C Programming
Lecture 4: Loop Control ↺

Lecturer: *Dr*. Wan-Lei Zhao
Autumn Semester 2022

[1] Email: wlzhao@xmu.edu.cn, copyrights are fully reserved by the author.

Outline

1. Loops
    - while loop
    - do-while loop
    - for loop
    - Examples of Loops

2. Miscellaneous

3. How to Debug

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
int main()
{
    int x = 1;
    double S = 0;
    S += 1.0/(x*x);
    x += 1;
    S += 1.0/(x*x);
    x += 1;
    S += 1.0/(x*x);
    x += 1;
    S += 1.0/(x*x);
    x += 1;
    S += 1.0/(x*x);
    printf("S = %lf\n", S);
    return 0;
}
```

- 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

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

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

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

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

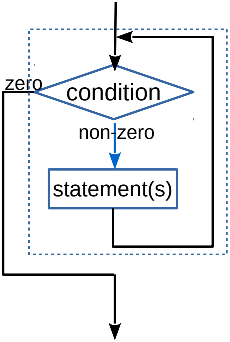
1. Loops
   - while loop
   - do-while loop
   - for loop
   - Examples of Loops

2. Miscellaneous

3. How to Debug

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 of 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 of 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 of 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 of 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 of 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 of 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 of 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**

Check Prime Number (1)

- Check whether an integer number if 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)

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;
      }
}
if(_PRIME_)
    printf("%d is a prime number\n", N);
else
    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

```
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

1. Loops
    - while loop
    - do-while loop
    - for loop
    - Examples of Loops
2. Miscellaneous
3. How to Debug

do...while (1)

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

```c
int i = 3;
do
{
    --i;
    printf("i=%d\n", i);
}while (i > 1);
```

```c
int i = 3;
while(i > 1)
{
    --i;
    printf("i=%d\n", i);
}
```

- What is the output??

do...while (2)

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

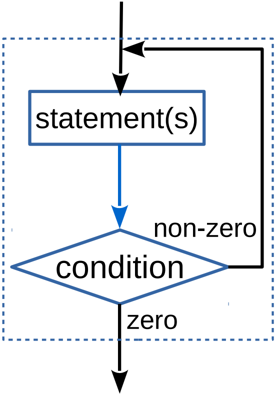
```
i =2
i =1
```

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

- What is the output??

do...while (3)

- 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

```
 float a = 5, x0 = 3.1, xn = 0, err = 0;
do
{
    xn   = 0.5*(xn+a/xn);
    err = abs(xn - x0);
}while(err >= 0.00001)
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

```c
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

1. Loops
   - while loop
   - do-while loop
   - for loop
   - Examples of Loops

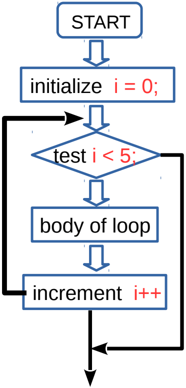
2. Miscellaneous

3. How to Debug

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:

```
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

```
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

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

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

Outline

1. Loops
   - while loop
   - do-while loop
   - for loop
   - Examples of Loops
2. Miscellaneous
3. How to Debug

Example 2 (1)

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

3 minutes to think about this problem ...

Example 2 (2)

- Given following series of fraction
- $\frac{2}{1}, \frac{3}{2}, \frac{5}{3}, \frac{8}{5}, \frac{13}{8}, \frac{21}{13}, \ldots$
- 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

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

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
- $\frac{2}{1}, \frac{3}{2}, \frac{5}{3}, \frac{8}{5}, \frac{13}{8}, \frac{21}{13}, \ldots$
- 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
- $\frac{2}{1}, \frac{3}{2}, \frac{5}{3}, \frac{8}{5}, \frac{13}{8}, \frac{21}{13}, \dots$
- Work out the sum of first *20* terms
- The full story

```c
#include <stdio.h>
int main()
{
   float n1 = 2, n2 = 3;
   int d1 = 1, d2 = 2, i = 0;
   float s = n1/d1 + n2/d2;
   for(i = 3; i <= 20; ++i)
   {
       n2 = n1 + n2;
       d2 = d1 + d2;
       s += n2/d2;
       n1 = n2 - n1;
       d1 = d2 - d1;
   }
}
```

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

Example 3 (1)

- Output following figure

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

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

```c
int n = 7, i = 0, j = 0;
int ns = 1, nb = n-1;
for(i = 0; i < n; i++)
{
    ...
}
```

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

```
#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

1. Loops
   - while loop
   - do-while loop
   - for loop
   - Examples of Loops

2. Miscellaneous

3. How to Debug

Try to avoid following case

- Be careful, this

```
while (1 > 0)
      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

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

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

```c
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

1. Loops
   - while loop
   - do-while loop
   - for loop
   - Examples of Loops

2. Miscellaneous

3. How to Debug

Story behind Bug

Overview about Programming Bugs (1)

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

```c
int main()
{
    int i = 0, s = 0;
    for(i = 0; i < 5; i++)
    {
        s += i;
    }
    return 0;
}
```

```c
int main()
{
    int i = 0, s;
    for(i = 0; i < 5; i++);
    {
        s += i;
    }
    return 0;
}
```

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!!

```c
int main()
{
    int i = 0, s = 0;
    for(i = 0; i < 5; i--);
    {
        s += i;
        if(i % 2 == 1);
           continue;
    }
    return 0;
}
```

```c
int main()
{
    int i = 0;
    int *p = NULL;
    for(i = 0; i < 5; i++)
    {
        p = NULL;
        p = (int*)malloc(sizeof(int));
    }
    return 0;
}
```

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;
}
```