Control Structures:
Examples

for-loop example
- Q: If a=1, b=3, and x=7, what is the value of x when the loop terminates?
- A: 
  ```
  for(k=a; k<=b; k++)
  {
    x=x-k;
  }
  
  First iteration (k=1, x=7)
  - test: 1<=3 is true
  - execute: x=7-1=6
  - update: k=2
  
  Second iteration (k=2, x=6)
  - test: 2<=3 is true
  - execute: x=6-2=4
  - update: k=3
  
  Third iteration (k=3, x=4)
  - test: 3<=3 is true
  - execute: x=4-3=1
  - update: k=4
  
  Fourth iteration (k=4, x=1)
  - test: 4<=3 is false
  - exit loop
  ```

Another for-loop example
- Q: If a=2, b=4, x=3, and y=9, what are the values of x and y when the loop terminates?
- A:
  ```
  for(k=a; k<b; k++)
  {
    x=x+k;
    y=y-x;
  }
  
  First iteration (k=2, x=3, y=9)
  - test: 2<4 is true
  - execute: x=3+2=5, y=9-3=6
  - update: k=3
  
  Second iteration (k=3, x=5, y=6)
  - test: 3<4 is true
  - execute: x=5+3=8, y=6-6=0
  - update: k=4
  
  Third iteration (k=4, x=8, y=0)
  - test: 4<3 is false
  - exit loop
  ```

while-loop example
- Q: What is the value of p when the loop terminates?
- A:
  ```
  p=0;
  t=1;
  n=10;
  while(n>t)
  {
    p=p+n*t;
    t=t+4;
    n=n-3;
  }
  
  First iteration (n=10, p=0)
  - test: 10>1 is true
  - execute: p=0+10*1=10
  - update: t=1+4=5, n=10-3=7
  
  Second iteration (n=7, p=10)
  - test: 7>1 is true
  - execute: p=10+7*5=60
  - update: t=5+4=9, n=7-3=4
  
  Third iteration (n=4, p=60)
  - test: 4>1 is true
  - execute: p=60+4*9=84
  - update: t=9+4=13, n=4-3=1
  
  Fourth iteration (n=1, p=84)
  - test: 1>0 is false
  - exit loop
  ```

Another while-loop example
- Q: What are the values of z, p, and n after executing the following statements?
- A:
  ```
  p=1;
  z=0;
  while(p<=10)
  {
    z=n*z*p;
    p=p+4;
    n=n-3;
  }
  
  First iteration (p=1, z=0, n=10)
  - test: 1<=10 is true
  - execute: z=1*0*1=0, p=1+4=5, n=10-3=7
  
  Second iteration (p=5, z=0, n=7)
  - test: 5<=10 is true
  - execute: z=7*0*5=0, p=5+4=9, n=7-3=4
  
  Third iteration (p=9, z=0, n=4)
  - test: 9<=10 is true
  - execute: z=4*9*0=0, p=9+4=13, n=4-3=1
  
  Fourth iteration (p=13, z=0, n=1)
  - test: 13>0 is false
  - exit loop
  ```

Algorithm Design:
Examples
Minimum of two integers

Problem: Find the minimum of two integers

- Analyze the problem
  - Inputs
    - \( x \) first integer
    - \( y \) second integer
  - Output
    - \( \text{min} \) minimum of \( x \) and \( y \)
- How do we find the minimum??
  - if the first number is smaller than the second number, then the first number is the minimum
  - else, the second number is the minimum

- Design an algorithm to solve the problem
  1. Get input values for \( x \) and \( y \)
  2. Compute minimum value
     
     \[
     \text{if}(x < y) \\
     \text{min} = x; \\
     \text{else} \\
     \text{min} = y;
     \]
  3. Return output value \( \text{min} \)

Sum of positive integers

Problem: Find the sum of all positive integers less than or equal to some positive integer \( n \)

- Analyze the problem
  - Input
    - \( n \) a positive integer
  - Output
    - \( \text{sum} \) sum of all positive integers \( \leq n \)
  - How to find the sum??
    - \( \text{sum} = 1 + 2 + 3 + \ldots + n \)
    - initialize \( \text{sum} = 0 \)
    - let \( k \) loop over the values \( [1, n] \)
    - compute \( \text{sum} = \text{sum} + k \) at each iteration of loop

- Design an algorithm to solve the problem
  1. Get input value for \( n \)
  2. Compute sum of integers 1 through \( n \)
     
     \[
     \text{sum}=0; \\
     \text{for}(k=1; k<=n; k++) \\
     \{
     \text{sum} = \text{sum} + k;
     \}
     \]
  3. Return output value \( \text{sum} \)
Factorial

Problem: Given some positive integer \( n \), compute the factorial of \( n \)

Definition: factorial

- The factorial of a positive integer \( n \), denoted \( n! \), is the product of the positive integers less than or equal to \( n \).
- For example:
  - \( 1! = 1 \)
  - \( 2! = 2 \times 1 = 2 \)
  - \( 3! = 3 \times 2 \times 1 = 6 \)
  - \( 4! = 4 \times 3 \times 2 \times 1 = 24 \)
  - \( 5! = 5 \times 4 \times 3 \times 2 \times 1 = 120 \)
- We define the factorial of 0 to be 1:
  - \( 0! = 1 \)

Factorial

- Analyze the problem
  - Input
    - \( n \) a positive integer
  - Output
    - \( fn \), the factorial of \( n \)
  - How to find the factorial??
    - \( fn = 1 \times 2 \times 3 \times \ldots \times n \)
    - initialize \( fn=1 \)
    - let \( k \) loop over the values \([2, n]\)
    - compute \( fn=fn \times k \) at each iteration of loop

Factorial

- Design an algorithm to solve the problem
  1. Get input value for \( n \)
  2. Compute product of integers 2 through \( n \)
  3. Return output value \( fn \)

  ```
  fn=1;
  for (k=2; k<=n; k++)
  {
    fn=fn*k;
  }
  ```

Practice problems

1. Design an algorithm to compute the inclusive sum between two integers
   - example: for the input values 2 and 6, your algorithm should output 20 (because 2 + 3 + 4 + 5 + 6 = 20)

2. Design an algorithm that computes \( x^n \) for an integer \( x \) and a non-negative integer \( n \)
   - \( x^n \) is defined as follows:
     \[
     x^0 = 1 \quad \text{if } n=0, \text{otherwise}
     \]
     \[
     x^n = x \times x \times x \times x \times \ldots \times x \quad \text{n times}
     \]

Practice problems

3. Design an algorithm to compute the maximum of two integers

4. Design an algorithm to compute the inclusive product between two integers
   - example: for the input values 3 and 6, your algorithm should output 360 (because \( 3 \times 4 \times 5 \times 6 = 360 \))
Subroutines

- Set of instructions to perform a particular computation
  - subproblems of more complex problems
  - repeated computation (e.g. different inputs)
  - may be used in solving other problems
- Also called subprograms, methods, functions, procedures
- Subroutines are named
- Subroutines have zero or more parameters
  - list (possibly empty) of input values and their types
- Subroutines have return types

A subroutine to add two integers

```c
int add(int x, int y)
{
    return x+y;
}
```

Minimum of two integers - algorithm

- Problem: Find the minimum of two integers
- Algorithm:
  1. Get input values for \(x\) and \(y\)
  2. Compute minimum value
     ```
     if(x < y)
        min = x;
     else
        min = y;
     ```
  3. Return output value \(min\)

Minimum of two integers - subroutine

```c
int minimum(int x, int y)
{
    int min;
    if(x < y)
        min=x;
    else
        min=y;
    return min;
}
```
Minimum of two integers v.2

```c
int minimum(int x, int y) {
    if(x < y)
        return x;
    else
        return y;
}
```

Sum of positive integers - algorithm

- Problem: Find the sum of all positive integers less than or equal to some positive integer \( n \)
- Algorithm:
  1. Get input value for \( n \)
  2. Compute sum of integers 1 through \( n \)
  3. Return output value \( \text{sum} \)

```c
int sum_integers(int n) {
    int k;
    int sum=0;
    for(k=1; k<=n; k++)
    {
        sum=sum+k;
    }
    return sum;
}
```

Sum of positive integers - subroutine

Factorial - algorithm

- Problem: Given some positive integer \( n \), compute \( n! \)
- Algorithm:
  1. Get input value for \( n \)
  2. Compute product of integers 2 through \( n \)
  3. Return output value \( \text{fn} \)

```c
int factorial(int n) {
    int k;
    int fn=1;
    for(k=2; k<=n; k++)
    {
        fn=fn*k;
    }
    return fn;
}
```

Factorial - subroutine

Practice problems

- Write subroutines to perform the following computations:
  1. Compute the inclusive sum between two integers
  2. Compute \( x^n \) for an integer \( x \) and a non-negative integer \( n \)
  3. Compute the maximum of two integers
  4. Compute the inclusive product between two integers