a given task.
- **Algorithm** : The description of the idea - *a step-by-step procedure to solve the problem*.
- **Pseudo-code** : A language-independent but program-like description of an algorithm.

Stepping back : Stages of Program Design ...

To solve a problem using a computer program :

- Need to develop an idea of the sequence of statements and the flow of the control through the statements to achieve a given task.
- **Algorithm** : The description of the idea - *a step-by-step procedure to solve the problem.*
- **Pseudo-code** : A language-independent but program-like description of an algorithm.
- **Flowchart** : A diagrammatic representation of the algorithm is called a *flowchart*.

An Illustration : Collatz Sequence

An Illustration : Collatz Sequence

Algorithm : To get a Collatz sequence from a number, if it's even, divide it by two, and if it's odd, multiply it by three and add one. Continue the operation on the result of the previous operation until the number becomes 1.

An Illustration : Collatz Sequence

Algorithm : To get a Collatz sequence from a number, if it's even, divide it by two, and if it's odd, multiply it by three and add one. Continue the operation on the result of the previous operation until the number becomes 1.

Pseudo-code:

```
begin program
  while n is more than 1
    show n
    if n is odd then
      set  $n = 3n + 1$ 
    else
      set  $n = n / 2$ 
    endif
  endwhile
  show n
end program
```

An Illustration : Collatz Sequence

Algorithm : To get a Collatz sequence from a number, if it's even, divide it by two, and if it's odd, multiply it by three and add one. Continue the operation on the result of the previous operation until the number becomes 1.

Pseudo-code:

```
begin program
  while n is more than 1
    show n
    if n is odd then
      set  $n = 3n + 1$ 
    else
      set  $n = n / 2$ 
    endif
  endwhile
  show n
end program
```

C-program segment:

```
while(n > 1) {
  printf("%d.",n);
  if (n%2 == 1)
    n = 3*n+1;
  else
    n = n/2;
}
printf("%d.",n);
```

Programs : Termination and Correctness

Someone claims that For every n , $\sum_{k=1}^n k = \frac{n(n+1)}{2}$.

Programs : Termination and Correctness

Someone claims that For every n , $\sum_{k=1}^n k = \frac{n(n+1)}{2}$. You ask for proof !

Programs : Termination and Correctness

Someone writes a program and claims that it solves the problem correctly for all inputs.

Programs : Termination and Correctness

Someone writes a program and claims that it solves the problem correctly for all inputs. **You ask for proof !**

Programs : Termination and Correctness

Someone writes a program and claims that it solves the problem correctly for all inputs. **You ask for proof !**

- "Hello World" program is easy. Hand simulation works.
Infeasible when program receives inputs (many possibilities).

Programs : Termination and Correctness

Someone writes a program and claims that it solves the problem correctly for all inputs. **You ask for proof !**

- "Hello World" program is easy. Hand simulation works.
Infeasible when program receives inputs (many possibilities).
- An interesting complication : even for a given input, a loop could run for infinite times (hence not hand-simulatable).

Programs : Termination and Correctness

Someone writes a program and claims that it solves the problem correctly for all inputs. **You ask for proof !**

- "Hello World" program is easy. Hand simulation works.
Infeasible when program receives inputs (many possibilities).
- An interesting complication : even for a given input, a loop could run for infinite times (hence not hand-simulatable).
- There has to be mathematical statements which proves:

Programs : Termination and Correctness

Someone writes a program and claims that it solves the problem correctly for all inputs. **You ask for proof !**

- "Hello World" program is easy. Hand simulation works.
Infeasible when program receives inputs (many possibilities).
- An interesting complication : even for a given input, a loop could run for infinite times (hence not hand-simulatable).
- There has to be mathematical statements which proves:
 - **Termination:** The loop (hence, the program) terminates.

Programs : Termination and Correctness

Someone writes a program and claims that it solves the problem correctly for all inputs. **You ask for proof !**

- "Hello World" program is easy. Hand simulation works.
Infeasible when program receives inputs (many possibilities).
- An interesting complication : even for a given input, a loop could run for infinite times (hence not hand-simulatable).
- There has to be mathematical statements which proves:
 - **Termination:** The loop (hence, the program) terminates.
(Idea : Some measure decreases as loop executes every time, and hence finally, it should terminate).

Programs : Termination and Correctness

Someone writes a program and claims that it solves the problem correctly for all inputs. **You ask for proof !**

- "Hello World" program is easy. Hand simulation works.
Infeasible when program receives inputs (many possibilities).
- An interesting complication : even for a given input, a loop could run for infinite times (hence not hand-simulatable).
- There has to be mathematical statements which proves:
 - **Termination:** The loop (hence, the program) terminates.
(Idea : Some measure decreases as loop executes every time, and hence finally, it should terminate).
 - **Correctness:** The program is correctly producing the expected output.

Programs : Termination and Correctness

Someone writes a program and claims that it solves the problem correctly for all inputs. **You ask for proof !**

- "Hello World" program is easy. Hand simulation works. Infeasible when program receives inputs (many possibilities).
- An interesting complication : even for a given input, a loop could run for infinite times (hence not hand-simulatable).
- There has to be mathematical statements which proves:
 - **Termination:** The loop (hence, the program) terminates. (Idea : Some measure decreases as loop executes every time, and hence finally, it should terminate).
 - **Correctness:** The program is correctly producing the expected output. (Idea : After every iteration of the loop, the partial result that it computes implies the final result).

Programs : Termination and Correctness

Someone writes a program and claims that it solves the problem correctly for all inputs. **You ask for proof !**

- "Hello World" program is easy. Hand simulation works. Infeasible when program receives inputs (many possibilities).
- An interesting complication : even for a given input, a loop could run for infinite times (hence not hand-simulatable).
- There has to be mathematical statements which proves:
 - **Termination:** The loop (hence, the program) terminates. (Idea : Some measure decreases as loop executes every time, and hence finally, it should terminate).
 - **Correctness:** The program is correctly producing the expected output. (Idea : After every iteration of the loop, the partial result that it computes implies the final result).
- Algorithm designer's job. Not so much programmer's.

More Practice Problems

- Write a program to check if a given number n is prime or not.

More Practice Problems

- Write a program to check if a given number n is prime or not.

Algorithm: Check, for every number m in the range 2 to $n - 1$, whether m divides n or not. If none divides, then you can declare that it is a prime number. If one of them divides, then you can declare right away that it is a composite number.

More Practice Problems

- Write a program to check if a given number n is prime or not.

Algorithm: Check, for every number m in the range 2 to $n - 1$, whether m divides n or not. If none divides, then you can declare that it is a prime number. If one of them divides, then you can declare right away that it is a composite number.

A typical example where for construct is natural.

More Practice Problems

- Write a program to check if a given number n is prime or not.

Algorithm: Check, for every number m in the range 2 to $n - 1$, whether m divides n or not. If none divides, then you can declare that it is a prime number. If one of them divides, then you can declare right away that it is a composite number.

A typical example where for construct is natural.

Notice the use case of break statement within the loop.

More Practice Problems

- Write a program to check if a given number n is prime or not.

Algorithm: Check, for every number m in the range 2 to $n - 1$, whether m divides n or not. If none divides, then you can declare that it is a prime number. If one of them divides, then you can declare right away that it is a composite number.

A typical example where for construct is natural.

Notice the use case of break statement within the loop.

- Write a program to find the greatest common divisor (gcd) of two given integers.

More Practice Problems

- Write a program to check if a given number n is prime or not.

Algorithm: Check, for every number m in the range 2 to $n - 1$, whether m divides n or not. If none divides, then you can declare that it is a prime number. If one of them divides, then you can declare right away that it is a composite number.

A typical example where for construct is natural.

Notice the use case of break statement within the loop.

- Write a program to find the greatest common divisor (gcd) of two given integers. (Hint : Use the fact that for every a, b ,
if $a \% b = 0$ then $\text{gcd}(a, b) = b$.
otherwise $\text{gcd}(a, b) = \text{gcd}(b, a \% b)$).

A typical example where while construct is natural.