Lecture 4 - Control statements

The *if-else* statement

The if-else statement is used to carry out a logical test and then take one of two possible actions depending on the outcome of the test.

The *else* portion of the if-else statement is optional. Hence the simplest general form for of the statement is

```
if (expression)
   statement
```

while the general form of an if statement which includes the else clause is

```
if (expression)
   statement 1
else
   statement 2
```

Examples:

```
if (status == 'S') tax=0.20 * pay;
else tax = 0.14 * pay;

if (flag) {
   printf("account number: %d", accountno);
   credit=0;
}

if (circle) {
   scanf("%f", &radius);
   area=3.14159 * radius * radius;
   printf("Area of circle = %f", area);
}
else {
   scanf("%f %f", &length, %width);
   area=length*width;
   printf("Area of rectangle = %f", area);
}
```

It is possible to nest *if-else* statements within one another. Below are some examples of two-layer nesting:

```
if e1 s1
else if e2 s2

if e1 s1
else if e2 s2
     else s3

if e1 if e2 s1
      else s2
else s3

if e1 if e2 s1   equivalent to  if e1 {
      else s2                     if e2 s1 else s2
                                 }
```

The *while* and *for* statements

```
  while (expression)
    statement
```

Loop

```
for (exp1; exp2; exp3)
  statement
```

is equivalent to the following

```
exp1;
while (exp2){
  statement
  exp3;
}
```

Example:

```
#include <ctype.h>
/*  atoi:  converts  s to an integer number  */
int atoi(char s[])
{
  int i,n,sign;
  for (i=0; isspace(s[i]); i++)  /*  jump over   */
     ;                           /*  white characters  */
  sign=(s[i]=='-') ? -1:1;
  if (s[i] == '+' || s[i] == '-') /* jump over  */
    i++;                          /* sign of the number */
  for (n=0; isdigit(s[i]); i++)
    n=10*n+(s[i]-'0');
  return sign*n;
}
```

Inside loops `for` the comma operator `,` is often used:
Example:

```
#include <string.h>
/*  function reverses text  s  in place  */
void reverse(char s[])
{
   int c,i,j;
   for (i=0, j=strlen(s)-1; i<j; i++,j--) {
      c=s[i];
      s[i]=s[j];
      s[j]=c;
   }
}
```

Expressions separated by comma are evaluated from the left to the right-hand side, and the result type and value is equal to the type and value of the right-hand side argument.

Loop *do-while*

```
do
  statement
while (expression);
```

First the statement is executed, and afterwards the expression is evaluated. The loop is stopped when the *expression* becomes false.

Example:

```
#include <stdio.h>
void main()
{
   int i,n,sum=0;
   for (i=0; i<4; i++) {
      printf("Enter an integer number: ");
      scanf("%d", &n);
      sum+=n;
   }
   printf("Sum: %d\n",sum);
}
```

The same program with the use of the **while** loop:

```
#include <stdio.h>
void main()
{
   int i=0,n,sum=0;       /* Do not forget neither  */
   while (i<4) {          /* the initialization  */
      printf("Enter an integer number: ");
      scanf("%d", &n);
      sum+=n;            /* nor the incrementation  */
      i++;
   }
   printf("Sum: %d\n",sum);
}
```

The same program with the use of the **do**–**while** loop:

```
#include <stdio.h>
void main()
{
   int i=0,n,sum=0;      /* Do not forget neither  */
   do {                  /* the initialization     */
      printf("Podaj liczbe calkowita: ");
      scanf("%d", &n);
      sum+=n;
      i++;               /* nor the incrementation */
   } while (i<4);
   printf("Sum: %d\n",sum);
}
```

The *switch* statement

```
switch (expression) {
  case  constant-expression: statements
  case  constant-expression: statements
  default: statements
}
```

Example (program counts the number of encountered letters l, spaces and other characters):

```
#include <stdio.h>
void main()
{  int letterl=0,spaces=0,rest=0;
   while ((c=getch()) != EOF) {
      switch (c) {
         case 'l' : letterl++;
                    break;
         case ' ' : spaces++;
                    break;
         default:   rest++;
                    break;
      }
   }
   printf("%d  %d  %d\n", letterl, spaces, rest);
}
```

Statements *break* and *continue*

break – terminates immediately the most nested loop or *switch* statement, inside which it appears.

continue – bypasses the remainder of the current pass through a loop and forces the program to proceed from the beginning of the next step of the loop. For *while* and *do* statements it means the immediate check of the stopping expression and in the *for* loop it transfers the control to the incrementation part.

Examples:

```
#include <stdio.h>  /* We are counting the number */
char *Ref="H e l l o"; /* of spaces appearance  */
void main()
{  int Count;
   for (Count=0; *Ref; Ref++) {
      if (*Ref != ' ')
        continue;
      Count++;
   }
   printf("%d\n", Count);
}

-------Second example---------------
#define MAXL 1000
#include <stdio.h>
void main()  /* Remove the final spaces and tabs */
{  int n;    char line[MAXL];
   while ((n=getline(line, MAXL)>0) {
      while (--n >= 0)
         if (line[n] != ' '  &&
             line[n] != '\t' &&
             line[n] != '\n')
           break;
      line[n+1]='\0';
      printf("%s\n", line);
   }
}
```

The *goto* statement and labels

Most often used to leave deeply nested loops, e.g.

```
for ( ... )
   for ( ... ) {
      ...
      if (failed)
        goto err:  /* jump to the error service  */
   }
...
err:  /*  correct or print out the message  */
```