Fundamentals of Computer Programming
Lecture 4: Loop Control ↺

Lecturer: Ahmad Siavashi

Spring 2023

Outline

An opening example

- Calculate $S = \sum_{x=1}^{5} \frac{1}{x^2}$
- Based on what we learn, we do it in following way

```c
1  int main()
2  {
3      int x = 1;
4      double S = 0;
5      S += 1.0/(x*x);
6      x += 1;
7      S += 1.0/(x*x);
8      x += 1;
9      S += 1.0/(x*x);
10     x += 1;
11     S += 1.0/(x*x);
12     x += 1;
13     S += 1.0/(x*x);
14     printf("S_=_%lf\n", S);
15     return 0;
16 }
```

- It is okay when the number of terms is small
- How about 1000 terms ...
- Share the story

Motivation of loops

- In the above example
- Following statement repeated for 5 times, only x changes each time

```
1    x += 1;
2    S += 1.0/(x*x);
```

- We can put it inside a loop
- Tell the loop that how many times we want to repeat

```
1  x = 0;
2  while(x <= 4)
3  {
4      x += 1;
5      S += 1.0/(x*x);
6  }
```

Loops

- To repeat statements as long as a certain condition is true (non-zero)
- C offers 3 different loops
- We can replace one with another
- Different loop offers different convenience

```
while ( condition )
    statement ;
```

```
do
    statement ;
while ( condition ) ;
```

```
for ( initialization ; condition ; statement )
    statement ;
```
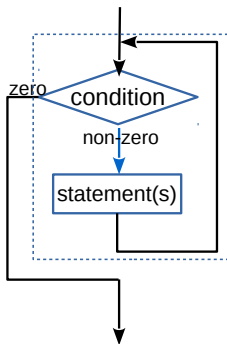
For multiple statements again, use braces.

Outline

while loop control (1)

- The execution checks if the condition is still non-zero
- If it is, execute the statement(s)
- Otherwise, gets out from the loop

while loop control (2)

- The execution checks if the condition is still non-zero
- If it is, execute the statement(s)
- Otherwise, gets out from the loop

```
1  int i = 2;
2  while (i > 0)
3      --i;
4  printf("done\n");
```

1. Check $(i > 0) \rightarrow$ **true** $\rightarrow$ go to line 3
2. Decrement $i \rightarrow i$ now is **1**, go back to line 2
3. Check $(i > 0) \rightarrow$ **true** $\rightarrow$ go to line 3
4. Decrement $i \rightarrow i$ now is **0**, go back to line 2
5. Check $(i > 0) \rightarrow$ **false** $\rightarrow$ go to line 4
6. Print **done**

- The execution checks if the condition is still non-zero
- If it is, execute the statement(s)
- Otherwise, gets out from the loop

```
1  int  i = 2;
2  while  ( i > 0 )
3      −−i ;
4  printf ( "done\n" ) ;
```

1. Check $(i > 0) \rightarrow$ **true** $\rightarrow$ go to line 3

2. Decrement i $\rightarrow$ i now is **1**, go back to line 2

3. Check $(i > 0) \rightarrow$ **true** $\rightarrow$ go to line 3

4. Decrement i $\rightarrow$ i now is **0**, go back to line 2

5. Check $(i > 0) \rightarrow$ **false** $\rightarrow$ go to line 4

6. Print **done**

- The execution checks if the condition is still non-zero
- If it is, execute the statement(s)
- Otherwise, gets out from the loop

```
int i = 2;
while (i > 0)
    --i;
printf("done\n");
```

❶ Check (i > 0) → **true** → go to line 3

❷ Decrement i → i now is **1**, go back to line 2

❸ Check (i > 0) → **true** → go to line 3

❹ Decrement i → i now is **0**, go back to line 2

❺ Check (i > 0) → **false** → go to line 4

❻ Print **done**

while loop control (2)

- The execution checks if the condition is still non-zero
- If it is, execute the statement(s)
- Otherwise, gets out from the loop

```
1  int i = 2;
2  while (i > 0)
3     --i;
4  printf("done\n");
```

1 Check (i > 0) → **true** → go to line 3

2 Decrement i → i now is **1**, go back to line 2

3 Check (i > 0) → **true** → go to line 3

4 Decrement i → i now is **0**, go back to line 2

5 Check (i > 0) → **false** → go to line 4

6 Print **done**

while loop control (2)

- The execution checks if the condition is still non-zero
- If it is, execute the statement(s)
- Otherwise, gets out from the loop

```c
int i = 2;
while (i > 0)
    --i;
printf("done\n");
```

❶ Check $(i > 0) \rightarrow$ **true** $\rightarrow$ go to line 3

❷ Decrement $i \rightarrow i$ now is **1**, go back to line 2

❸ Check $(i > 0) \rightarrow$ **true** $\rightarrow$ go to line 3

❹ Decrement $i \rightarrow i$ now is **0**, go back to line 2

❺ Check $(i > 0) \rightarrow$ **false** $\rightarrow$ go to line 4

❻ Print **done**

while loop control (2)

- The execution checks if the condition is still non-zero
- If it is, execute the statement(s)
- Otherwise, gets out from the loop

```
1  int  i = 2;
2  while  (i > 0)
3      --i;
4  printf("done\n");
```

① Check $(i > 0) \rightarrow$ **true** $\rightarrow$ go to line 3

② Decrement $i \rightarrow i$ now is **1**, go back to line 2

③ Check $(i > 0) \rightarrow$ **true** $\rightarrow$ go to line 3

④ Decrement $i \rightarrow i$ now is **0**, go back to line 2

⑤ Check $(i > 0) \rightarrow$ **false** $\rightarrow$ go to line 4

⑥ Print **done**

while loop control (2)

- The execution checks if the condition is still non-zero
- If it is, execute the statement(s)
- Otherwise, gets out from the loop

```c
int i = 2;
while (i > 0)
    --i;
printf("done\n");
```

❶ Check (i > 0) → **true** → go to line 3

❷ Decrement i → i now is **1**, go back to line 2

❸ Check (i > 0) → **true** → go to line 3

❹ Decrement i → i now is **0**, go back to line 2

❺ Check (i > 0) → **false** → go to line 4

❻ Print **done**

Check Prime Number (1)

- Check whether an integer number is a prime number
- Prime number: number that is only dividable by 1 and itself
- 81 is not a prime number; 173 is prime number
- Any idea to solve this problem??

5 minutes to think about this problem ...

Check Prime Number (2)

❶ Start from 2 to N

❷ Check whether N is dividable by any number in this range

```
1  int i = 2, N = 177;
2  int _PRIME_ = 1;
3  while (i < N)
4  {
5      if (N%i == 0)
6      {
7          _PRIME_ = 0;
8      }
9  }
10 if (_PRIME_)
11     printf("%d is a prime number\n", N);
12 else
13     printf("%d is not a prime number\n", N);
```

- Do we miss anything?

Check Prime Number (3)

1. Start from 2 to N
2. Check whether N is dividable by any number in this range

```c
int i = 2, N = 177;
int _PRIME_ = 1;
while ( i < N )
{
    if ( N%i == 0)
    {
        _PRIME_ = 0;
    }
    i++;
}
if ( _PRIME_ )
    printf("%d is a prime number\n", N);
else
    printf("%d is not a prime number\n", N);
```

Check Prime Number (4)

1 In the above example, we no need to all the numbers in $[2 \cdots \text{N-1}]$

2 N is not dividable by numbers after $\lceil \sqrt{N} \rceil$

```c
#include <stdio.h>
#include <math.h>
int main()
{
    int i = 2, N = 177;
    int _PRIME_ = 1, bnd = (int)ceil(sqrt(N));
    while(i <= bnd)
    {
        if(N%i == 0)
        {
            _PRIME_ = 0;
        }
        i++;
    }
    if(_PRIME_)
        printf("%d is a prime number\n", N);
    else
        printf("%d is not a prime number\n", N);
    return 0;
}
```

Outline

do...while (1)

- do...while executes the statement(s) first
- Checks the condition after each run

```
1  int i = 3;
2  do
3  {
4      −−i;
5      printf("i=%d\n", i);
6  } while (i > 1);
```

```
1  int i = 3;
2  while (i > 1)
3  {
4      −−i;
5      printf("i=%d\n", i);
6  }
```

- What is the output??

do...while (2)

- do...while executes the statement(s) first
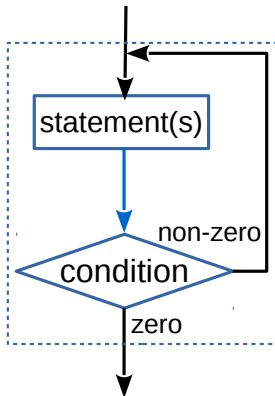- Checks the condition after each run

```
1  i=2
2  i=1
```

```
1  i=2
2  i=1
```

- What is the output??

- do...while executes the statement(s) first
- Checks the condition after each run

Example 1 (1)

- Calculate $x = \sqrt{a}$
- $x_{n+1} = \frac{1}{2}(x_n + \frac{a}{x_n})$
- Loop until error less than 10^{-5} in consecutive iterations
- Hints: x_1 is an arbitrary positive value

   3 minutes to think about this problem ...

Example 1 (2)

- Calculate $x = \sqrt{a}$
- $x_{n+1} = \frac{1}{2}(x_n + \frac{a}{x_n})$
- Loop until error less than 10^{-5} in consecutive iterations
- Hints: x_1 is an arbitrary positive value

1. We need a loop (while? do-while or for?)
2. We need to keep two results from consecutive iterations
3. Anything else??
4. Let's do it!

Example 1 (3)

- Calculate $x = \sqrt{a}$
- $x_{n+1} = \frac{1}{2}(x_n + \frac{a}{x_n})$
- Loop until error less than 10^{-5} in consecutive iterations
- Hints: x_1 is an arbitrary positive value

```
1   float a = 5, x0 = 3.1, xn = 0, err = 0;
2   do
3   {
4       xn  = 0.5*(xn+a/xn);
5       err = abs(xn - x0);
6   } while(err >= 0.00001)
7   printf("sqrt(a)=%f\n", xn);
```

- Anything wrong??

Example 1 (4)

- Calculate $x = \sqrt{a}$
- $x_{n+1} = \frac{1}{2}(x_n + \frac{a}{x_n})$
- Loop until error less than 10^{-5} in consecutive iterations
- Hints: x_1 is an arbitrary positive value

```
float a = 5, xk = 0, xn =  3.1, err = 0;
int i = 0;
do
{
    xk  = xn;
    xn  = 0.5*(xk+a/xk);
    //printf("%f\t%f\t%f\n", xk, xn, err);
    err = fabs(xn - xk);
    i++;
} while(err >= 0.00001);
printf("iters = %d, sqrt(a)=%f\n",  i, xn);
```

- Use "fabs(.)" instead of "abs(.)"

Outline
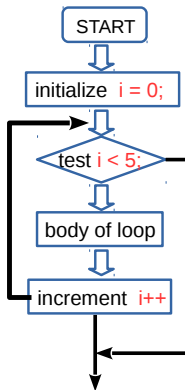
for loop (1)

The For-Loop is comfortable for iterating. It takes three arguments.

- Initialization (i=0;)
- Condition ($i < 5;$)
- Iteration statement (i+=1)

for loop (2)

- Consider a program printing the numbers 1 to 10:

```c
int i;
for (i = 1; i <= 10; ++i)
{
    printf("%d\n", i);
}
```

- i starts from 1
- Check if i is less than or equal to 10
- Go into the loop if it is true (non-zeor)
- Increment i, e.g., i++ or i+=2

break

- Similar as switch-case
- break can be used inside a loop
- Jumping out from the loop as soon as it is called

```c
int i, s = 0;
for (i = 1; i <= 10; ++i)
{
    s += 2*i;
    if(i%4 == 0)
    break;
}
printf("s=%d\n", s);
```

continue

- Different from break
- continue can be ONLY used inside a loop
- Ingore statements followed, go to next round of loop

```
1  int i, s = 0;
2  for (i = 1; i <= 10; ++i)
3  {
4      s += 2*i;
5      if(i%4 == 0)
6      break;
7  }
8  printf("s=%d\n", s);
```

```
1  int i, s = 0;
2  for (i = 1; i <= 10; ++i)
3  {
4      s += 2*i;
5      if(i%4 == 0)
6      continue;
7  }
8  printf("s=%d\n", s);
```

Outline

Ahmad Siavashi   **Fundamentals of Computer Programming**   26 / 50

Example 2 (1)

- Given following series of fraction
- $\frac{2}{1}$, $\frac{3}{2}$, $\frac{5}{3}$, $\frac{8}{5}$, $\frac{13}{8}$, $\frac{21}{13}$, ...
- Work out the sum of first _20_ terms

3 minutes to think about this problem ...

Example 2 (2)

- Given following series of fraction
- $\frac{2}{1}$, $\frac{3}{2}$, $\frac{5}{3}$, $\frac{8}{5}$, $\frac{13}{8}$, $\frac{21}{13}$, ...
- Work out the sum of first *20* terms

- We observe that numerator is the sum of the numerators of last two

- The denominator is the sum of the denominators of last two

- We have following things first

```
1  float n1 = 2, n2 = 3;
2  int d1 = 1, d2 = 2;
3  for (i = ?; i <= 20; ++i)
4  {
5      ....
6  }
```

Example 2 (3)

- Given following series of fraction
- $\frac{2}{1}, \frac{3}{2}, \frac{5}{3}, \frac{8}{5}, \frac{13}{8}, \frac{21}{13}, \ldots$
- Work out the sum of first *20* terms
- We need the variable to keep the result
- We need an iterator

```c
float n1 = 2, n2 = 3;
int d1 = 1, d2 = 2, i = 0;
float s = n1/d1 + n2/d2;
for (i = ?; i <= 20; ++i)
{
    ....
}
printf("s=%f\n", s);
```

Example 2 (4)

- Given following series of fraction
- $\frac{2}{1}$, $\frac{3}{2}$, $\frac{5}{3}$, $\frac{8}{5}$, $\frac{13}{8}$, $\frac{21}{13}$, ...
- Work out the sum of first *20* terms

- We need an iterator, start from where??

- How to work out *n1* and *d1*

```c
float n1 = 2, n2 = 3;
int d1 = 1, d2 = 2, i = 0;
float s = n1/d1 + n2/d2;
for(i = ?; i <= 20; ++i)
{
    n1 = ?;
    d1 = ?;
    s += n1/d1;
}
printf("s=%f\n", s);
```

Example 2 (5)

- Given following series of fraction
- $\frac{2}{1}, \frac{3}{2}, \frac{5}{3}, \frac{8}{5}, \frac{13}{8}, \frac{21}{13}, \cdots$
- Work out the sum of first *20* terms
- The full story

```c
1  #include <stdio.h>
2  int main()
3  {
4     float n1 = 2, n2 = 3;
5     int d1 = 1, d2 = 2, i = 0;
6     float s = n1/d1 + n2/d2;
7     for(i = 3; i <= 20; ++i)
8     {
9        n2 = n1 + n2;
10       d2 = d1 + d2;
11       s += n2/d2;
12       n1 = n2 - n1;
13       d1 = d2 - d1;
14    }
15 }
```

```c
15    printf("s=%f\n", s);
16    return 0;
17 }
```

Example 3 (1)

- Output following figure

$$
\begin{array}{c}
* \\
*** \\
***** \\
******* \\
***** \\
*** \\
*
\end{array}
$$

Example 3 (2)

- On the 1st line, we print 3 blanks and 1 star
- On the 2nd line, we print 2 blanks and 3 stars
- On the 3rd line, we print 1 blanks and 5 stars
- On the 4th line, we print 0 blank and 7 stars
- Do the following in reverse...
- There should be a loop controls of printing blanks of one line
- There should be a loop controls of printing stars of one line
- There should be a loop controls of printing all the lines
- How to organize them??

Example 3 (3)

- There should be a loop controls of printing blanks of one line
- There should be a loop controls of printing stars of one line
- There should be a loop controls of printing all the lines
- How to organize them??

1. Loop print all lines
2.   Loop print blank(s)
3.   Loop print star(s)
4. End-Loop

Example 3 (4)

① Loop print all lines

②   Loop print blank(s)

③   Loop print star(s)

④ End-Loop

① We need a bound for the number of stars

② We need a bound for the number of blanks

```
1  int n = 7, i = 0, j = 0;
2  int ns = 1, nb = n−1;
3  for(i = 0; i < n; i++)
4  {
5      ...
6  }
```

Example 3 (5)

① Print things to the bounds

```
int n = 5, i = 0, j = 0;
int ns = 1, nb = n-1;
for(i = 0; i < n; i++)
{
    for(j = 0; j < nb; j++)
    {
        printf(" ");
    }
    for(j = 0; j < ns; j++)
    {
        printf("*");
    }
    ns += 2;
    nb--;
    printf("\n");
}
```

```
nb += 2; //<——why?
ns -= 4; //<——why?
for(i = 0; i < n; i++)
{
    for(j = 0; j < nb; j++)
    {
        printf(" ");
    }
    for(j = 0; j < ns; j++)
    {
        printf("*");
    }
    ns -= 2;
    nb++;
    printf("\n");
}
```

Example 4 (1): solving by exhaustive search

- 30 people dine together, 50 cents are paid
- It takes 3 cents for a gentleman
- It takes 2 cents for a lady
- It takes 1 cent for a child
- How many gentlemen, ladies and children are there

$$\begin{cases} 3*x + 2*y + z = 50 \\ x + y + z = 30 \end{cases}$$

- We are actually trying to solve linear equations
- Notice that only integer solutions are valid

Example 4 (2): solving by exhaustive search

- Solution is, we enumerate all possible solution
- To see whether they satisfy all the equations

$$\begin{cases} 3*x + 2*y + z = 50 \\ x + y + z = 30 \end{cases}$$

- Enumerate x from 1 to 30
- Enumerate y from 1 to 30
- Enumerate z from 1 to 30
- Now let's do it!

Example 4 (3): solving by exhaustive search

- Enumerate x from 1 to 30
- Enumerate y from 1 to 30
- Enumerate z from 1 to 30

$$\begin{cases} 3*x + 2*y + z = 50 \\ x + y + z = 30 \end{cases}$$

```
for(x = 0; x <= 30; x++)
{
    for(y = 0; y <= 30; y++)
    {
        for(z = 0; z <= 30; z++)
        {
            ....
        }
    }
}
```

Example 4 (4): solving by exhaustive search

```c
#include <stdio.h>
int main()
{
    int x = 0, y = 0, z = 0, c1 = 0, c2 = 0;
    for(x = 0; x <= 30; x++)
    { for(y = 0; y <= 30; y++)
        { for(z = 0; z <= 30; z++)
            {
                c1 = 3*x+2*y+z;
                c2 = x+y+z;
                if(c1 == 50 && c2 == 30)
                {
                    printf("x = %d, y = %d, z = %d\n", x, y, z);
                }
            }//for(z)
        }//for(y)
    }//for(x)
}
```

Example 4 (5): solving by exhaustive search

```c
#include <stdio.h>
int main()
{
    int x = 0, y = 0, z = 0, c1 = 0, c2 = 0;
    for(x = 0; x < 17; x++) //<— why??
    { for(y = 0; y <= 25; y++) //<— why??
        { for(z = 0; z <= 30; z++)
            {
                c1 = 3*x+2*y+z;
                c2 = x+y+z;
                if(c1 == 50 && c2 == 30)
                {
                    printf("x = %d, y = %d, z = %d\n", x,y,z);
                }
            }//for(z)
        }//for(y)
    }//for(x)
}
```

Outline

Try to avoid following case

- Be careful, this

```
1 while (1 > 0)
2     printf("Did you miss me?\n");
```

- Runs till the end of all days
- ∞ loops are common mistakes, and you will experience many of them
- Check for conditions that are always true
- By the way,

Do not be evil!

Valid variants of for-loop (1)

- The arguments for the for loop are optional
- If you already have defined your iterating variable

```
int i = 1;
for (; i <= 10; ++i)
    printf("%d\n", i);
```

- Or if you have the iteration statement in your loop body

```
for (i = 1; i <= 10;)
    printf("%d\n", ++i);    /* seems more like a while loop
    */
```

Valid variants of for-loop (2)

- If you're not passing anything, it runs **for**ever

```
for (;;)
    printf("I'm still here\n");
```

Note: the semicolons are still there.

Outline

Overview about Programming Bugs (1)

- It happens!
- Two types of bugs
  1. Grammar mistakes
  2. Logic bugs

```
1 int main()
2 {
3     int i = 0, s = 0;
4     for(i = 0; i < 5; i++)
5     {
6         s += i;
7     }
8     return 0;
9 }
```

```
1 int main()
2 {
3     int i = 0, s;
4     for(i = 0; i < 5; i++);
5     {
6         s += i;
7     }
8     return 0;
9 }
```

Overview about Programming Bugs (2)

- From my humble point of view
- Two types of bugs
  1. Non-memory leakage bugs
  2. Memory leakage bugs
- Grammar mistakes are not bug!!

```
1  int main()
2  {
3      int i = 0, s = 0;
4      for(i = 0; i < 5; i--);
5      {
6          s += i;
7          if(i % 2 == 1);
8              continue;
9      }
10     return 0;
11 }
```

```
1  int main()
2  {
3      int i = 0;
4      int *p = NULL;
5      for(i = 0; i < 5; i++)
6      {
7          p = NULL;
8          p = (int *)malloc(
9          sizeof(int));
10     }
11     return 0;
12 }
```

Debug by "printf()"

- From my humble point of view
- Two types of bugs
  1. Non-memory leakage bugs
  2. Memory leakage bugs
- Grammar mistakes are not bug!!

```c
int main()
{
    int i = 0, s = 0;
    printf("bug 1.0\n");
    for(i = 0; i < 5; i--);
    {
        s += i;
        printf("bug 1.1 %d %d\n", i, s);
        if(i % 2 == 1);
            continue;
    }
    printf("bug 2.0\n");
    return 0;
}
```