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){
  statements
}

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

• The statements will never get executed if expression is false.
Example: Inputting positive numbers

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

• Loop structure:
  
  ```
  do {
    statements
  } while (expression);
  ```

  — First of all, `statements` are being executed.
  — after this, `expression` is checked. If it is true, then we execute again `statements`, and so forth.

• Note that if `expression` is false, `statements` will be executed only once.
Example: Inputting positive numbers

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

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

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

- Loop structure:
  for \((\text{expression}_1; \text{expression}_2; \text{expression}_3)\){
      \text{statement}
  }

- \text{expression}_1\) is the initialization step, therefore it is executed only once.

- If \text{expression}_2\) is true:
  - \text{statement} is executed.
  - \text{expression}_3\) is executed: this is the finishing step of the iteration.

- Again, we check that \text{expression}_2\) is true, and if so \text{statement} and then \text{expression}_3\) are executed.
```c
#include <stdio.h>
int main(void)
{
    int count;
    for (count = 1; count <= 500; count++)
        printf("I will not throw paper airplanes in class.");
    return 0;
}
```
Example: Calculating factorial

- The expression: $n!$

```java
factorial = 1;
for(i = 1; i <= n; i++){
    factorial *= i;
}
```
The “comma” operator

• Using the comma operator “,”:
  – Calculating an expression of form:
    \[ \text{exp}_1, \text{exp}_2 \]
    \[ \text{exp}_1 \text{ executes and afterward } \text{exp}_2 \text{ executes.} \]
  – 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 calculation above is:
\[
\text{for (factorial = 1, i = 1; i } \leq \text{n; factorial } \times= \text{i, i++);}
\]
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. Otherwise, we exit the loop.
  – For loops execute *expression*₃ before checking the condition.
  – Can only appear in the loops we saw.
    • What about break – can it appear in other places?
Example using continue

```c
for(i = 0; i < 4; i++) {
    scanf("%d", &num);
    if(num < 0)
        continue;
    sum += num;
}
```

- Reads 4 integers and adds the non-negative ones.
Example using break

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

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

• A loop which its stopping condition is never false, is called an **infinite loop**.
• An infinite loop is useful when a special instruction is used to end the loop, such as a **return** or a **break**.
• What is problematic about the following loops?
• Which of these loops are infinite?

```c
int i,j;
1) for (i=40 ; i>0 ; i++)
2) for (i=1 ; i != 40 ; i+=2)
3) for ( ; j<40 ; j++)
4) while (i=2)
```
Nested loops

- A loop that contains one or more loops inside 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

continue
    statement

break

expression
0
do-while loop

do statement
while(expression);

statement
break
continue

expression
≠ 0

0
for loop

\[
\text{for}(\text{expression}_1; \text{expression}_2; \text{expression}_3)
\]

\[
\text{statement}
\]

expression1

expression2

\[
0
\]

\[
\neq 0
\]

break
continue

expression3

statement
Example 1: Counting numbers
Example

• Write a program which accepts a number x and a digit d, and then counts how many times d appears in x.

• Example:
  – The number 1 appears twice in 215610
  – The number 0 appears once in 10

• Assume that d isn’t negative, and x is positive.
Example

```c
#include <stdio.h>

int main() {
    int x, d, count=0;
    printf("Please enter a number and a digit");
    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?
int main() {
    int x, d, count=0;
    printf("Please enter a number and a digit");
    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...

- Write a program that adds numbers on an old computer. How old?
- 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) --

Pascal's calculator (1652)
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) 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 discovered that people which have the DNA sequence “ATTAC” are prone to violence.
• Write a program that searches through DNA 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.

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

Characters under the “magnifying glass”

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

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

AGGCTACGAGTACGATTACTAGCGATCGAGTCGA
AGGCTACGAGTACGATTACTAGCGATCGAGTCGA
AGGCTACGAGTACGATTACTAGCGATCGAGTCGA
AGGCTACGAGTACGATTACTAGCGATCGAGTCGA
AGGCTACGAGTACGATTACTAGCGATCGAGTCGA
AGGCTACGAGTACGATTACTAGCGATCGAGTCGA
How do 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
int main(){
    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

"X" marks the end of the sequence

Valid input: We update the current nucleotide.

We ignore all other input
The program: Bingo?

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

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

If yes, stop and break the loop.

End of while loop.

End of main.
Fibonacci Numbers

• Write a program that gets a positive integer n as input, and prints the first n Fibonacci numbers.

• For example:
  If n=8 the output will be

  1, 1, 2, 3, 5, 8, 13, 21
#include <stdio.h>

int main()
{
    int a = 0, b = 1, c = 1, i, n;
    /* Input number from user */
    printf("Enter a positive number: ");
    scanf("%d", &n);

    printf("Fibonacci numbers! \n");

    /* Iterate through n terms */
    for(i=1; i<=n; i++)
    {
        printf("%d," , c);
        c = a + b; // New Fibo number
        a = b; // Copy n-1 to n-2
        b = c; // Copy current to n-1
    }

    return 0;
}