Programming for Beginners

Lecture 3: control structures

Kai Lampka

Uppsala University

kai.lampka@it.uu.se

control structures

Two Basic techniques for computing something: **Branching** and **Looping**

- The decision to branch or not to branch or to continue with a loop depends on the value of a given expression.
- As expression one may use whatsoever,
  everything is evaluated in C, commonly from right to left
  e.g., a = 10; evaluates to 10.
- expressions can be combined via logical operators
  AND, OR, XOR (&&, ||, ^)
- **Do not confuse:**
  - The execution of each operator of an expression gives a value, which serves only as intermediate results and is discarded immediately after use (rvalues).
  - Other values, however, are of permanent nature. They are either in memory or in a specially reserved registers of the processor saved. Such values are denoted as lvalues and required by operators which store a value or access the memory address of a value.

if-statement

The if statement can be used to test conditions so that we can alter the flow of a program.

Syntax

```
if (expression){ body of if-statement }
```

✧ The body can be a single statement, than you do not need the curly brackets

✧ the expresion can be anything which evaluates to 0 or unequal 0.

```
int mynumber;
scanf("%d",&mynumber);

if ( mynumber == 10){
    printf("is equal 10\n");
    printf("closing program\n");
}
```

The `& operator` s used above (`&mynumber)` says that we passing in the address where variable `mynumber` is stored.

if-then-else-statement

An if statement can be extended with an else-statement. In case the expression of the if-statement is 0, the body of the else is executed.

Syntax

```
if (expression){ body of if-statement }
else { body of if-statement }
```

- The body can be a single statement, than you do not need the curly brackets
- An else-statement always refers to the previous if-statement, curly brackets improve readability!

```
int mynumber = scanf("%d",&mynumber);

if ( mynumber == 10){
    printf("is equal 10\n");
    printf("closing program\n");
    return 0;
}
else printf("is not equal 10\n");
```

if-then-else-statement

```
const int MYONE 7;

int main(){
    //read inputs
    int mynumber;
    scanf("%d",&mynumber);

    // if my special number was given exit programme
    if ( mynumber == MYONE ){
        printf("Is equal\n");
        printf("Closing program\n");
        return 0;
    }
    else{ // print message and continue
        printf("Not equal\n");
        printf("Closing program\n");
    }
}
```

✧ Comments helps with the understanding and remembering of the functionality of the program, please use them

✧ The placement of the curly brackets and how the indentations are placed, this is all done to make reading easier and to make less mistakes in large programs.

nesting of if-statement

- ✧ You use an “if statement” in an “if statement” in an “if-statement”… it is called nesting.
- ✧ Nesting “if statements” can make a program very complex, but sometimes there is no other way.

```
#include<stdio.h>
int main(){
    int grade;
    scanf("%d",&passedAssignments);
    if (passedAssignments <= 3 )  {
        printf("YOU DID NOT STUDY.\n");
        printf(”TRY HARDER NEXT TIME ! \n");
    } //if closes
    else{
        if ( passedAssignments >= 5 ) {
            printf("YOU PASSED THE ASSIGNMENTS! \n");
            if ( passedAssignments == 6 )
                printf(”EXCELLENT JOB! \n");
            else
                printf(”WELL DONE! \n");
        } //if closes
    } // else closes
    return 0;} //main closes
```

if-then-else-statement (more)

`Elseif-statement.`

✧ Does not exists in C, instead one may sue else if { .. }.

✧ This works as the if-statement and its body is seen as single line.

```
if( expression1 )
   statement1;
else if(expression2 )
   statement2;
else if(expression3 )
   statement3;
   .
   .
else
   statementN;
```

✧ **An else-statement always refers to the previous if-statement, curly brackets improve readability!**

```
int p = 0
if(0)
if(1) p = 5;
else p = 1;
```

?-statement

if-then-else can be replaced with a single statement

Syntax

```
result = test-expression ? value1 : value2;
```

If `test-expression` evaluates to true result is assigned the value `value1`, otherwise result is assigned the value `value2`.

```
int mynumber = scanf("%d",&mynumber);

if ( mynumber == 10)
    printf("is 10\n");
else
    printf("is not 10\n");
```

```
int mynumber = scanf("%d",&mynumber), c;
c = (mynumber != 10) ? printf("is not 10\n") : printf("is 10\n");
```

```
// another example
int max;
max = (v1 > v2) ? v1 : v2;
```

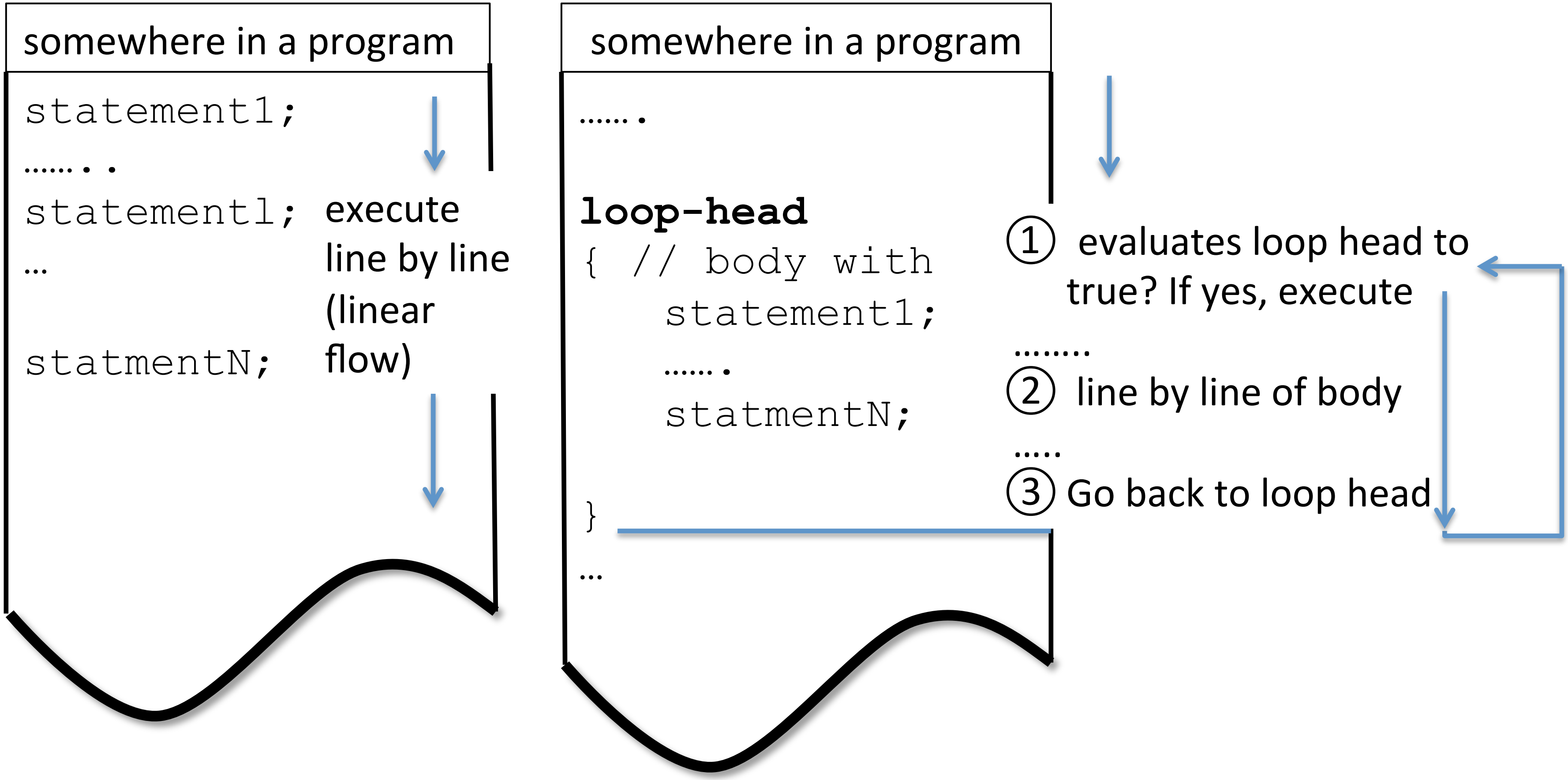
switch-statement

- The switch statement can have many conditions. You start the switch statement with a `switch-expression` which is evaluated.
- If one of the `case expressions` equals the value of the expression, the instructions are executed until a break is encountered.
- If none of the `case expressions` equals the switch expression the default is executed

```
int main() {
    char myinput;
    printf(“Choose: \t a) Program 1 \t b) Program 2\n");
    scanf("%c", &myinput);
    switch (myinput){ // variables are evaluated
        case 'a': //compare value to case-expression
            printf("Run program 1\n");
            break;
        case 'b’:
            printf("Run program 2\n");
            printf("Please Wait\n");
            break;
        default:
            printf("Invalid choice\n");
            break;
    }
return 0;}
```

The break-statements are required to exit the switch statement, **otherwise everything behind will be executed as well, until the next break or the end of the switch** *–no re-testing of the variable again!*

Loops



somewhere in a program

```
statement1;
……. .
statement1;
…

statmentN;
```

execute line by line (linear flow)

somewhere in a program

```
……. 

loop-head
{ // body with
    statement1;
    ……. 
    statmentN;

}
…
```

① evaluates loop head to true? If yes, execute

……..

② line by line of body

…..

③ Go back to loop head

for-Loop

```
for (pre-loop statement; loop-condition; post-loop statement)
{
    // loop body
} // brackets can be omitted for a single statement
```

- The pre-loop statement is executed before the first loop entry.
- The loop condition is the expression which tells us if we can enter (again).
- post-loop statement is executed after each loop iteration.

```
#include<stdio.h>

int main(){
    int i;
    for (i = 0; i < 10; i++){
      printf ("Hello World: %d\n”,i);
    }
    return 0;
}
```

Be aware of endless loops and that variables have always the intended value.

while-Loop

```
while (loop-condition)
{
    // loop body
} // brackets can be omitted for a single statement
```

- The loop condition is the expression which tells us if we can enter (again).
- No pre- and post-loop statements

```
#include<stdio.h>

int main(){
 int i, howmuch;
 scanf("%d", &howmuch);
 i = 0;
 while(i < howmuch) printf ("Hello World: %d\n”,++i);
 return 0;
}
```

Be aware of endless loops and that variables have always the intended value.

do-while-Loop

- The “do while loop” is almost the same as the while loop. But loop-condition is tested after the body!.
- The “do while loop” has the following form:

```
do
{
    // loop body
} // brackets can be omitted for a single statement
while (loop-condition);
```

```
#include<stdio.h>

int main(){
 int i, howmuch;
 scanf("%d", &howmuch);
 i = 0;
 do {
    printf ("Hello World: %d\n”,++i);
 } while(i < howmuch);
 return 0;
}
```

Be aware of endless loops and that variables have always the intended value.

pre-mature leave or re-entering of a loop

- ✧ With a break-statement the loop is left immediately
- ✧ with a continue-statement one directly jumps to the loop-head and tests the loop condition again (for and while loop)

```
#include<stdio.h>
int main(){
    int i;
    for (i = 0; i < 10; i++){
     if(!(i % 5)) continue;
     printf ("Hello World: %d\n”,i);
    }
    return 0;}
```

```
#include<stdio.h>
int main(){
 int i, p;
 printf (”\nGive a number to be tested for being prime\n”);
 scanf("%d", &p);
 for (int i = 2; i < p; i++){
  if (p % i == 0) break;
 }
 if (i == p) printf (”%d is a prime \n”,p);
 else printf (”%d is not a prime \n”,p);
 return 0;}
```

Recursion

Recursion is a special form of branching: a function calls itself with modified input parameters until a return-value has been computed. This give the following two ingredients:

✧ Re-invocation of itself, but with modified input parameters

✧ Test for ending recursion, test must include either one of the modified input parameters

```
#include<stdio.h>
int factorialRec(int n){
   if (n == 0) //termination testing
      return 1;
    else
      return(n * factorialRec(n-1)); // recursive call
    }
}

int main(){
   int n = 32,
   return(factorialRec(n));
}
```

Iteration

Instead of recursion one may use another scheme for successively computing an output. The function calls a helper function until the iteration criterion is satisfied. Notice: this is what you implement with a loop anyway.

```
#include<stdio.h>
int factorial(int n){
   int res = n;
   while (n >= 0)
      res *= factorialHelper(n--);
   return(res);
}
int factorialHelper(int k){
   if(k == 0)
      return 1;
   else
      return(k);
}
int main(){int n = 32; return(factorial(n));}

// This saves stack space as only one function factorialHelper
// is allocated at a time!!!
```