Introduction to C

Tutorial 6: Loops
Last week...

- If statements
- Switch statements
Agenda

- Types of loops in C
- Examples
Types of Loops
while loops

• Structure of the loop

```
while (expression)
    statement
```

– If `expression` is true (not 0), `statement` will be executed.
– After this we check again : If `expression` is true again , `statement` will be executed again.
– And so forth, until `expression` is no longer true.

• `-statement` will never get executed if `expression` is never true.
Example: Inputting positive numbers

```c
int num;
printf("Enter a positive number: ");
scanf("%d", &num);
while (num <= 0)
{
    printf("Number must be positive.\n");
    printf("Please try again: ");
    scanf("%d", &num);
}
```
do-while loop

- Loop structure:

```c
    do {
       statement
    }
    while (expression);
```

- `statement` gets executed.
- after this, we check `expression`. If it is true, then we execute `statement`, and so forth.

- `statement` will get executed at least once no matter what
Example: Inputting positive numbers

```c
int num;
do {
    printf("Enter a positive number: ");
    scanf("%d", &num);
} while (num <= 0);
```

What is the difference to the user if they typed a positive number on the first try?

What is the difference for the user between the two versions, if you get a few failed attempts?
for loops

• Loop structure:
  
  for \((expression_1; expression_2; expression_3)\)
  
  \textit{statement}

  – \textit{expression}_1 is the initialization step
  – If \textit{expression}_2 is true:
    • \textit{statement} is executed. This is the “body of the loop.”
    • \textit{expression}_3 is executed: This is the finishing step of the iteration.
    – Again, we check that \textit{expression}_2 is true, and if so \textit{statement}
      and then \textit{expression}_3 are executed.

• \textit{expression}_1 is only executed once.
• \textit{statement} and \textit{expression}_3 get executed the same number of times – 0 or more.
Example: Calculate sum of squares.

- The expression: \[ \sum_{i=1}^{n} i^2 \]

```c
sum = 0;
for(i = 1; i <= n; i++)
    sum += i * i;
```
The “comma” operator

• Using the comma operator “,”:
  – Calculating an expression of form:
    \[ \text{exp}_1, \text{exp}_2 \]
    \( \text{exp}_1 \) executes and afterward \( \text{exp}_2 \) executes.
  – The value (type) of the combined expression is the value (type) the last expression.
  – You can chain several instructions, for example:
    \[ \text{exp}_1, \text{exp}_2, \text{exp}_3 \]
  – Therefore, another way to execute the sum above is:
    \[
    \text{for (sum = 0, i = 1; i <= n; sum += i * i, i++);}
    \]
  – In general, expressions in parentheses should only have expressions related to loop control. Thus, the first solution is preferable.
break and continue instructions

• The break instruction results in the loop ending immediately
  – Without checking the loop condition again

• The continue instruction results in the ending of the current **iteration**
  – The loop condition is checked again – if it is true, we continue in the loop. Else, exit the loop.
  – For loops execute expression\(^3\) before checking the condition.
  – Can only appear in for, while, and do-while loops.
    • What about break – can it appear in other places?
Example using `continue`

```c
for(i = 0; i < 100; i++) {
    scanf("%d", &num);
    if(num < 0)
        continue;
    sum += num;
}
```

- Reads 100 integers and adds the non-negative ones.
Example using `break`

- Adds numbers until a negative number is input by the user.

```c
while(1) {
    scanf("%d", &num);
    if(num < 0)
        break;
    sum += num;
}
```
Infinite loops

• A loop whose stopping condition is never reached is called an **infinite loop**.

An infinite loop is only useful when a special instruction to end the loop is used, such as a return or a break.

• What is problematic about the following loops?

  1) for (i=40 ; i>0 ; i++)

  2) for (i=1 ; i != 40 ; i+=2)

  3) int i;
    for ( ; i<40 ; i++)

  4) while (i=2)

• Which of these loops are infinite?
Nested loops

• A loop that contains one or more loops inside of it is called a nested loop
• Example: what will the following code print?

```c
int i, j, n=5; 
for (i=1; i<=n; ++i) {
    for (j=1; j<=n; ++j) {
        printf("%d,%d\n",i,j);
    }
}
```
while loop

while (expression)
  statement

expression

0 ≠ 0

continue

break

statement
do-while loop

```
do
    statement
while (expression);
```
for loop

for(expression\_1; expression\_2; expression\_3)
  statement

expression\_1

expression\_2

expression\_3

0 ≠ 0

expression\_2

break
continue
Example 1: Counting numbers
Example

- We write a program that accepts a number \( x \) and counts how many times a digit \( d \) appears in the input.
- Example:
  - The number 1 appears twice in 215610
  - The number 0 appears once in 10
- Assume that no number will be negative.
#include <stdio.h>

int main() {
    int x, d, count=0;
    printf("Please enter a number and a digit:\n");
    scanf("%d%d", &x, &d);
    do {
        if (x%10 == d)
            count++;
        x /= 10;
    } while (x > 0);
    printf("%d", count);
    return 0;
}

• What needs to be changed so that the program will work for negative numbers?
Fix for negative numbers

```c
int main() {
    int x, d, count=0;
    printf("Please enter a number and a digit:\n");
    scanf("%d%d", &x, &d);
    do {
        if (x%10 == d || x%10 == -d)
            count++;
        x /= 10;
    } while (x != 0);
    printf("%d", count);
    return 0;
}
```
Example II: Adding without +
The problem...

• You’ve been asked to write a program that adds numbers on an old computer.
• It’s so old that it can’t even add two numbers.
  • In other words, you cannot use *, -, +, or /.
  • The only operators that can be used are (increment) ++ and (decrement) --.
The challenge

- **Input:** two whole numbers
- **Output:** their sum
- **Execution** can take place only with ++ and --!

```c
int a;
int b;
int total;
scanf("%d", &a);
scanf("%d", &b);
....
printf("%d + %d = %d\n", a, b, total);
```
int a;
int b;
int total;
int i;

scanf("%d", &a);
scanf("%d", &b);

total = a;
for (i = 0; i < b; i++)
    total++;

printf("%d + %d = %d\n", a, b, total);
The next challenges

• Computing a subtraction between a and b requires only a small change. How will it be done?

• What about writing a program that does multiplication between a and b (a*b) using the same rules?

• ...What about exponent $a^b$?

• ...and division a/b?
Example III: Searching DNA
DNA (molecules containing genetic information in living cells) consists of four building blocks (nitrogen bases):

- **Adenine**
- **Cytosine**
- **Guanine**
- **Thymine**

We will use the first letter (A / C / G / T) to represent each base.
Searching through DNA

• Technion scientists have discovered that people with that have the DNA sequence “ATTAC” are prone to violence
• We will write a program that searches through DNA and will search for this sequence
• **Input:** A sequence of letters
  – End of the sequence will be marked by an X
• **Output:** Whether or not the sequence was found.
How will we execute the search

• We are looking for a sequence of 5 characters within a larger sequence (where size is not known in advance)

Consider this like looking at a line with a magnifying glass:
How do we execute the search?

• At any given moment we keep the last five characters we read.
• In variables n1, n2... n5 , of type char.

AGGCTACGAGTAGCATTACTAGCGATCGAGTCGA

Characters that we read and checked earlier, and that no longer interest us.

Characters under the “magnifying glass”

Characters that have not been read yet. They will be read later on.
How to we proceed?

- Will we enter 5 characters each time and check them?

AGGCTACGAGTAGCATTACTAGCGATCGAGTCGA
AGGCTACGAGTAGCATTACTAGCGATCGAGTCGA
AGGCTACGAGTAGCATTACTAGCGATCGAGTCGA
AGGCTACGAGTAGCATTACTAGCGATCGAGTCGA
AGGCTACGAGTAGCATTACTAGCGATCGAGTCGA
AGGCTACGAGTAGCATTACTAGCGATCGAGTCGA
How to we proceed?

• Each time we need to read in one character
  
  “Throw away” the oldest character —
  – The second character becomes the first (n1 gets the value of n2)
  – And so on....
  – At the end, n5 gets a new value.
The program: Declaring and initializing variables

• We'll be sure to initialize the variables to unacceptable values to avoid incorrect results due to random initialization!

```c
#include <stdio.h>

int main()
{
    char n1 = 0;
    char n2 = 0;
    char n3 = 0;
    char n4 = 0;
    char n5 = 0;
    char tmp;
```

This variable will be used for input. Only if we see that a valid value was received (a letter from ACGT) we’ll use this to update the 5 nucleotides.
The program: Input and its validation

```c
while(1) {
    scanf("%c", &tmp);
    switch(tmp) {
        case 'X':
            printf("Not found.\n");
            return 0;
        case 'A': case 'C': case 'G': case 'T':
            n1 = n2;
            n2 = n3;
            n3 = n4;
            n4 = n5;
            n5 = tmp;
    }
}
```

The program runs in an infinite loop. We exit the loop if we discover an X or we find the sequence “ATTAC”

“X” marks the end of the sequence

Valid input: We update the current nucleotide.

We ignore all other input
The program: Bingo?

At the end of every round we see if the desired value was found

if (n1 == 'A' && n2 == 'T' && n3 == 'T' && n4 == 'A' && n5 == 'C')
{
    printf("ATTAC found!\n");
    break;
}

return 0;

End of while loop

If yes, stop

End of main

At the end of every round we see if the desired value was found