Outline

- Program Looping
  - The for Statement
  - Relational Operators
  - Nested for Loops
  - Increment Operator
  - Program Input
    - for Loop Variants
  - The while Statement
  - The do Statement
  - The break Statement
  - The continue Statement
Program Looping

• Looping: doing one thing over and over
• Program loop: a set of statements that is executed repetitively for a number of times
• Forms of controlling the program flow:
  – Executing a sequence of statements
  – Repeating a sequence of statements (until some condition is met) (looping)
  – Using a test to decide between alternative sequences (branching)
The need for program looping

Example problem: computing triangular numbers.
(The n-th triangular number is the sum of the integers from 1 through n)

```c
#include <stdio.h>
int main (void) {
    int triangularNumber;
    triangularNumber = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8;
    printf ("The eighth triangular number is %i\n", triangularNumber);
    return 0;
}
```

What if we have to compute the 200-th (1000-th, etc) triangular number?

Program looping: enables you to develop concise programs containing repetitive processes that could otherwise require many lines of code!

In C: 3 different statements for looping: for, while, do
Example - for

/* Program to calculate the 200th triangular number
Introduction of the for statement */

#include <stdio.h>
int main (void)
{
    int n, triangularNumber;
    triangularNumber = 0;
    for ( n = 1; n <= 200; n = n + 1 )
        triangularNumber = triangularNumber + n;
    printf ("The 200th triangular number is %i\n",
            triangularNumber);
    return 0;
}
Example - for

Statement before loop

init_expression

n=1

loop_condition

n<=200

yes

statement

triangularNumber = triangularNumber + n

no

loop_expression

n=n+1

Statement after loop

Print triangularNumber
The for statement

for ( init_expression; loop_condition; loop_expression )
program statement

1. init_expression
2. loop_condition
   - yes: Program statement
   - no: Loop expression
3. Program statement
4. Loop expression
5. loop_condition
The \texttt{for} statement

\begin{align*}
\text{for } & (\ n = 1; \ n \leq 200; \ n = n + 1) \\
& \text{triangularNumber} = \text{triangularNumber} + n;
\end{align*}
How for works

• The execution of a for statement proceeds as follows:
  1. The initial expression is evaluated first. This expression usually sets a variable that will be used inside the loop, generally referred to as an *index* variable, to some initial value.
  2. The looping condition is evaluated. If the condition is not satisfied (the expression is false – has value 0), the loop is immediately terminated. Execution continues with the program statement that immediately follows the loop.
  3. The program statement that constitutes the body of the loop is executed.
  4. The looping expression is evaluated. This expression is generally used to change the value of the index variable.
  5. Return to step 2.
Infinite loops

• It’s the task of the programmer to design correctly the algorithms so that loops end at some moment!

// Program to count 1+2+3+4+5
#include <stdio.h>
int main (void)
{
    int i, n = 5, sum =0;
    for ( i = 1; i <= n; n = n + 1 ){
        sum = sum + i;
        printf ("%i %i %i\n", i , sum, n);
    }
    return 0;
}
Relational operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>Is equal to</td>
</tr>
<tr>
<td>!=</td>
<td>Is not equal to</td>
</tr>
<tr>
<td>&lt;</td>
<td>Is less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Is less or equal</td>
</tr>
<tr>
<td>&gt;</td>
<td>Is greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Is greater or equal</td>
</tr>
</tbody>
</table>

The relational operators have lower precedence than all arithmetic operators: \( a < b + c \) is evaluated as \( a < (b + c) \)

ATTENTION! Do not confuse: the “is equal to” operator == and the “assignment” operator =

ATTENTION when comparing floating-point values! Only < and > comparisons make sense!
Example – for with a body of 2

// Program to generate a table of triangular numbers
#include <stdio.h>
int main (void)
{
    int n, triangularNumber;
    printf ("TABLE OF TRIANGULAR NUMBERS\n\n");
    printf (" n Sum from 1 to n\n");
    printf ("--- ---------------\n");
    triangularNumber = 0;
    for ( n = 1; n <= 10; ++n ) {
        triangularNumber += n;
        printf (" %i %i\n", n, triangularNumber);
    }
    return 0;
}

The body of the loop consists in a block of 2 statements
Increment operator

- Because addition by 1 is a very common operation in programs, a special operator was created in C for this.
- Increment operator: the expression `++n` is equivalent to the expression `n = n + 1`.
- Decrement operator: the expression `--n` is equivalent to `n = n – 1`.
- Increment and decrement operators can be placed in front (prefix) or after (postfix) their operand.
- The difference between prefix and postfix: can be noticed only in certain cases:
  - Example: if `n=4`:
    - `a=n++` leads to `a=4`, `n=5`  
      ```c
      // a = n++
      ```
    - `a=++n` leads to `a=5`, `n=5`  
      ```c
      // a = ++n
      ```
#include <stdio.h>
int main (void)
{
    int n, number, triangularNumber;
    printf ("What triangular number do you want? ");
    scanf ("%i", &number);
    triangularNumber = 0;
    for ( n = 1; n <= number; ++n )
        triangularNumber += n;
    printf ("Triangular number %i is %i\n", number, triangularNumber);
    return 0;
}

Scanf: similar to printf: first argument contains format characters, next arguments tell where to store the values entered at the keyboard
More details -> in a later chapter!
Nested loops

```c
#include <stdio.h>
int main (void) {
    int n, number, triangularNumber, counter;
    for ( counter = 1; counter <= 5; ++counter ) {
        printf ("What triangular number do you want? ");
        scanf ("%i", &number);
        triangularNumber = 0;
        for ( n = 1; n <= number; ++n )
            triangularNumber += n;
        printf ("Triangular number %i is %i\n\n", number, triangularNumber);
    }
    return 0;
}
Remember indentations!
```
for loop variants

- Multiple expressions (comma between …)
  ```
  for(i=0 , j=10 ; i<j ; i++ , j--)
  ```

- Omitting fields (semicolon have to be still …)
  ```
  i=0;
  for( ; i<10 ; i++ )
  ```

- Declaring variables
  ```
  for(int i=0 ; i=10 ; i++ )
  ```
#include <stdio.h>
int main (void)
{
    int count = 1;
    while ( count <= 5 ) {
        printf ("%i\n", count);
        ++count;
    }
    return 0;
}
The `while` statement

```c
while ( expression )
    program statement
```

while ( number <= 0 ) {
    printf ("The number must be >0");
    printf ("Give a new number: ");
    scanf("%i", &number);
}
The **while** statement

```plaintext
while ( expression )
  program statement
```

Loop with the test in the beginning!
Body might never be executed!
/* Program to find the greatest common divisor of two nonnegative integer values */
#include <stdio.h>
int main (void)
{
    int u, v, temp;
    printf ("Please type in two nonnegative integers.\n");
    scanf ("%i%i", &u, &v);
    while ( v != 0 ) {
        temp = u % v;
        u = v;
        v = temp;
    }
    printf ("Their greatest common divisor is %i\n", u);
    return 0;
}
Example - while

// Program to reverse the digits of a number
#include <stdio.h>
int main (void)
{
    int number, right_digit;
    printf("Enter your number. \n");
    scanf("%i", &number);
    while ( number != 0 ) {
        right_digit = number % 10;
        printf("%i", right_digit);
        number = number / 10;
    }
    printf("\n");
    return 0;
}

The **do** statement

```
do
  program statement
while ( loop_expression );
```

Loop with the test at the end! Body is executed at least once!
Example – do while

// Program to reverse the digits of a number
#include <stdio.h>
int main ()
{
    int number, right_digit;
    printf ("Enter your number.\n");
    scanf ("%i", &number);
    do {
        right_digit = number % 10;
        printf ("%i", right_digit);
        number = number / 10;
    }
    while ( number != 0 );
    printf ("\n");
    return 0;
}
Which loop to choose?

• Criteria: Who determines looping
  – Entry-condition loop -> for, while
  – Exit-condition loop -> do

• Criteria: Number of repetitions:
  – Indefinite loops -> while
  – Counting loops -> for

• In C, you can actually rewrite any while as a for and vice versa!
Statements `break` and `continue`

- Programming style: don’t abuse `break` !!!

```c
... 
while ( number != 0 ) { 
    // Statements to do something in loop 
    printf("Stop, answer 1: ");
    scanf ("%i", &answer);
    if(answer == 1) 
        break; // very bad idea to do this 
}
```
Statements break and continue

Continue also not so good style!!!

...  

while ( number != 0 ) {
     // Statements to do something in loop
     printf("Skip next statements answer 1: ");
     scanf ("%i", &answer);
     if(answer == 1)
         continue; // not so good idea...
     // Statements to do something in loop
     // If answer was 1 these statements are
     // not executed. They are skipped.
     // Go straight to the beginning of while
}