**Lecture 2:**

int main(void)

{

return 0;

}

#include <stdio.h>

int main(void)

{

printf("Hello world!");

return 0;

}

#include <stdio.h>

int add(int n1, int n2)

{

int result = n1 + n2;

return result;

}

int main(void)

{

int first;

int second;

int sum;

printf("Input the first number: ");

scanf("%d", &first);

printf("Input the second number: ");

scanf("%d", &second);

sum = add(first, second);

printf("The sum of the numbers: %d\n", sum);

return 0;

}

**Lecture 3:**

#include <stdio.h>

int add(int n1, int n2)

{

int result = n1 + n2;

return result;

}

int main(void)

{

int first;

int second;

int sum;

printf("Input the first number: ");

scanf("%d", &first);

printf("Input the second number: ");

scanf("%d", &second);

sum = add(first, second);

printf("The sum of the numbers: %d\n", sum);

return 0;

}

**Lecture 5:**

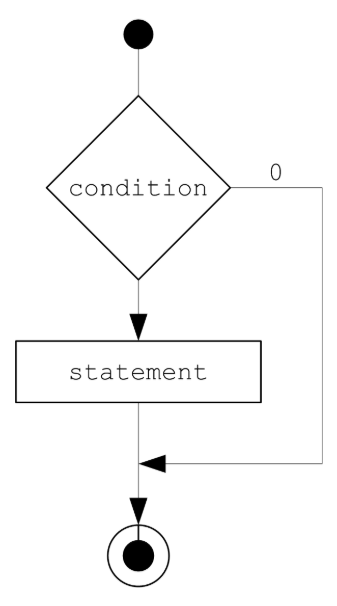


Figure 1. If Statement

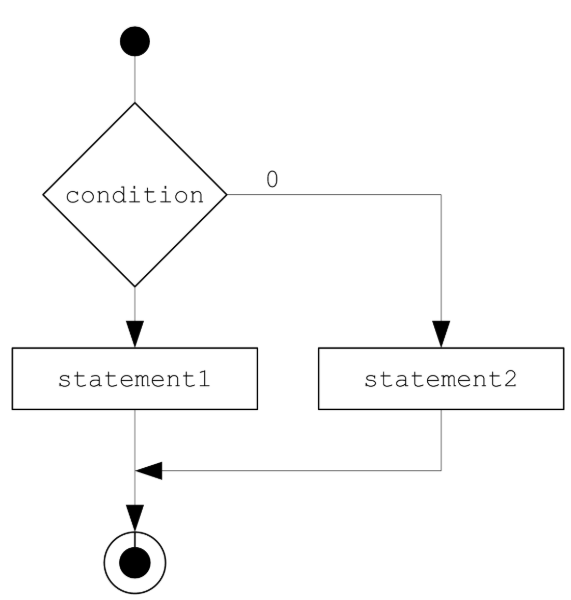


Figure 2. If Else statement

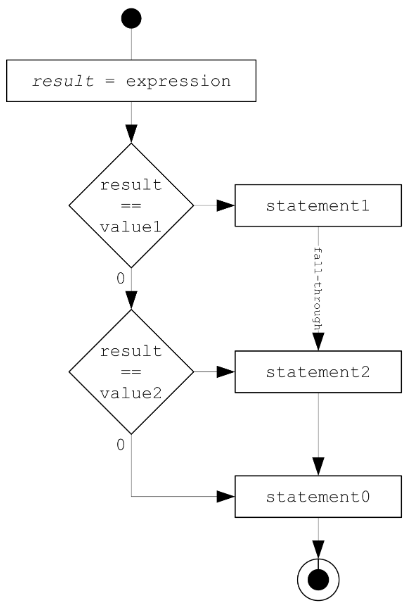


Figure 3. Switch statement

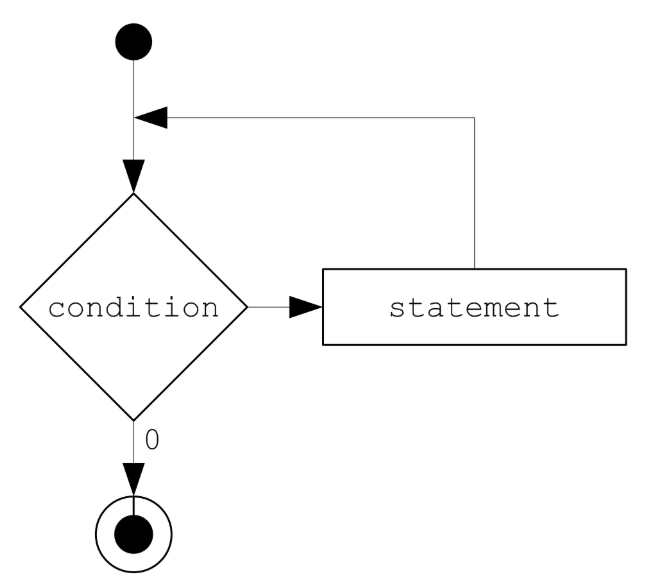


Figure 4. While loop

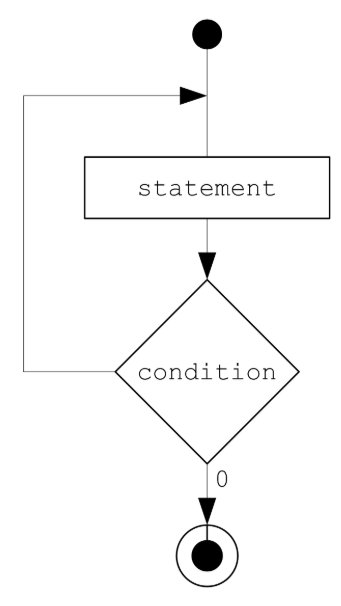


Figure 5. Do While loop

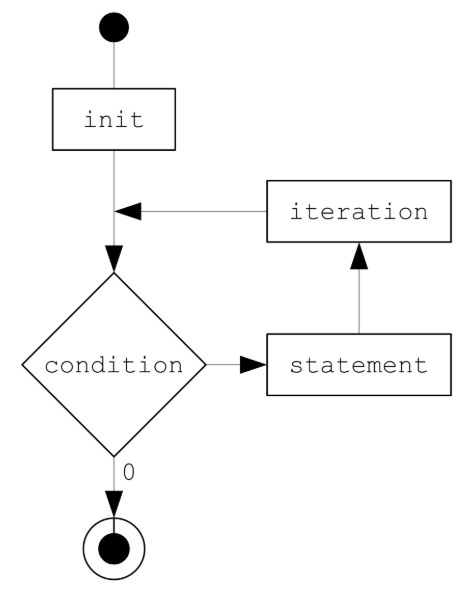


Figure 6. For loop

**Lecture 7:**

Comparision:

#include <stdio.h>

int main(void)

{

int a = 1;

/\*

check for equality:

"a == 3" returns 0 (false).

\*/

if (a == 3)

{

printf("Values are equal!\n");

}

else

{

printf("Values are different!\n");

}

/\*

assignment:

"a = 5" returns 5; a has been set to 5, the expresion returns 5 (true).

As 5 is always different than 0, the positive branch would be always

selected. Many modern compilers (including gcc) require you to wrap

an assignment in additional brackets, when used as a condition to show

your intent: if ((a = 5)) statement; as shown below.

\*/

if ((a = 5))

{

printf("Values are equal!\n");

}

else

{

printf("Values are different!\n");

}

/\* the following would not compile if mistaken for assignment \*/

/\*

5 is a literal, and thus it is not an "lvalue" (a locator value; a value

that can be located in writable memory) and therefore it is a "rvalue".

On the left-side of the assignment operator only an lvalue is allowed,

and so a mistake with the operator will result in a compilation error.

Therefore, some programmers prefer this style.

\*/

if (7 == a)

{

printf("Values are equal!\n");

}

else

{

printf("Values are different!\n");

}

return 0;

}

Short-circuiting:

#include <stdio.h>

/\*

Let's assume our "special" numbers are numbers equal to or greater than 100,

and at the same time divisible by 5.

Test this program with numbers: 99, 100, and 101.

\*/

int isDivisibleBy5(int value)

{

int result;

result = (value % 5 == 0);

/\* side effects \*/

if (result)

{

printf("Divisible by 5.\n");

}

else

{

printf("Not divisible by 5!\n");

}

return (value % 5 == 0);

}

int main(void)

{

int number;

printf("Input a number: ");

scanf("%d", &number);

/\*

Short-circuit evaluation: if number is 99 or less, there is no need to

check if the number is divisible by 5; either way it is not going to

be special.

false && anything => false

\*/

if ((number > 99) && (isDivisibleBy5(number)))

{

printf("The number is special.");

}

else

{

printf("The number is not special.");

}

return 0;

}

#include <stdio.h>

/\*

Let's assume our "special" numbers are numbers equal to or greater than 100,

or divisible by 5, or both.

Test this program with numbers: 95, 99, 100, and 105.

\*/

int isDivisibleBy5(int value)

{

int result;

result = (value % 5 == 0);

/\* side effects \*/

if (result)

{

printf("Divisible by 5.\n");

}

else

{

printf("Not divisible by 5!\n");

}

return (value % 5 == 0);

}

int main(void)

{

int number;

printf("Input a number: ");

scanf("%d", &number);

/\*

Short-circuit evaluation: if number is 99 or more, there is no need to

check if the number is divisible by 5; either way it is going to be

special.

true || anything => true

\*/

if ((number > 99) || (isDivisibleBy5(number)))

{

printf("The number is special.");

}

else

{

printf("The number is not special.");

}

return 0;

}

#include <stdio.h>

/\* Function declaration: announces without allocating memory \*/

/\* In declaration we can omit parameter names \*/

int add(int, int);

int main(void)

{

int a;

int b;

int result;

printf("Enter first number: ");

scanf("%d", &a);

printf("Enter second number: ");

scanf("%d", &b);

/\* Actual parameters of add() function call: a and b. \*/

result = add(a, b);

printf("Sum: %d", result);

return 0;

}

/\* Function definition: \*/

/\* Formal parameters of add() function call: x and y. \*/

int add(int x, int y)

{

return x + y;

}

**Lecture 8:**

/\*

This program demonstrates how to request a single positive integer number

from the user and handle incorrect input.

\*/

#include <stdio.h>

int main(void)

{

int input;

printf("Enter a positive number: ");

if (scanf("%d", &input) != 1)

{

printf("You should have input a positive number.");

}

else if (input > 0)

{

printf("Good job!");

}

else

{

printf("You have failed me.");

}

/\* Compare with an implementation using the opposite condition. \*/

/\*

if (scanf("%d", &input) == 1)

{

if (input > 0)

{

printf("Good job!");

}

else

{

printf("You have failed me.");

}

}

else

{

printf("You should have input a positive number.");

}

\*/

return 0;

}

/\*

This program demonstrates how to display a particular number of stars

based on a positive number input by the user.

\*/

#include <stdio.h>

int main(void)

{

int counter;

int input;

printf("Enter a positive number: ");

if ((scanf("%d", &input) == 1) && (input > 0))

{

/\*

This is a "for" loop:

- for each value of counter

- from 0

- as long as it is less than input

- execute the statement

- and (pre)increment counter by 1

This loop uses an additional variable "counter" to keep track of

how many more iterations must be performed, and leaves "input"

variable untouched.

\*/

for (counter = 0; counter < input; ++counter)

{

printf("\*");

}

printf("\n");

counter = input;

/\*

This is a "for" loop too; each expression can be omitted:

- for each iteration

- as long as input is non-zero

- execute the statement

- and (pre)decrement counter by 1

This loop does not use an additional variable, and so it alters

the value of "input" to keep track of iterations. Since we have

one more loop to demonstrate, we have copied "input" into "counter".

\*/

for (;input;--input)

{

printf("\*");

}

printf("\n");

/\*

This is a while loop:

- while counter is non-zero

- execute the statement

- and (pre)decrement couter

\*/

while (counter)

{

printf("\*");

--counter;

}

}

else

{

printf("You have failed me.");

}

return 0;

}

#include <stdio.h>

int main(void)

{

unsigned int n;

unsigned int rowIndex;

unsigned int colIndex;

unsigned int lvlIndex;

printf("Please input a number from 1 to 4: ");

if ((scanf("%u", &n) == 1) && (n >= 1) && (n <= 4))

{

for (lvlIndex = 0; lvlIndex < n; ++lvlIndex)

{

for (rowIndex = 0; rowIndex < n; ++rowIndex)

{

printf("\*");

colIndex = 0;

while (colIndex < rowIndex)

{

printf("\*\*");

++colIndex;

}

printf("\n");

}

}

}

return 0;

}

/\*

This program demonstrated a mix of various types of statements working

together to solve a particular business problem: finding square roots

by narrowing the search range.

\*/

#include <stdio.h>

float square\_root(float n)

{

const float EPSILON = 1e-5f;

float a; /\* lower boundary of the search range \*/

float b; /\* upper boundary of the search range \*/

float c; /\* midpoint of the range \*/

float delta; /\* difference between midpoint squared and expected value \*/

a = 1;

b = n;

/\*

Infinite loop

There is a break (jump statement) that takes us out of the loop.

\*/

while (1)

{

c = (a + b) / 2;

delta = c \* c - n;

/\* Absolute value \*/

if (delta < 0)

{

delta = -delta;

}

if (delta < EPSILON)

{

/\* Found the result if it is within an error margin. \*/

break;

}

else if (c \* c < n)

{

/\*

Midpoint squared not large enough.

Continue search in the upper half of the range.

\*/

a = c;

}

else

{

/\*

Midpoint squared too large.

Continue search in the lower half of the range.

\*/

b = c;

}

}

return c;

}

int main(void)

{

float n;

printf("Input a number greater or equal to 1: ");

if ((scanf("%f", &n) != 1) || (n < 1))

{

printf("\nWrong input\n");

}

else

{

printf("\n%f\n", square\_root(n));

}

return 0;

}