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?

```
for(k=a; k<=b; k++)
{
  x=x-k;
}
```

- A: x=1

- 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=1**, and **y=9**, what are the values of **x** and **y** when the loop terminates?

```
for(k=a; k<b; k++)
{
  x=x+k;
  y=y-x;
}
```

- A: x=6, y=0

- First iteration (k=2, x=1, y=9)
  - test: **2<4** is **true**
  - execute: **x=1+2=3**
    y=9-3=6
  - update: **k=3**
- Second iteration (k=3, x=3, y=6)
  - test: **3<4** is **true**
  - execute: **x=3+3=6**
    y=6-6=0
  - update: **k=4**
- Third iteration (k=4, x=6, y=0)
  - test: **4<3** is **false**
  - exit loop

while-loop example

- Q: What is the value of **p** when the loop terminates?

```
p=0;
t=1;
n=10;
while(n>t)
{
  p=p+n*t;
  t=t+4;
}
```

- A: p=150

- First iteration (t=1, p=0)
  - test: **10>1** is **true**
  - execute: **p=0+10*1=10**
    t=1+4=5
- Second iteration (t=5, p=10)
  - test: **10>5** is **true**
  - execute: **p=10+10*5=60**
    t= 5+4=9
- Third iteration (t=9, p=60)
  - test: **10>9** is **true**
  - execute: **p=60+10*9=150**
    t= 9+4=13
- Fourth iteration (t=13, p=150)
  - test: **10>13** is **false**
  - exit loop

Another while-loop example

- Q: What are the values of **z**, **p**, and **n** after executing the following statements?

```
n=-1;
z= 0;
p= 1;
while(p<=10)
{
  z=n*z*p;
  p=p+4;
  n=n-3;
}
```

- A: z=0, p=13, n=-10

- First iteration (n=-1, z=0, p=1)
  - test: **1<=10** is **true**
  - execute: **z=-1*0*1=0**
    p= 1+4=5
    n=-1-3=-4
- Second iteration (n=-4, z=0, p=5)
  - test: **5<=10** is **true**
  - execute: **z=-4*0*5=0**
    p= 5+4=9
    n=-4-3=-7
- Third iteration (n=-7, z=0, p=9)
  - test: **9<=10** is **true**
  - execute: **z=-7*0*9=0**
    p= 9+4=13
    n=-7-3=-10
- Fourth iteration: exit loop

Algorithm Design:

Examples

Minimum of two integers

Problem: Find the minimum of two integers

Minimum of two integers

- Analyze the problem
  - Inputs
    - **x** first integer
    - **y** second integer
  - Output
    - **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

Minimum of two integers

- Design an algorithm to solve the problem

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**

Sum of positive integers

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

Sum of positive integers

- Analyze the problem
  - Input
    - **n** a positive integer
  - Output
    - **sum** sum of all positive integers **<=n**
  - How to find the sum??
    - **sum = 1+2+3+ … +n**
    - initialize **sum=0**
    - let **k** loop over the values [**1**, **n**]
    - compute **sum=sum+k** at each iteration of loop

Sum of positive integers

- Design an algorithm to solve the problem

1. Get input value for **n**
2. Compute sum of integers **1** through **n**

```
sum=0;

for(k=1; k<=n; k++)
{
   sum=sum+k;
}
```

3. Return output value **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*1 = 2**
  - **3! = 3*2*1 = 6**
  - **4! = 4*3*2*1 = 24**
  - **5! = 5*4*3*2*1 = 120**
- We define the factorial of **0** to be **1**
  - **0! = 1**

Factorial

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

➥ why don't we set **fn=0**?

Factorial

- Design an algorithm to solve the problem
  1. Get input value for **n**
  2. Compute product of integers **2** through **n**

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

➥ what happens when **n=1**?

  3. Return output value **fn**

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:

$x^0 = 1$ if **n=0**, otherwise

$x^n = \underbrace{x*x*x*x*x* \cdots *x}_{n \text{ times}}$

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*4*5*6 = 360)

Subroutines

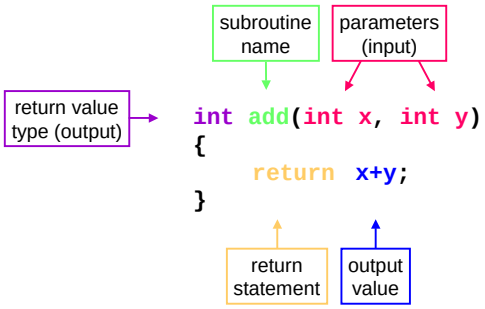
Subroutines

- 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

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

A subroutine to add two integers



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

```
int minimum(int x, int y)
{
    int min;

    if(x < y)
        min=x;
    else
        min=y;

    return min;
}
```

variables used inside the subroutine must be declared

variables can not have the same name as the subroutine

Minimum of two integers v.2

```
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**

```
sum=0;
for(k=1; k<=n; k++)
{
   sum=sum+k;
}
```

  3. Return output value **sum**

Sum of positive integers - subroutine

```
int sum_integers(int n)
{
    int k;
    int sum=0;

    for(k=1; k<=n; k++)
    {
        sum=sum+k;
    }
    return sum;
}
```

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**

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

  3. Return output value **fn**

Factorial - subroutine

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

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