Course
ELEC-A7100
Course materials
Your points
Microsoft Teams
Code Vault

This course has already ended.

« 2 Conditional statements Course materials 4 Round feedback »
ELEC-A7100 / 2. Input and output / 3 Loops

Second round tasks as a `ZIP file`.

Now that the C basic data types and syntax start to be in order, in this module we get familiar with **input** and **output**, and **control structures** that are essential in programming, such as loops and conditional statements. If you have programmed before with any language, basics of control structures are probably familiar, but the syntax in C differs from many other languages, particularly Python.

After the module you can read user input to different types of variables, and output data using varying formatting rules. The conditional and loop statements will also become familiar.

Loops¶

while and do-while loops¶

The **while** loop repeats a single statement or compound statements as long as the specified expression is true (i.e., has non-zero value). Below is an example of a simple loop that repeats until the value of a is 10 (or lower).

```
int a = 0;  // Initializing the variable is important in case of loops.
while (a < 10)
    a++;
```

The termination condition is tested before executing the statement. Therefore it is possible that the statement is never executed, if the termination condition is false from the beginning.

As always, the statement inside while loop can be a compound statements, as in the following:

```
#include <stdio.h>

int main(void)
{
    int a = 0;
    while (a < 10)
    {
        printf("value of a is now %d\n", a);
        a++;
    }

    return 0;
}
```

Modify the above program so that it will print the numbers from 100 to 10 in the reverse order (in the interval of 10) ie:

```
value of a is now 100
value of a is now 90
value of a is now 80
```

and so on.

If it is desired to test the termination condition after the (compound) statement, a **do-while** loop can be used as follows:

```
#include <stdio.h>

int main(void)
{
    int a = 20;
    do {
        printf("value of a is now %d\n", a);
        a++;
    } while (a < 10);

    return 0;
}
```

Because in this example the initial value of 'a' is 20, it prints one line before the termination condition is evaluated, and the loop is terminated.

for loop¶

for is another statement for constructing loops, and can be used as a convenient alternative to **while** in many cases. It takes the form

```
for (init; condition; operation)
    expression;
```

The above **for** construct could be built using **while**, in which case an equivalent pattern would be:

```
init;
while (condition) {
    expression;
    operation;
}
```

- **init** initializes the for loop.
- **condition** is the iteration check.
- **operation** is the adjustment expression at the end of the for loop.

It is possible to use any loop in C language. It is therefore possible that condition to update any variable and at the same time used to check the loop ending condition.

For example, it is possible to re-write the above **while loop** example with for loop like below:

```
int a;
for (a = 0; a < 10; a++) {
    printf("value of a is now %d\n", a);
}
```

Any of the three expressions can be empty. For example, the following programs has init and operation empty.

```
int a = 0;
for ( ; a < 10; ) {
    printf("value of a is now %d\n", a);
    a++;
}
```

If the loop ending condition (ie condition) is left blank, it is assumed to be always true. This makes it possible to create endless loops. Such a loop can be interrupted, for example by a **return** statement, which exits the function, or **break** statement.

It is possible to nest multiple **for** loops.

In c99 standard, it is possible to initialize variable inside a for loop itself like in below example.

```
#include <stdio.h>

int main(void)
{
    for (int a = 0; a < 10; a++) {
        printf("value of a is now %d\n", a);
    }
}
```

Variable initialized in a for loop statement can be used only inside **for** loop.

break and continue¶

The **break** statement can be used to terminate a loop before the specified condition is evaluated. For example, the below example never makes it to 10, but stops counting at 5:

```
#include <stdio.h>

int main(void)
{
    int a;
    for (a = 0; a < 10; a++) {
        printf("value of a is now %d\n", a);
        if (a == 5)
            break;
    }
}
```

The **continue** statement causes the next iteration of a loop to start immediately. For example, the following code only shows the even numbers (a % 2 takes modulo 2 of variable **a**):

```
#include <stdio.h>

int main(void)
{
    int a;
    for (a = 0; a < 10; a++) {
        if (a % 2 == 1)
            continue;
        printf("value of a is now %d\n", a);
    }

    return 0;
}
```

Task 2.4: For¶

Modify the above program so that it will print even-numbers in reverse order:

```
value of a is now 8
value of a is now 6
value of a is now 4
```

and so up to zero.

Points `5 / 5` My submissions `1` Deadline Sunday, 13 June 2021, 19:59 To be submitted alone

This course has been archived (Saturday, 31 December 2022, 20:00).

For

Select your files for grading

main.c

Choose File No file chosen

Submit

Task 2.5: Triangle¶

In this task, students learn how to implement nested loops to produce below mentioned triangle prints.

Implement function **void draw_triangle(unsigned int size)** that draws an ASCII box that contains a triangle.

The box should be **size** characters wide, and **size** characters tall. The box is split diagonally in two such that the righthand and bottom characters are `'#'`, and the lefthand and top characters are `' '`.

The first line contains one `'#'` character at the right edge, the second line contains two `'##'` characters, and so on. On the last line all characters are `'#'`.

All lines (also the last) end with a newline character `{'\n'}`.

Here is an example calling draw_triangle(5):

```
....#
...##
..###
.####
#####
```

To test **draw_triangle(unsigned int size)** function, for example, you should write a main function that asks user for an input unsigned integer (`%u`), and calls **draw_triangle** function using the given input, and repeat this loop until the program is interrupted (using command line Ctrl + C key combination).

Points `20 / 20` My submissions `4` Deadline Sunday, 13 June 2021, 19:59 To be submitted alone

This course has been archived (Saturday, 31 December 2022, 20:00).

Triangle

Select your files for grading

source.c

Choose File No file chosen

Submit

Task 2.6: ASCII¶

This task focuses on the ASCII table, helping to understand how the different encoding works. At the same time become familiar with the various printf formats, as well as provide more practice on loops

Implement function **void ascii_chart(char min, char max)** that outputs (a portion of) ASCII character mapping. It should iterate through numbers starting from 'min' and ending to (and including) 'max'.

- three-character field that shows the given number as an integer. If the number takes less than three characters (it is < 100), it is aligned right.
- one space, followed by four-character field that shows the same number in hexadecimal format. Each hexadecimal number should take two characters, and one-digit numbers are prefixed with 0. The whole hexadecimal number is prefixed with '0x'. For example, number 1 is shown as '0x01'.
- one space, followed by the same number when printed in character format (always one-character field). The number is converted into a character according to ASCII coding system.

  Some character codes are not "printable", and do not produce sensible output with this formatting. For non-printable characters, just '?' should be shown. You can use function `int isprint(int c)` (`man page`) to test if character in variable 'c' is printable. If function returns 0, the character is not printable and should show as '?'. If it is non-zero the character should be printed normally.
- one tab `{'\t'}`, if the current line has less than four entries printed. On the fourth entry, you should change to the next line, i.e., instead of tab, print newline `{'\n'}`.

You should cycle through each number in the parameter range in the above-mentioned format (also including the max value). For example, call ascii_chart(28,38), should show the following:

```
28 0x1c ?       29 0x1d ?       30 0x1e ?       31 0x1f ?
32 0x20         33 0x21 !       34 0x22 "       35 0x23 #
36 0x24 $       37 0x25 %       38 0x26 &
```

Points `20 / 20` My submissions `3` Deadline Sunday, 13 June 2021, 19:59 To be submitted alone

This course has been archived (Saturday, 31 December 2022, 20:00).

ASCII

Select your files for grading

source.c

Choose File No file chosen

Submit

« 2 Conditional statements Course materials 4 Round feedback »