

Training Document for Comprehensive Automation Solutions

Totally Integrated Automation (T I A)

MODULE C2

High Level Language Programming with S7-SCL

This document has been written by Siemens AG for training purposes for the project entitled "Siemens Automation Cooperates with Education (SCE)".
Siemens AG accepts no responsibility for the correctness of the contents.

Transmission, use or reproduction of this document is only permitted within public training and educational facilities. Exceptions require the prior written approval by Siemens AG (Michael Knust michael.knust@siemens.com).

Offenders will be liable for damages. All rights, including the right to translate the document, are reserved, particularly if a patent is granted or utility model is registered.

We would like to thank the following: Michael Dziallas Engineering, the teachers at vocational schools, and all others who helped to prepare this document.

		PAGE
1.	Preface	5
2.	 Notes on the Development Environment S7 SCL	7
3.	 Installing the Software S7 SCL	8
4.	The Development Environment S7 SCL	9
5.	 Sample Exercise	11
6.	 Setting Up a STEP7 Project	12
7.	 Writing a STEP7 Program in S7 SCL	16
8.	 Testing the STEP7 Program in the CPU	23
8.1	Operating Modes – Test/Process Mode	23
8.2	Monitoring the STEP7 Program.....	24
8.2.1	Continuous Monitoring.....	24
8.2.2	Step by Step Monitoring.....	26

The following symbols are used as a guide through Module C2:



Information



Installation



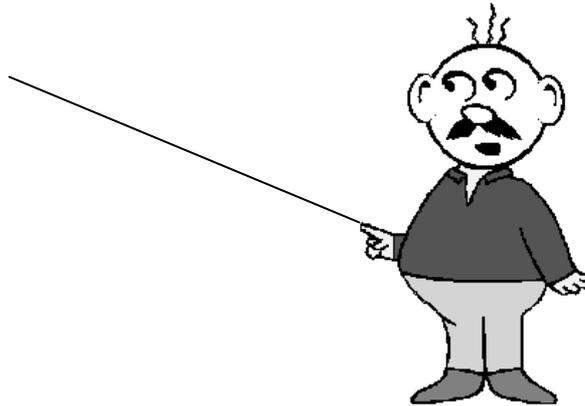
Programming



Sample Exercise

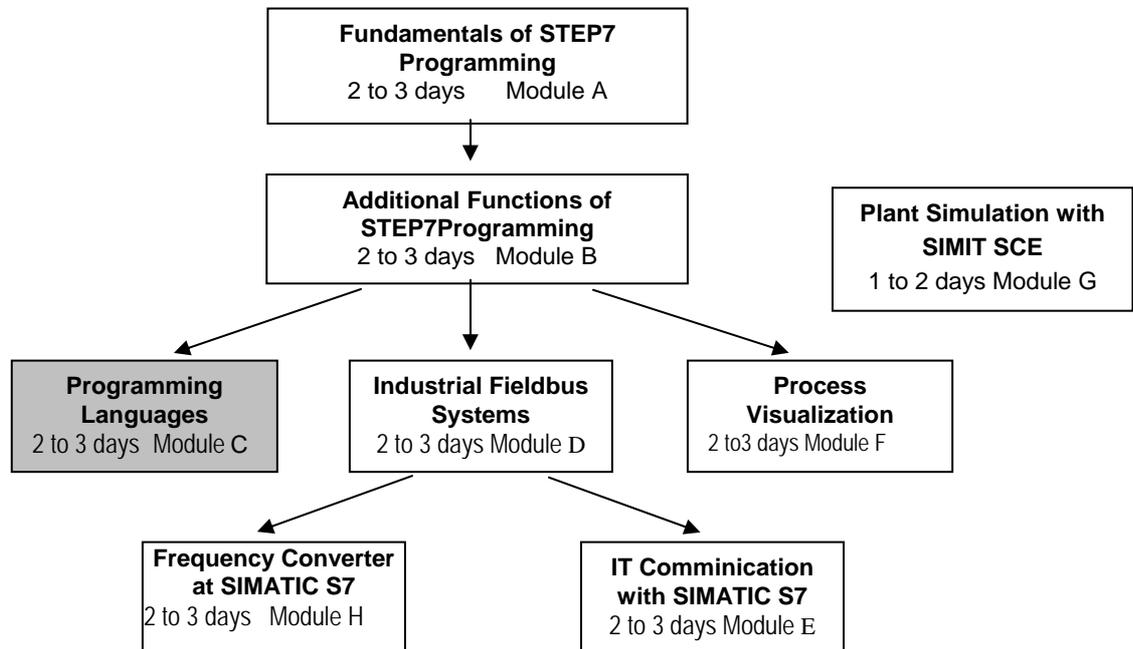


Notes



1. PREFACE

In terms of its contents, Module C2 is part of the teaching unit entitled '**Programming Languages**'.



Learning Objective:

In module C2, the reader is introduced to the basic functions of the S7-SCL development environment. In addition, test functions are discussed for removing logical programming errors.

Prerequisites:

To successfully work through Module C2, the following knowledge is assumed:

- Knowledge in handling Windows
- Fundamentals of PLC programming with STEP 7 (for example, Module A3 – 'Startup' PLC Programming with STEP 7)
- Basic knowledge of high level language programming, such as Pascal

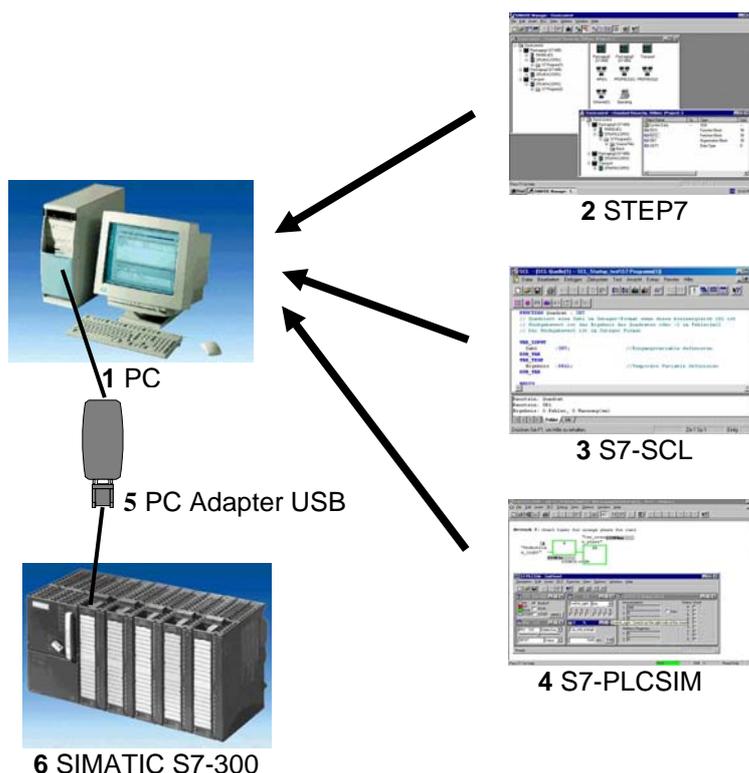
Preface	Notes	Installation	Development Environment	Sample Exercise	Project	Program	Testing
---------	-------	--------------	-------------------------	-----------------	---------	---------	---------

Hardware and software required

- 1 PC, operating system Windows 2000 Professional starting with SP4/XP Professional starting with SP1/Server 2003 with 600MHz and 512RAM, free hard disk storage 650 to 900 MB, MS Internet Explorer 6.0
- 2 Software STEP7 V 5.4
- 3 Software S7 SCL V5.x
- 4 Software S7 PLCSIM V5.x
- 4 MPI interface for the PC (for example, PC adapter USB)
- 5 PLC SIMATIC S7-300 with at least one digital input and output module. The inputs have to be taken to a panel

Sample configuration:

- Power supply: PS 307 2A
- CPU: CPU 314
- Digital inputs: DI 16xDC24V
- Digital outputs: DO 16xDC24V/0.5 A



Preface	Notes	Installation	Development Environment	Sample Exercise	Project	Program	Testing
---------	-------	--------------	-------------------------	-----------------	---------	---------	---------

2. NOTES REGARDING THE PROGRAMMING LANGUAGE S7-SCL



S7-SCL (Structured Control Language) is a higher level programming language that is based on PASCAL and makes structured programming possible. The language corresponds to the sequential function chart SFC specified in the standard DIN EN-61131-3 (IEC 1131-3). In addition to the higher level language elements, S7-SCL also includes typical PLC elements such as inputs, outputs, timers, flags, block calls, etc. as language elements. It supports the STEP7 block concept and allows for non-standard programming of blocks, in addition to STL, LAD, and FBD. That means, S7-SCL supplements and expands the programming software STEP7 and its programming languages LAD, FBD, and STL.

You do not have to generate each function yourself. You can resort to preassembled blocks such as system functions and system function blocks that exist in the operating system of the CPU.

Blocks that are programmed with S7-SCL can be mixed in with STL, LAD and FBD blocks. That means that a block that is programmed with S7-SCL can call another block that is programmed in STL, LAD or FBD. Correspondingly, S7-SCL blocks can be called in STL, LAD, and FBD programs.

When used in actual application cases, S7-SCL blocks can be recompiled into the STEP7 programming language STL. Recompiling into the S7-SCL is not possible. During compilation, those blocks are generated from the previously edited program that are runnable -starting with CPU314- on all CPUs of the S7-300/400 automation system.

The test functions of the S7-SCL allow for the search for logical programming errors in a faultless compilation. The errors are searched for in the source language.

Preface	Notes	Installation	Development Environment	Sample Exercise	Project	Program	Testing
---------	--------------	--------------	-------------------------	-----------------	---------	---------	---------

3. INSTALLING THE SOFTWARE S7-SCL



S7-SCL is an option package for STEP7. This means, it is assumed that STEP7 is already installed on your computer. (Refer to Module A2 – Installation of STEP7 V5.x/Handling the Authorization). S7-SCL is delivered on a CD-ROM that includes a diskette. The diskette contains the license key (authorization) that has to be transferred to the PC. It makes utilizing the S7-SCL possible. The authorization can be retransferred to the diskette, to be used on another PC. Starting with STEP7 Professional V5.3, this license can also be managed via a network. Regarding the topic Installation and Transfer of Authorizations, please also refer to Module A2 – Installation of STEP7 V5.x/Handling the Authorization.

Now, to install S7-SCL, do the following:

1. Place the S7-SCL CD in the CD-ROM drive.
2. The setup program is started automatically. If not, start it by double clicking on the file '→ **setup.exe**'.

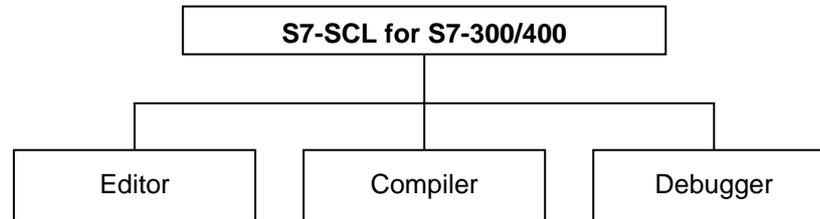
The setup program guides you through the entire installation of S7-SCL.

3. To utilize S7-SCL, a license key (authorization) -that is, a utilization authorization- is required on your computer. You have to transfer it from the authorization diskette to the computer. This is done at the end of the installation. There, you will be asked in a dialog window by the setup program whether you want to perform the authorization. If you select **YES**, you only have to insert the authorization diskette and the authorization is transferred to your computer.

4 THE DEVELOPMENT ENVIRONMENT S7-SCL



To use S7-SCL, a development environment is provided that is adjusted to the specific features of S7-SCL as well as of STEP7. This development environment consists of an Editor, a Compiler, and a Debugger.



Editor

The S7-SCL editor is a text editor with which any text can be edited. The central task that you will perform with it is generating and editing source files for STEP7 programs. In a source file, you can program one or several blocks. The syntax is not checked during the input.

The editor offers the following:

- Editing a complete source file with one or several blocks
- Editing a compilation control file. With it, compiling several source files can be automated
- Utilizing additional functions that make editing the source text possible; such as Search and Replace.
- Setting the editor according to your requirements; for example, through syntax-conforming coloration of the different language elements

Compiler

After you have generated your source files with the editor, they have to be compiled into S7 blocks.

The compiler offers the following:

- Compiling an S7-SCL source file with several blocks in one compilation run
- Compiling several S7-SCL source files by using the compilation control file that contains the names of the source files
- Selectively compiling individual blocks from one source
- Setting the compiler according to your requirements
- Displaying all errors and warnings that occur during compilation
- Localizing the faulty location in the source text, optionally with error description and information for error removal

Debugger

The S7-SCL debugger provides for checking a program during its run in the automation system, and locating possible logical errors.

S7-SCL offers two test modes for this:

- Step by step monitoring
- Continuous monitoring

For “step by step monitoring”, the logical program sequence is traced. You can execute the program algorithm instruction by instruction, and observe in a result window how the edited variable contents change.

With “continuous monitoring“ you can test a group of instructions within a block of the source file. During the test run, the values of the variables and the parameters are displayed in a chronological sequence and -to the extent possible- updated cyclically.

5. SAMPLE EXERCISE



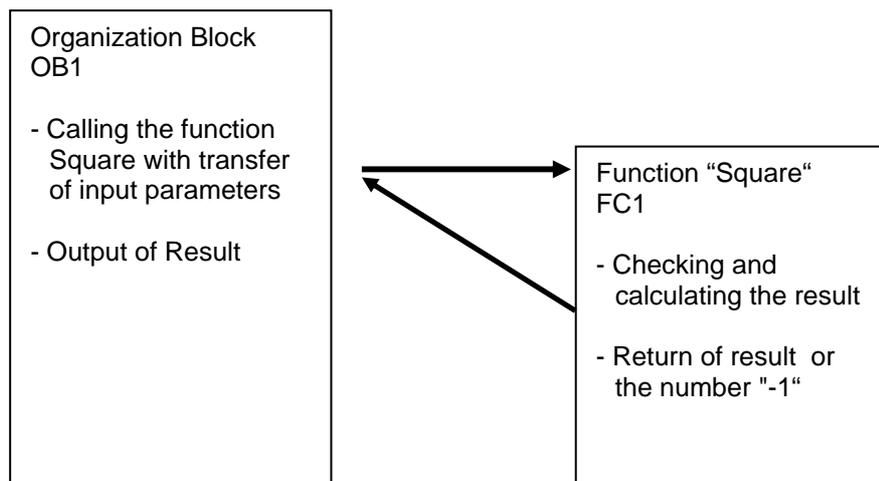
In this example, a natural number is to be squared. Before the result is calculated, a check is to be made whether the value of the number is larger or equal to 181. If this is the case, the arithmetic function can be executed, and the result is returned. If not, the return value is to be "-1".

The number to be squared is to be entered in the integer format by means of an input word. The output is made by means of an output word. The check and the calculation is to be programmed in a function.

Assignment List:

IW 0	Input	Number to be squared
OW 0	Output	Result
FC 1	Square	Square function

Program Structure:



Note: The value range of a number in integer format is between -32768 and +32767. The largest possible natural number whose square is in this range is 181. To avoid exceeding the range, a check is made prior to squaring.

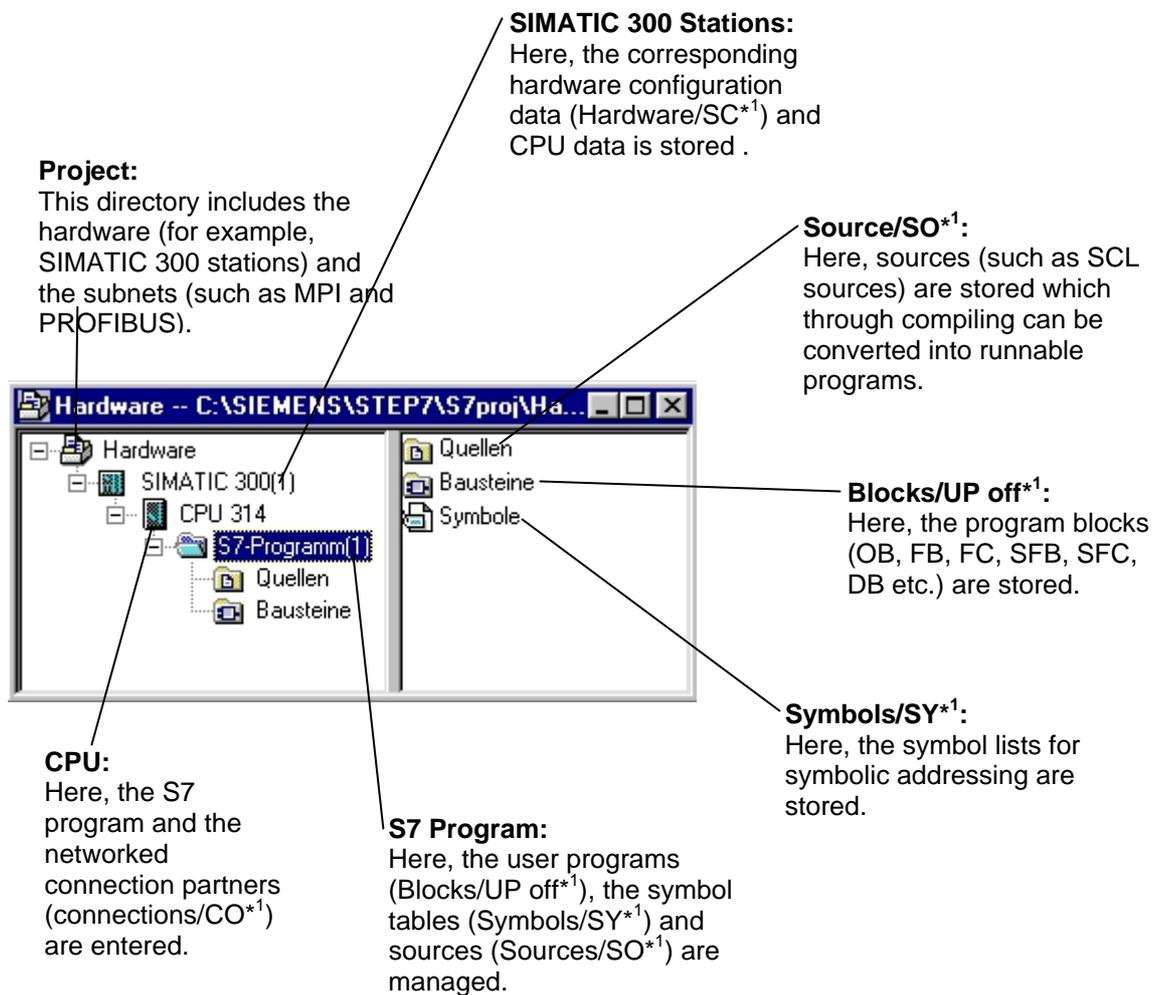
6. SETTING UP A STEP7 PROJECT



Files are managed with the 'SIMATIC Manager', as in STEP7. Here, program blocks can be copied, for example, or called for further editing with other tools by clicking on them with the mouse. The operator input corresponds to the standards that are customary for WINDOWS 95/98/2000/ME/NT4.0/XP. (For example, it is possible with one click of the right mouse key to get the selection menu for each component.)

The hardware configuration of the PLC is mapped in the folders 'SIMATIC 300 Station' and 'CPU'. Accordingly, such a project can always also be viewed specific to the hardware.

As in STEP7, each project is set up in a structure that is permanently specified. The programs are stored in the following directories:



*¹ Designations from STEP 7 Version 2.x.

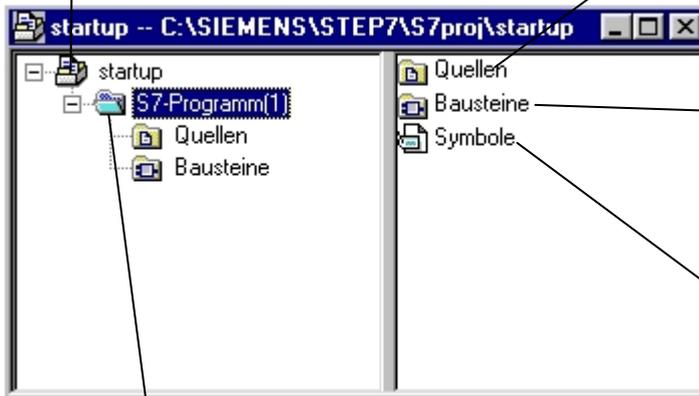


To generate a project regardless of the hardware configuration, it is possible to set up a project that does not include these folders.

It then has the following structure:

Project:

This directory includes the hardware (such as SIMATIC 300 stations) and the subnetworks (such as MPI and PROFIBUS).



Sources/SO*¹:

Here, sources (such as SCL sources) are stored that can be converted into runnable programs through compilation.

Blocks/UP off*¹:

Here, program blocks are stored (OB, FB, FC, SFB, SFC, DB etc.).

Symbols/SY*¹:

Here, the symbol lists for symbolic addressing are stored.

S7 Program:

Here, user programs (Blocks/UP off*¹), symbol tables (Symbols/SY*¹) and sources (sources/SO*¹) are managed.

*¹ Designations from STEP7 Version 2.x



Note: This example is generated without hardware configuration. Therefore, the programs can be loaded to any configuration of SIMATIC S7-300, S7-400 or WinAC. Only the addresses of the inputs and outputs have to be adapted from case to case.

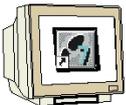


The user has to perform the following steps to generate a project where the solution program can be written.

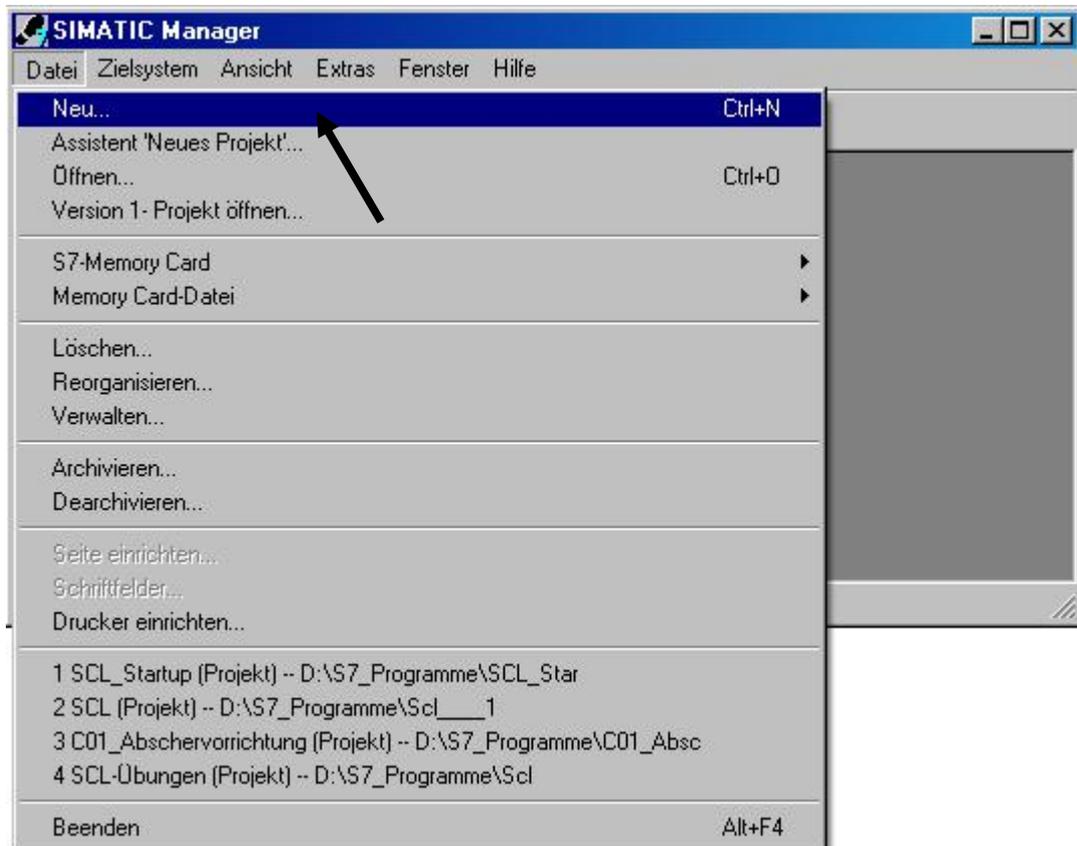
1. The central tool in STEP7 is the **SIMATIC Manager**. It is called here with a double click. (→ SIMATIC Manager)



SIMATIC Manager

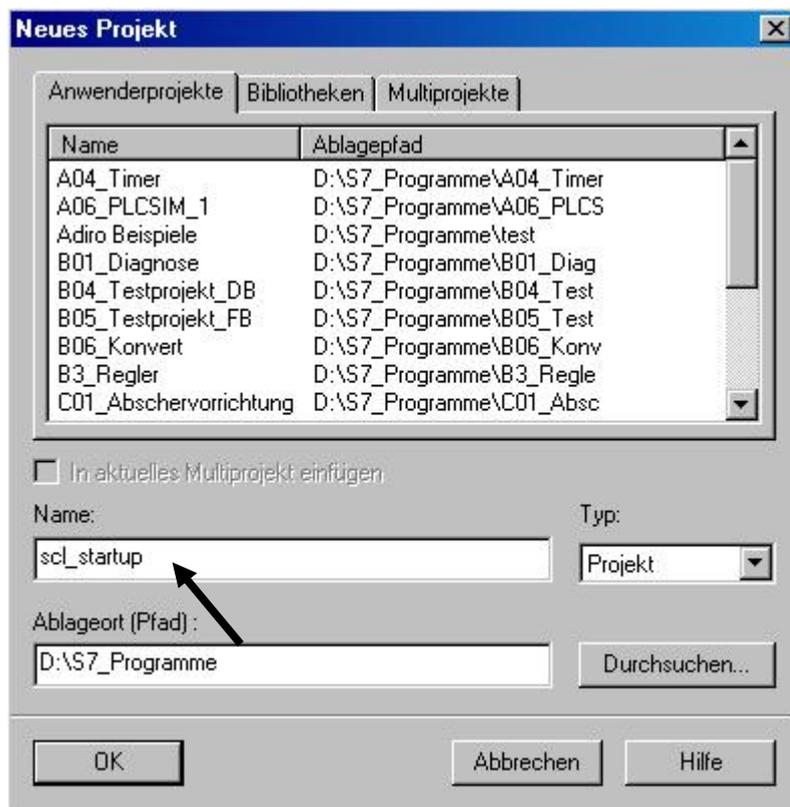


2. S7-SCL programs are managed in STEP7 projects. Such a project will now be set up (→ File → New)

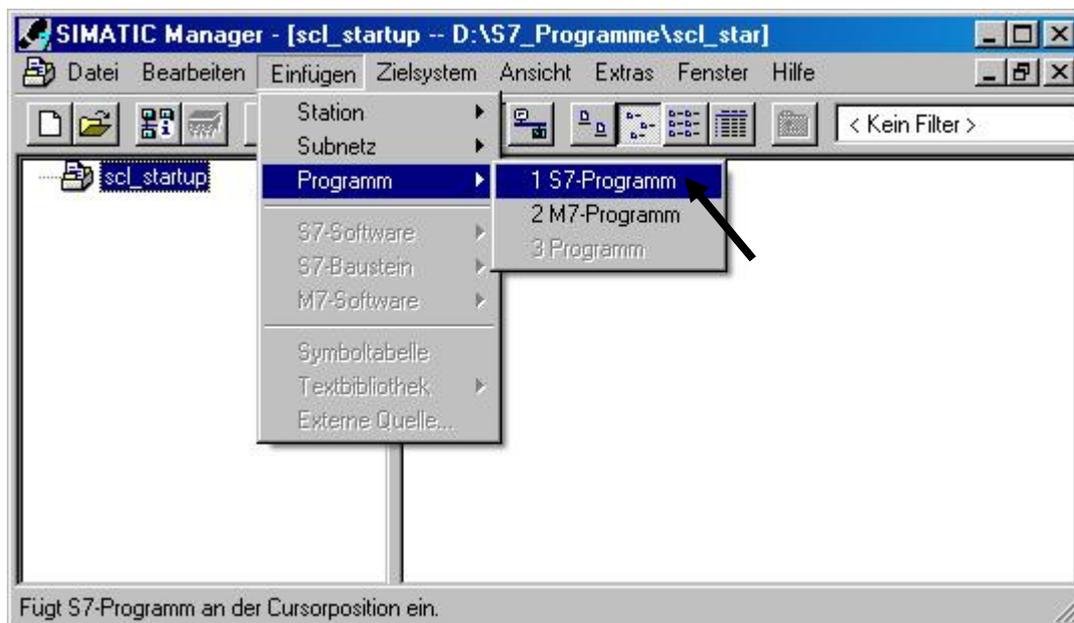




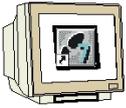
3. The project is now assigned the 'Name' 'scl_startup'. (→ scl_startup → OK)



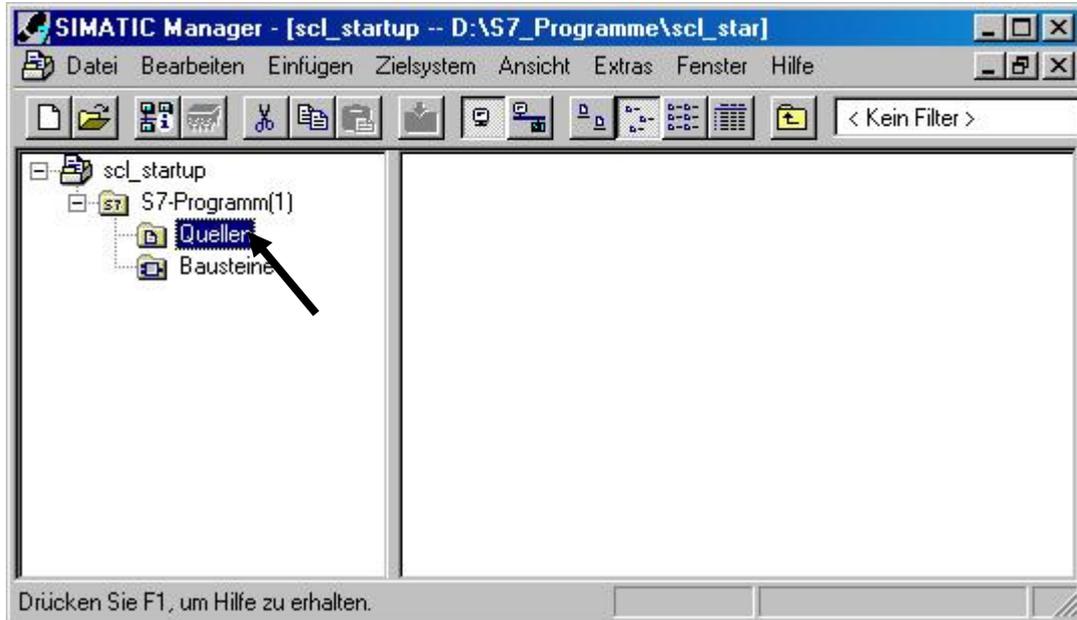
4. In the project 'scl_startup', a new 'S7 Program' is inserted. (→ scl_startup → Insert → Program → S7 Program)



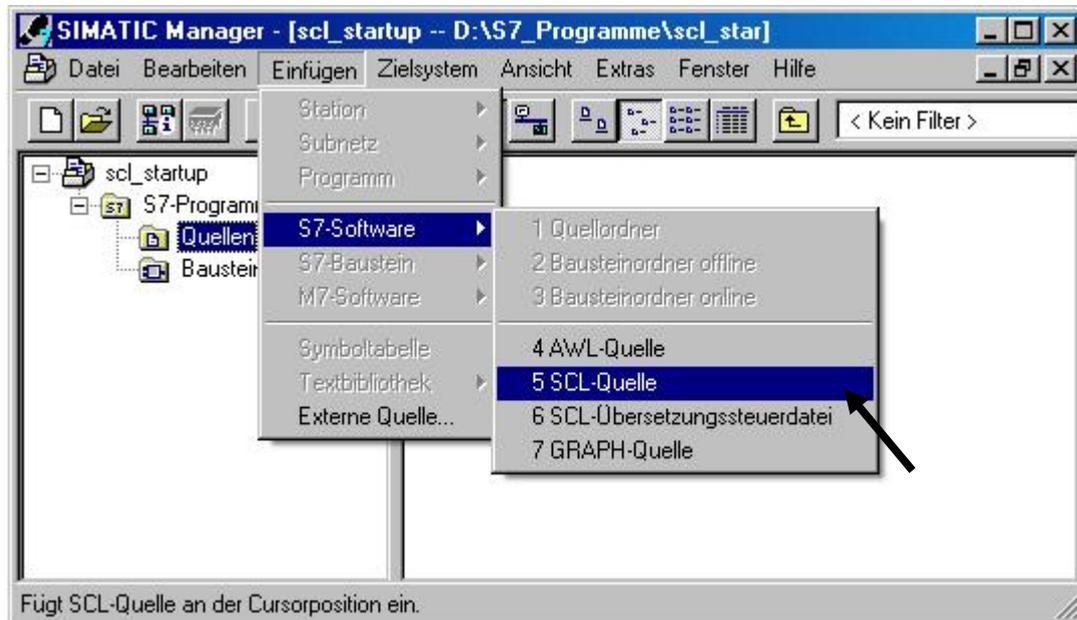
7. WRITING THE STEP 7 PROGRAM IN S7-SCL

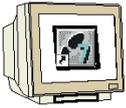


1. To insert an S7-SCL source, the folder 'Sources' has to be highlighted. (→ Sources)

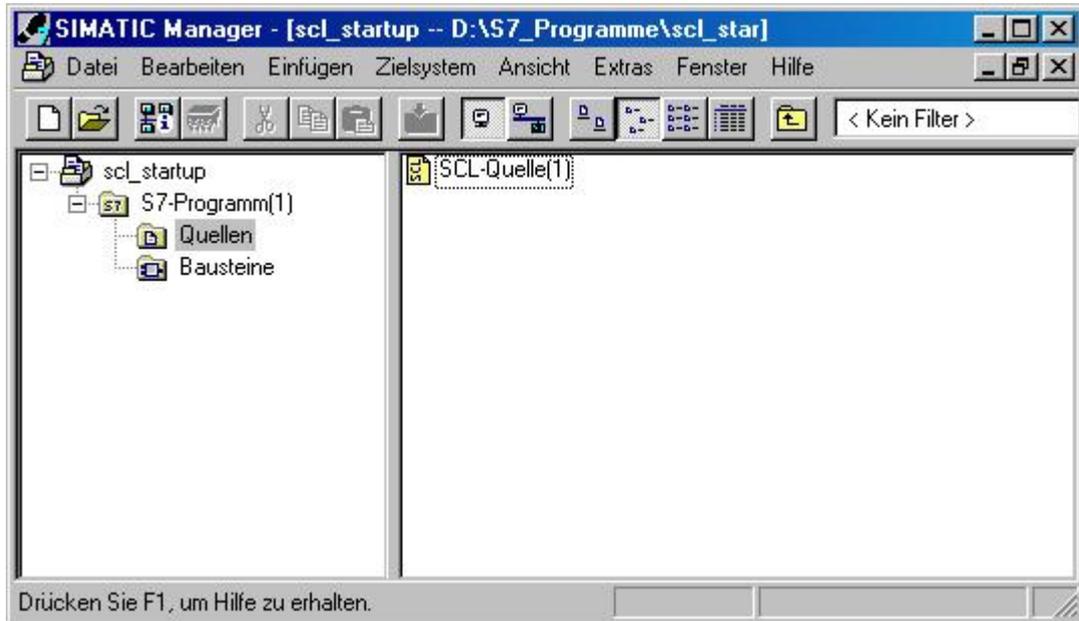


2. An 'SCL Source' is inserted. (→ Insert → S7 Software → SCL Source)





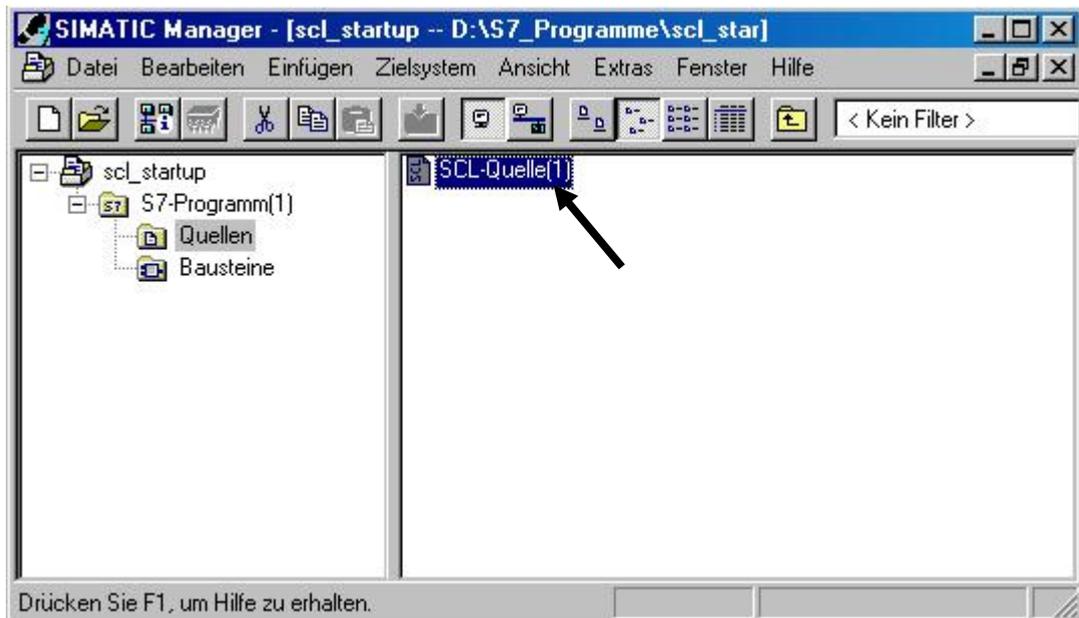
- In the SIMATIC Manager, an SCL source is now available for programming.



Note: You can change the name of the SCL source. This is advisable for better readability of the program in the case of a complex control task with several sources. To this end, click on the source with the right mouse key, and select '**Rename**' from the menu.

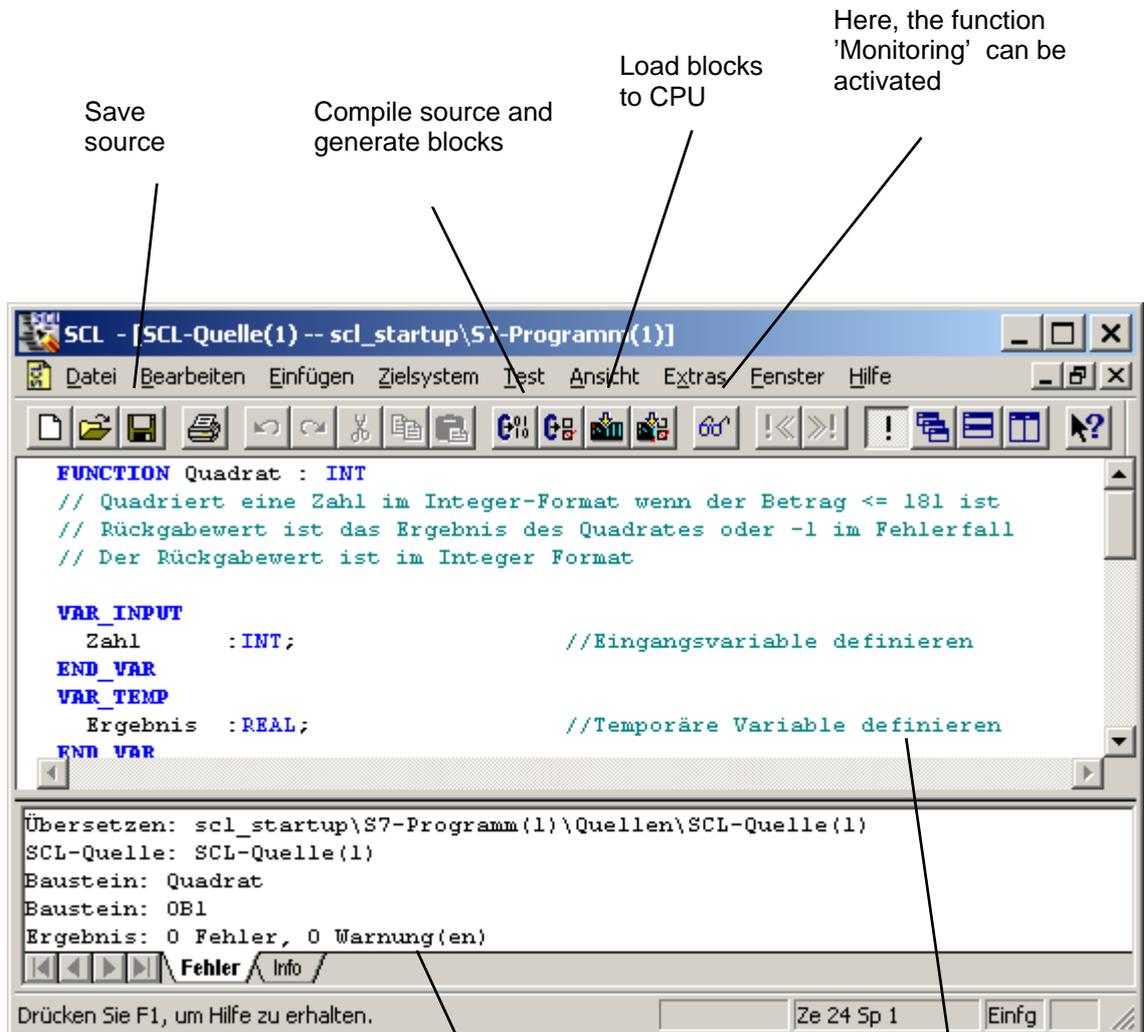


- Now, the SCL editor is to be started. To this end, the source in the SIMATIC Manager is opened with a double click. (→ SCL Source)





5. The programming interface for programming in SCL looks like this:



Save source

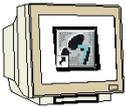
Compile source and generate blocks

Load blocks to CPU

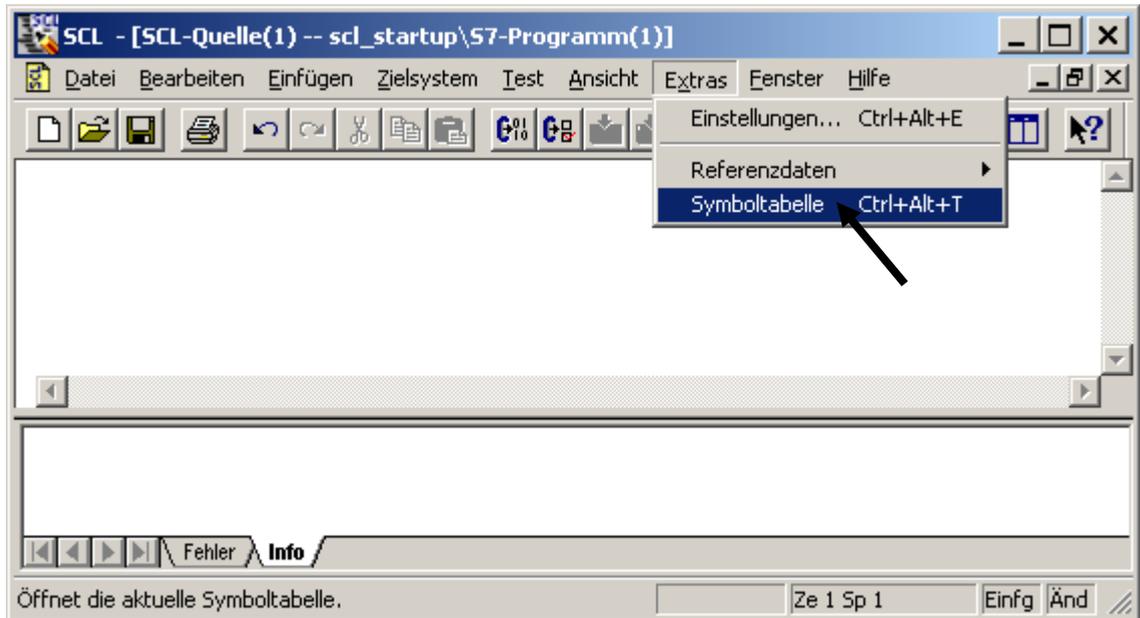
Here, the function 'Monitoring' can be activated

Here, errors and warnings are displayed that possibly occurred during compilation

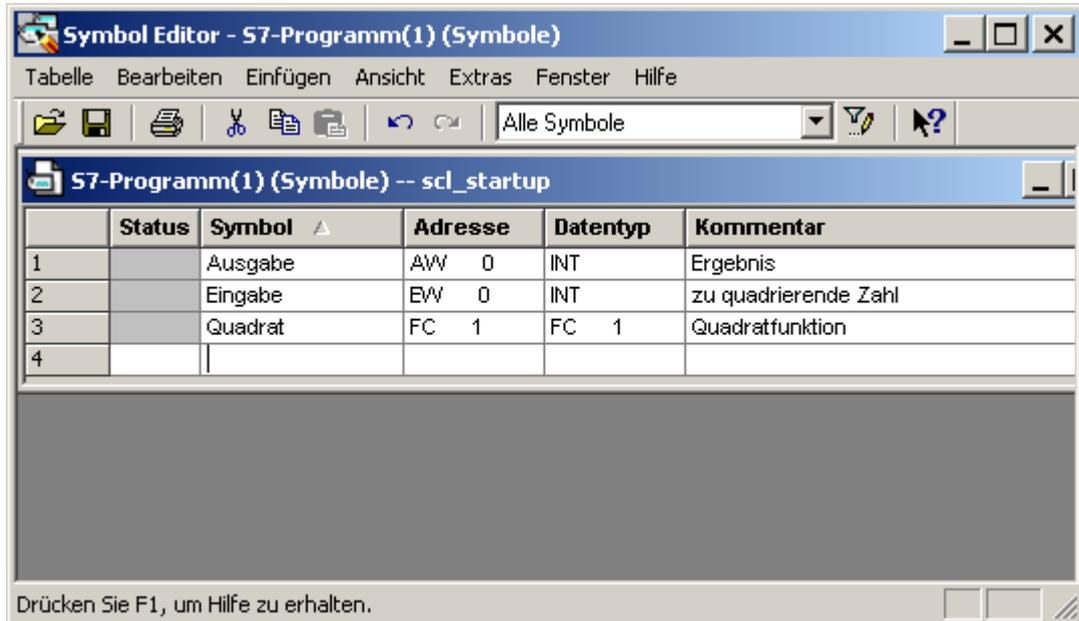
In this text editor, the control job can be generated by means of the source code in S7-SCL

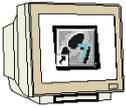


6. First, the symbol table is to be created. (→ Options → Symbol Table)



7. Enter operands in symbol table (→ Symbol → Address → Data Type → Comment) and save symbol lists (→ )





8. Now, the program can be entered in the editor's text field by using the symbolic names. For the example in S7-SCL, this could look like this:

```

FUNCTION Square : : INT
  // Squares a number in the integer format if the value is <= 181
  // The return is the result of the square or -1 if there is an error      1
  // The return value is in the integer format

VAR_INPUT
  Nbr      : INT;           //Define input variable
END_VAR
VAR_TEMP
  Result  :  : REAL;       //Define temporary variable      1
END_VAR

BEGIN
  IF ABS(Nbr) <= 181 THEN           //Check whether the value of number <= 181
    Result := SQR(Nbr);             //If yes, result is square of number
  ELSE
    Result := -1;                   //If no, result is -1
  END_IF;
  Square := REAL_TO_INT (Result); //Type conversion and return
END_FUNCTION

ORGANIZATION_BLOCK OB1
  //Organization block with call of a function block

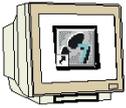
VAR_TEMP
  info : ARRAY[0..19] OF BYTE; //reserved
END_VAR

BEGIN
  Output := square (input);        //Calling the function Square
END_ORGANIZATION_BLOCK
    
```

Drücken Sie F1, um Hilfe zu erhalten. Ze 28 Sp 4 Einfg Änd



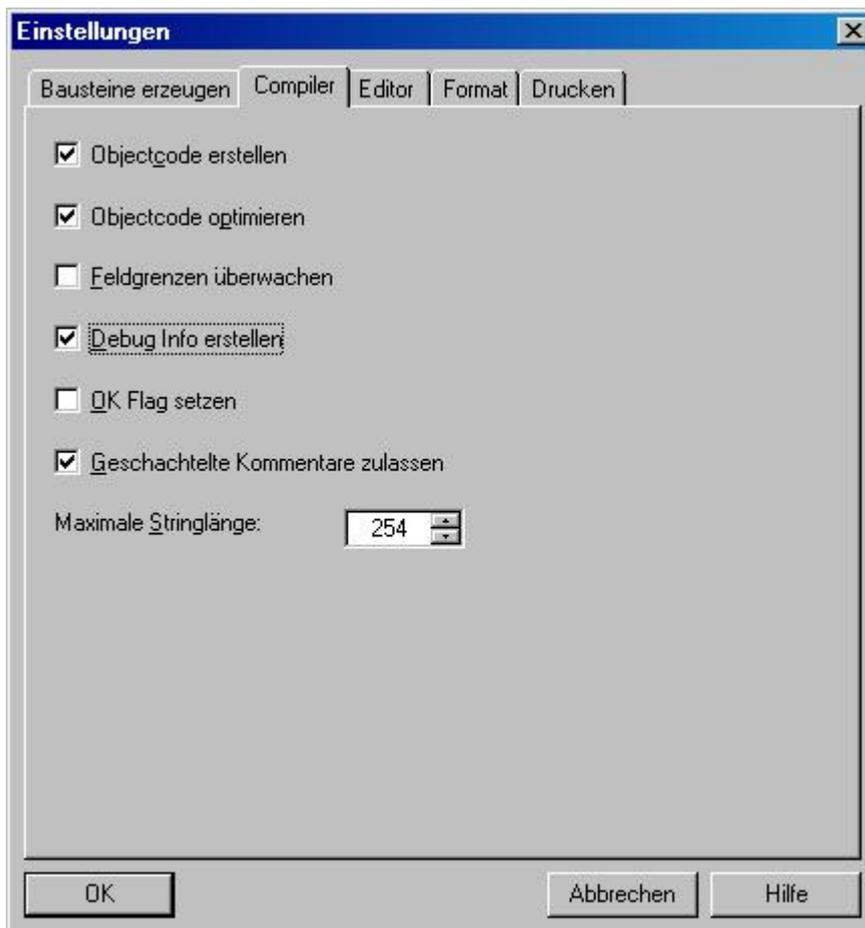
Note: It is possible to program all blocks of a source file. It should be noted in this case that the block that is called is always programmed before the block that is calling.

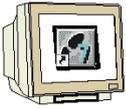


9. Before the source is stored, the compiler **'Settings'** have to be adjusted. (→ Options → Settings)

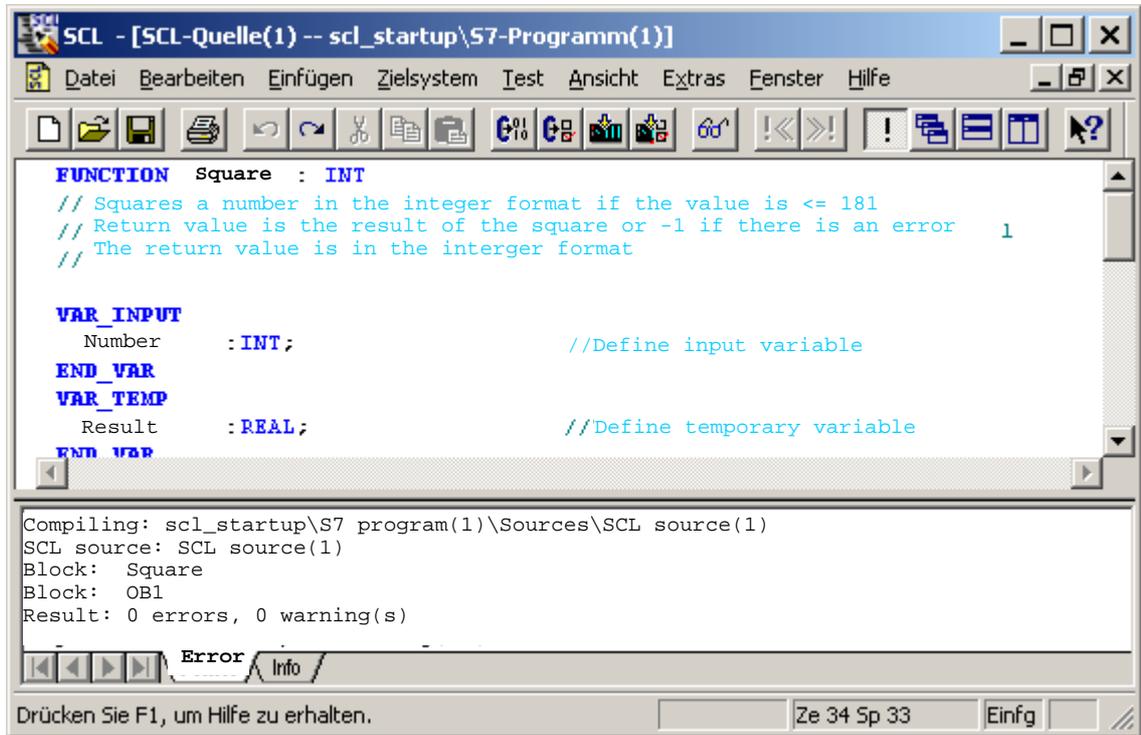


10. Under the tab **'Compiler'**, the following settings have to be made. (→ Compiler → OK)





11. Now, the program can be saved , compiled  and checked for syntax errors. After all errors are remedied, the blocks can be loaded to the CPU . (→  →  → )



Note: If several blocks are programmed in one source, it is possible to compile the blocks individually, () and to load them to the CPU individually. ()

8. TESTING THE STEP 7 PROGRAM IN THE CPU



The S7-SCL test functions allow for checking a program during its run in the CPU and find possible errors. Syntax errors are displayed during compilation. Timeouts during program execution are also indicated by system alarms. Logical programming errors can be determined with the test functions.

8.1 Operating Modes



Scanning test information usually extends the cycle time. In order to be able to influence this time extension, SCL offers two different operating modes.

Test Mode

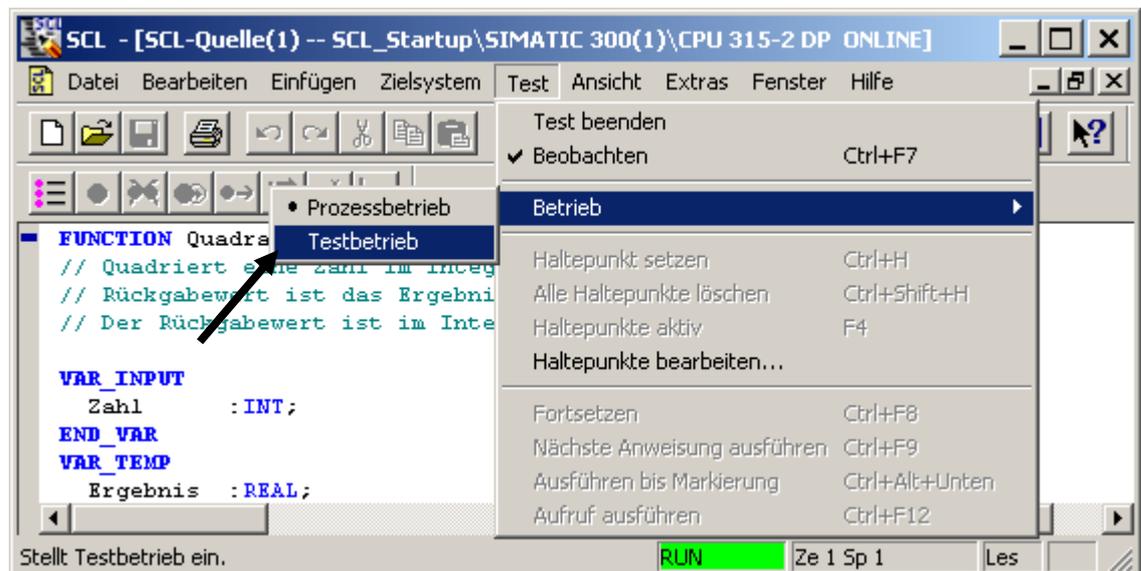
In the operating mode **'Test Mode'**, the monitoring range is limited only by the performance capability of the connected CPU. All test functions can be used without restriction. Longer extensions of the CPU cycle time can occur since, for example, the status of instructions in programmed loops is ascertained for each run.

Process Mode

In the operating mode **'Process Mode'**, the SCL debugger limits the maximum monitoring range in a way that the cycle duration during the test process does not exceed or only insignificantly exceeds the real run time.



1. The desired operating mode can be selected in the menu **'Test'**. The mode that is actually set is marked with a dot (→ Test → Mode → Test Mode)



8.2 Monitoring the STEP7 Program



The development environment S7-SCL makes two different test functions available.

Continuous monitoring

With this function, you can read out names and current values of variables in the SCL source. During the test run, the values of the variables and parameters of this area are displayed in a chronological sequence, and updated cyclically.

Step by step monitoring (only S7-400 CPUs)

With this function, you can set stop points, and then perform a test run in single steps. It is possible to execute the program algorithm instruction by instruction, for example, and monitor how the values of the processed variables change.

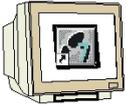
8.2.1 Continuous Monitoring



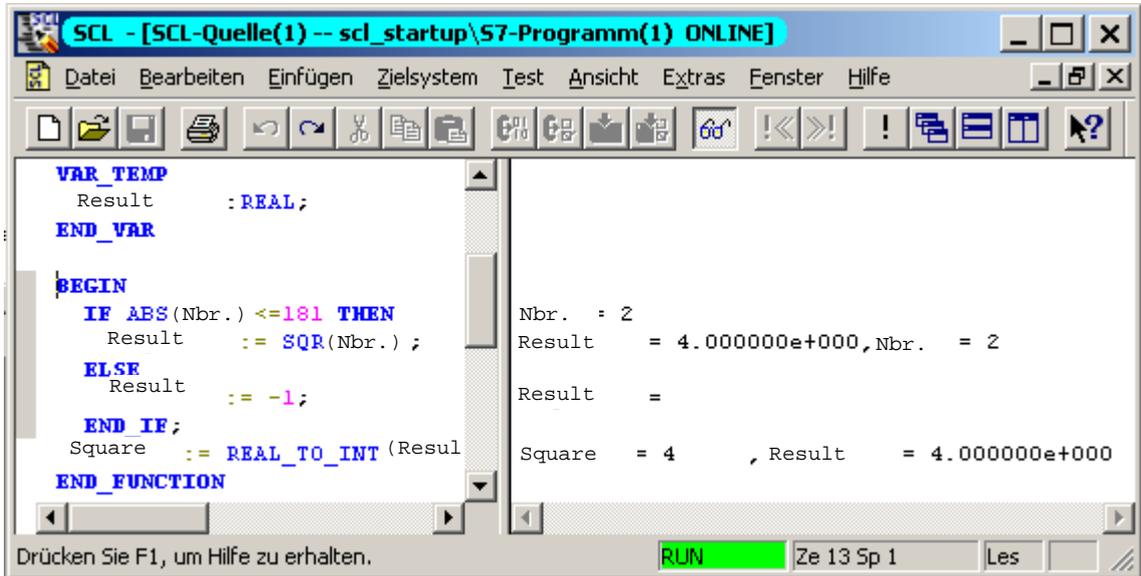
When continuously monitoring a program, you can test a group of instructions. This group of instructions is also called Monitoring Area. If the monitoring area is located in a program part that is passed in each cycle, the values of the variables can, as a rule, not be recorded from successive cycles.

Values that changed during the current run, and values that did not change can be differentiated by color.

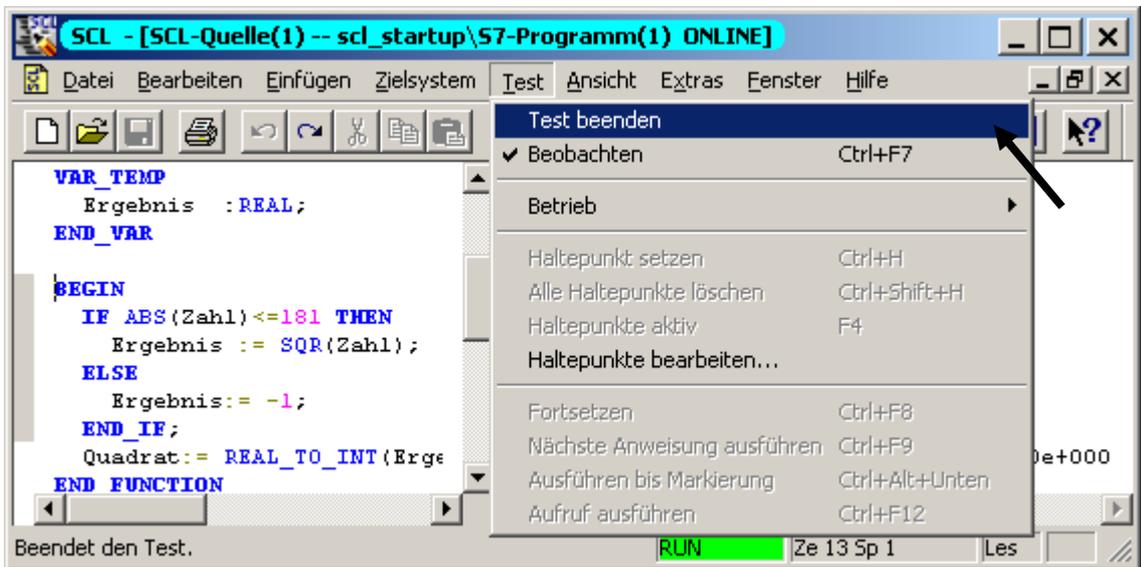
The scope of instructions to be tested is limited by the performance capacity of the connected CPU. Since the SCL instructions of the source code are mapped to a varying number of instructions in the machine code, the length of the monitoring area is variable. The SCL debugger ascertains and marks the length if you select the first instruction of the desired monitoring area.



1. By clicking on the eye glass symbol , the program can be monitored continuously. In the partial window on the left, the area that is monitored is shown with a gray bar. In the right window, the values are displayed. (→ )



3. The test function has to be closed in the menu 'Test'. (→ Test → Close Test)



Note: If you deactivate the function 'Monitoring' with the eye glass symbol , the split window is saved. The program continues to be open ONLINE and can not be edited. However, you can set, clear, or edit stop points, and monitor the program sequence step by step.

8.2.2 Step by Step Monitoring



When testing with stop points, the test is run in individual steps. The program algorithm can be executed instruction by instruction, and you can monitor how the values of the edited variables change. The number of stop points depends on the CPU that is connected.

After the stop points are set, you can have the program executed initially up to a stop point. When the CPU reaches this instruction, it enters the STOP mode, and from there, you can start with step by step monitoring.



Allows for activating/deactivating/clearing stop points that were set, and for specifying that certain stop points are active only in a defined call environment.



Sets or clears stop points at the current position of the mouse pointer. This menu command is active only if you are not performing the function "Monitoring" at the moment; that is, if the menu command Monitor is not marked with a check mark.



Clears all stop points of the current S7 program. This menu command is active only if you are not performing the function "Monitoring" at the moment; that is, if the menu command Monitor is not marked with a check mark.



Turns on the test mode "Testing with stop points". This menu command is active only if you are not performing the function "Monitoring" at the moment; that is, if the menu command Monitor is not marked with a check mark.



Executes the program up to the next stop point. When this stop point is reached, the CPU enters the STOP mode.



Executes the SCL instruction at the current position. After the instruction is executed, the CPU enters the STOP mode, the insertion point is moved to the next instruction, and the contents of the variables in the instruction that was edited last are displayed.



The program is executed up to the current position of the insertion point. The CPU enters the STOP mode.



If the block that is called is a block generated by SCL, the menu command Execute Call causes the block that is called to be opened, and the program stops at the first instruction of the block. You now can monitor the block step by step with the single step functions. When the end of the block is reached, the program jumps back to the block that is calling, and stops in the line following the block call. If the block that is called is a block that is not generated by SCL, the call is skipped and the program line that follows is marked.



Sample Exercise: Expanding and Testing

To better illustrate the different possibilities in the test mode, the sample exercise is supplemented.

Three different integers are to be squared and read out. To this end, the function Square in OB1 is called three times in succession. At each call, the input parameters are to be adapted to the respective input word. The output of the different results is to be placed on different output words.

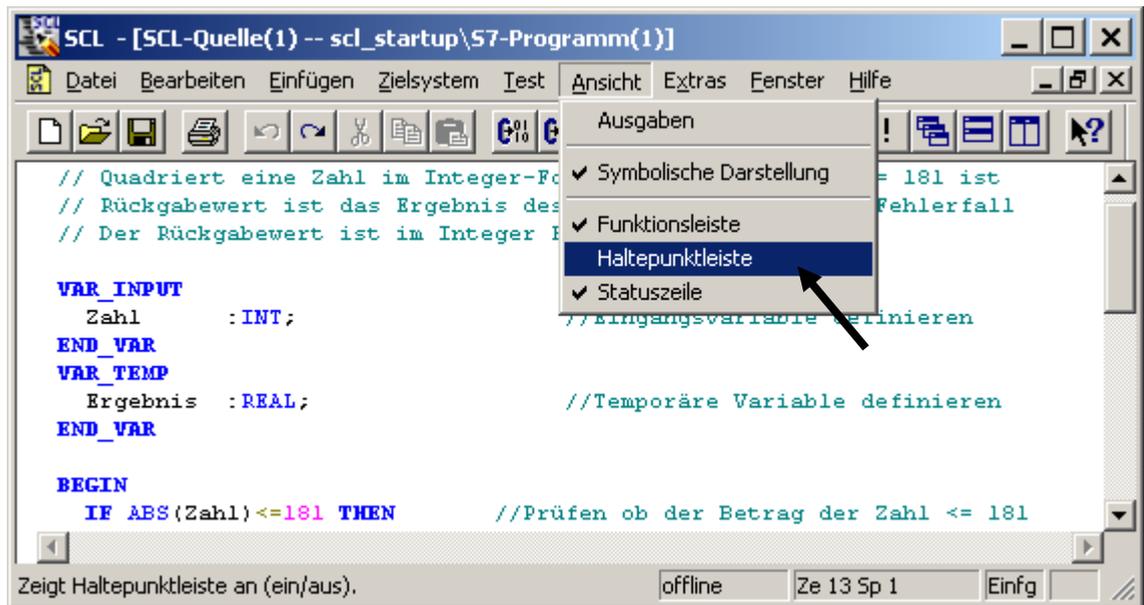


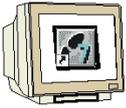
1. To this end, supplement the symbol table as follows:

IW 0	Input_1	First number to be squared
IW 2	Input_2	Second number to be squared
IW 4	Input_3	Third number to be squared
OW 0	Output_1	Result 1
OW 2	Output_2	Result 2
OW 4	Output_3	Result 3
FC 1	Square	Square function

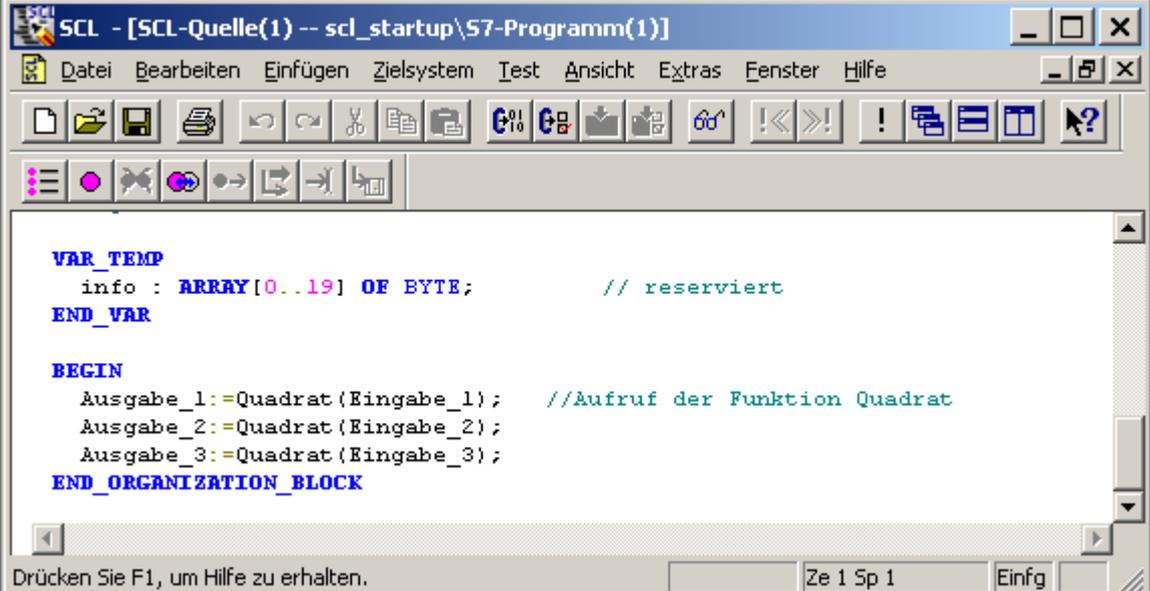


2. The symbol bar for the stop point commands can be inserted in the menu View 'Stop Point Bar'. This is confirmed with a check mark at the menu option (→ View → Stop Point Bar).





3. The additional calls of the function Square are inserted in the SCL source. Then, the program has to be saved , compiled  and checked for syntax errors. After all errors are removed, the blocks can be loaded to the CPU . (→  →  → ).



```

SCL - [SCL-Quelle(1) -- scl_startup\S7-Programm(1)]
Datei Bearbeiten Einfügen Zielsystem Test Ansicht Extras Fenster Hilfe

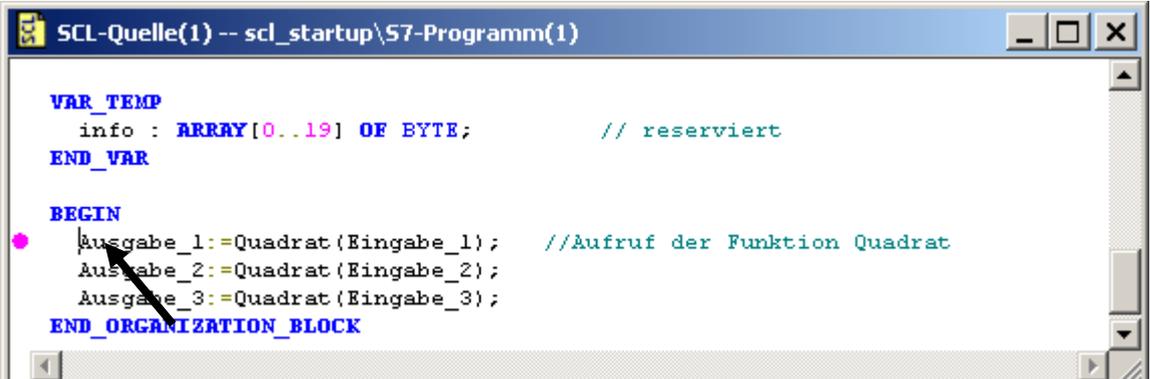
VAR_TEMP
  info : ARRAY[0..19] OF BYTE;           // reserviert
END_VAR

BEGIN
  Ausgabe_1:=Quadrat(Eingabe_1);        //Aufruf der Funktion Quadrat
  Ausgabe_2:=Quadrat(Eingabe_2);
  Ausgabe_3:=Quadrat(Eingabe_3);
END_ORGANIZATION_BLOCK
    
```

Drücken Sie F1, um Hilfe zu erhalten. Ze 1 Sp 1 Einfg



4. Place the mark in the line with the first call of function FC1 and insert a stop point by clicking on the button . (→ )



```

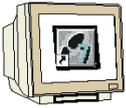
SCL-Quelle(1) -- scl_startup\S7-Programm(1)

VAR_TEMP
  info : ARRAY[0..19] OF BYTE;           // reserviert
END_VAR

BEGIN
  • Ausgabe_1:=Quadrat(Eingabe_1);        //Aufruf der Funktion Quadrat
  Ausgabe_2:=Quadrat(Eingabe_2);
  Ausgabe_3:=Quadrat(Eingabe_3);
END_ORGANIZATION_BLOCK
    
```



Note: It is possible to insert several stop points. To this end, mark the desired line and insert an additional stop point . To clear a stop point that you already set, mark it and then use the same button .



5. Now, step by step monitoring is started by activating the stop points that were set . Click on  to edit the first instruction and have it displayed in the right partial window. (→  → )

The screenshot shows the SCL editor window titled "SCL-Quelle(1) -- scl_startup\S7-Programm(1) ONLINE". The left pane contains the following code:

```

ORGANIZATION_BLOCK OB1
  //Organisationsbaustein mit Aufx

  VAR_TEMP
    info : ARRAY[0..19] OF BYTE;
  END_VAR

  BEGIN
    Ausgabe_1:=Quadrat(Eingabe_1);
    Ausgabe_2:=Quadrat(Eingabe_2);
    Ausgabe_3:=Quadrat(Eingabe_3);
  END_ORGANIZATION_BLOCK
    
```

The right pane displays the instruction being edited: "Ausgabe_1 = 4 , Eingabe_1 = 2". A pink dot in the left pane marks the first instruction.



6. You can have the next instruction edited by another click on . (→ )

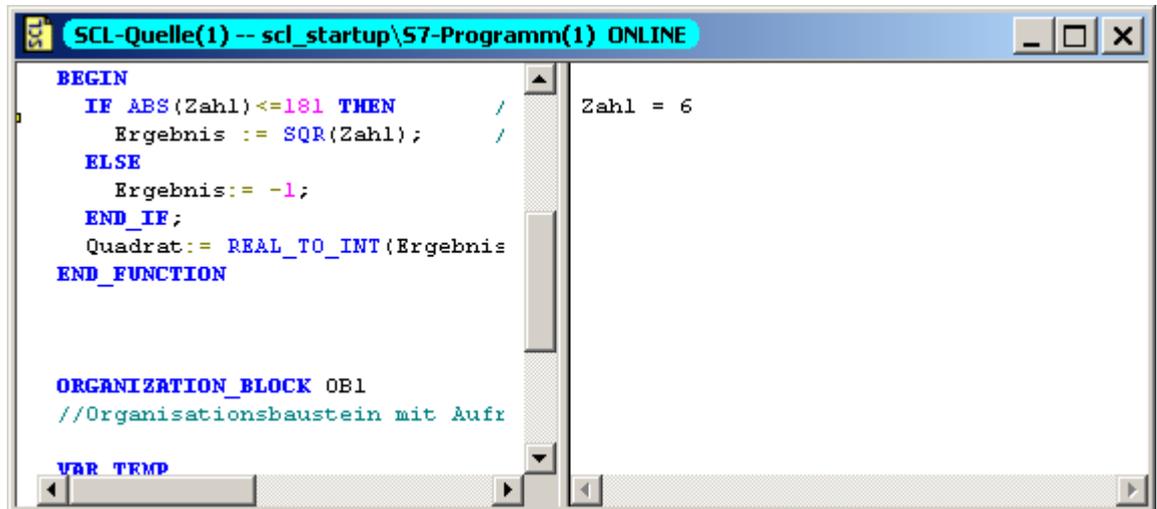
The screenshot shows the SCL editor window titled "SCL-Quelle(1) -- scl_startup\S7-Programm(1) ONLINE". The left pane contains the same code as in the previous screenshot. The right pane displays the second instruction being edited: "Ausgabe_2 = 16 , Eingabe_2 = 4". A pink dot in the left pane marks the second instruction.



Note: For step by step monitoring, only the values of the instruction that is currently processed is displayed. Although block calls are executed with the command 'To next instruction' , they are not displayed step by step.



7. A called block can be monitored step by step with the command 'Execute Call' . The first instruction is displayed with the function 'To next instruction' . (→  → ).



The screenshot shows the SCL editor window titled "SCL-Quelle(1) -- scl_startup\S7-Programm(1) ONLINE". The code on the left is as follows:

```

BEGIN
  IF ABS(Zahl) <= 181 THEN /
    Ergebnis := SQR(Zahl); /
  ELSE
    Ergebnis := -1;
  END_IF;
  Quadrat := REAL_TO_INT(Ergebnis);
END_FUNCTION

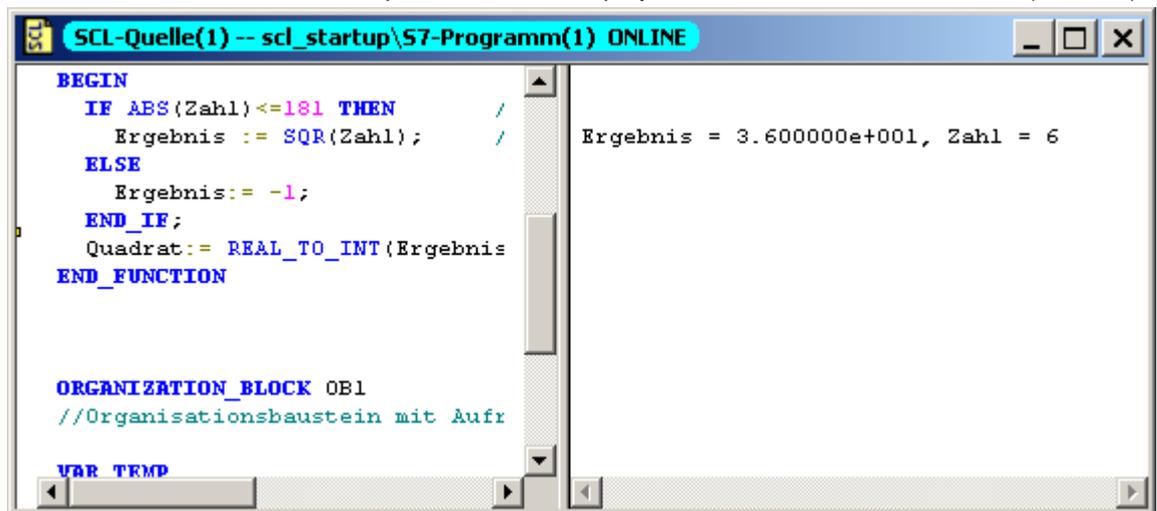
ORGANIZATION_BLOCK OB1
//Organisationsbaustein mit Aufg

VAR TEMP
    
```

The right-hand pane shows the current state of the program: "Zahl = 6".



8. Each additional instruction is processed and displayed with the same function . (→ )



The screenshot shows the SCL editor window titled "SCL-Quelle(1) -- scl_startup\S7-Programm(1) ONLINE". The code on the left is identical to the previous screenshot:

```

BEGIN
  IF ABS(Zahl) <= 181 THEN /
    Ergebnis := SQR(Zahl); /
  ELSE
    Ergebnis := -1;
  END_IF;
  Quadrat := REAL_TO_INT(Ergebnis);
END_FUNCTION

ORGANIZATION_BLOCK OB1
//Organisationsbaustein mit Aufg

VAR TEMP
    
```

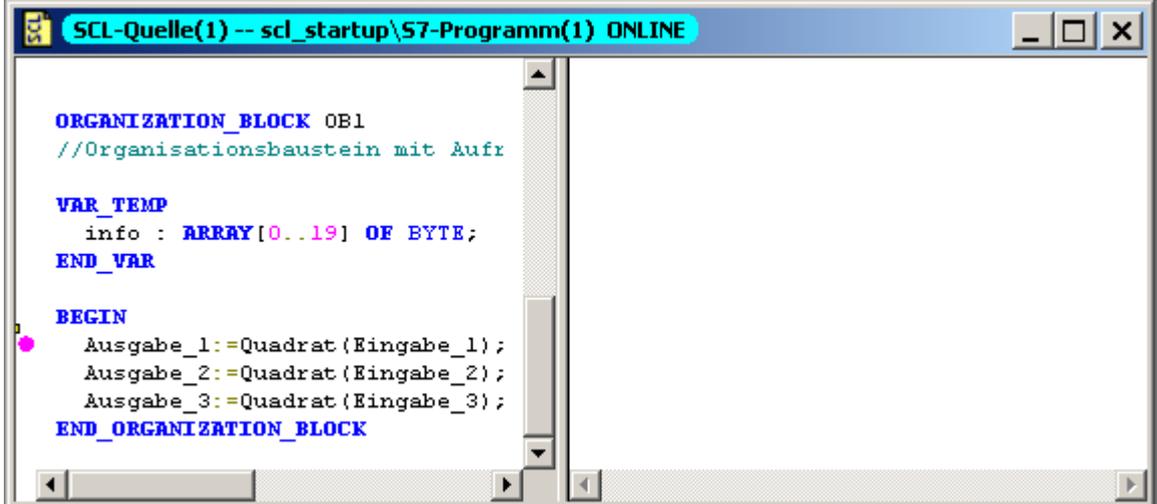
The right-hand pane now shows: "Ergebnis = 3.600000e+001, Zahl = 6".



Note: If the end of the block is reached and if the command 'To next instruction'  is executed again, the instruction is edited after the block call of the calling block.



9. With the command 'Continue' , the program is executed up to the next stop point. If no stop point is set when the end of the program is reached, the program cycle is restarted and processed up to the next stop point. (→ )



```

SCL-Quelle(1) -- scl_startup\S7-Programm(1) ONLINE

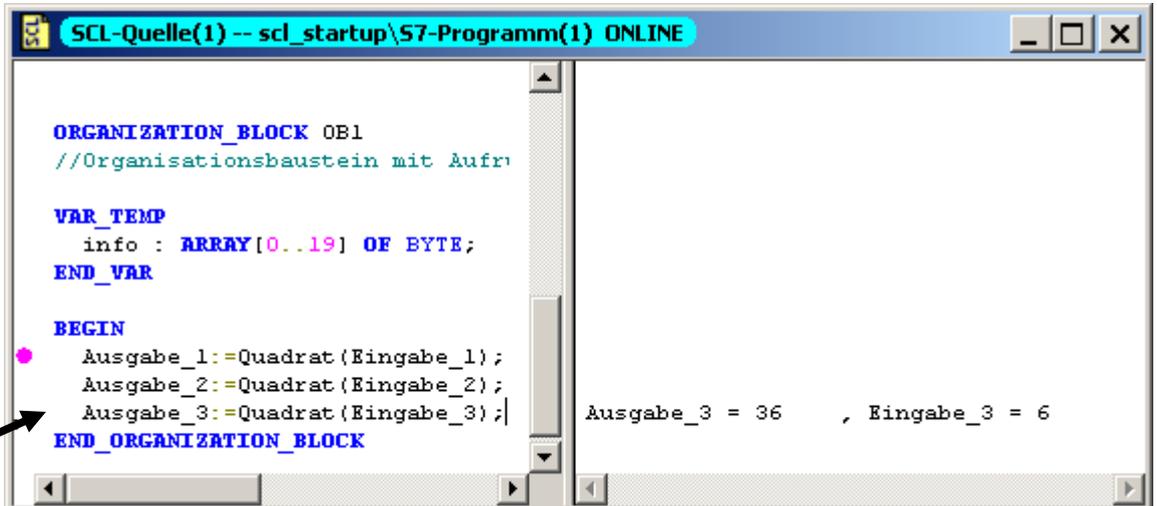
ORGANIZATION_BLOCK OB1
//Organisationsbaustein mit Aufgr

VAR_TEMP
info : ARRAY[0..19] OF BYTE;
END_VAR

BEGIN
  Ausgabe_1:=Quadrat(Eingabe_1);
  Ausgabe_2:=Quadrat(Eingabe_2);
  Ausgabe_3:=Quadrat(Eingabe_3);
END_ORGANIZATION_BLOCK
    
```



10. Now, mark the start of the line with the third call of the function Square. With the command 'Up to the Marking' , the program is processed up to this instruction. With 'To next instruction' , the instruction can be processed and displayed. (→  → )



```

SCL-Quelle(1) -- scl_startup\S7-Programm(1) ONLINE

ORGANIZATION_BLOCK OB1
//Organisationsbaustein mit Aufgr

VAR_TEMP
info : ARRAY[0..19] OF BYTE;
END_VAR

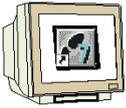
BEGIN
  Ausgabe_1:=Quadrat(Eingabe_1);
  Ausgabe_2:=Quadrat(Eingabe_2);
  Ausgabe_3:=Quadrat(Eingabe_3);
END_ORGANIZATION_BLOCK
    
```

Ausgabe_3 = 36 , Eingabe_3 = 6

Mark here!



Note: The command 'Up to the marking'  can only be used within the block. If the program is to be processed to a position in another block, this has to be done with a stop point and the command 'Continue' .



11. Again mark the start of the line with the third call of the function Square. Insert an additional stop point at this location. (→ )

```

SCL-Quelle(1) -- scl_startup\S7-Programm(1) ONLINE
//Organisationsbaustein mit Aufruf
VAR_TEMP
  info : ARRAY[0..19] OF BYTE;
END_VAR
BEGIN
  Ausgabe_1:=Quadrat(Eingabe_1);
  Ausgabe_2:=Quadrat(Eingabe_2);
  Ausgabe_3:=Quadrat(Eingabe_3);
END_ORGANIZATION_BLOCK
  
```

Output: Ausgabe_3 = 36 , Eingabe_3 = 6

Mark here!

12. With 'Edit stop points' , stop points can be activated, deactivated or cleared. Deactivate the first stop point in OB1. (→  → Deactivate → OK)

Haltepunkte bearbeiten

Haltepunkte im Code

Haltepunkte (mit Haken = Freigegeben):

<input type="checkbox"/>	OB1	32	SCL-Quelle(1) -- SCL_Startup\SIMATIC 300(1)\CPU 3
<input checked="" type="checkbox"/>	OB1	34	SCL-Quelle(1) -- SCL_Startup\SIMATIC 300(1)\CPU 3

Buttons: Gehe zu, Löschen, Alle löschen

Aufrufumgebung

Aufrufpfad kein Aufrufpfad

Aufrufpfad:

Instanz-DB:

Global-DB:

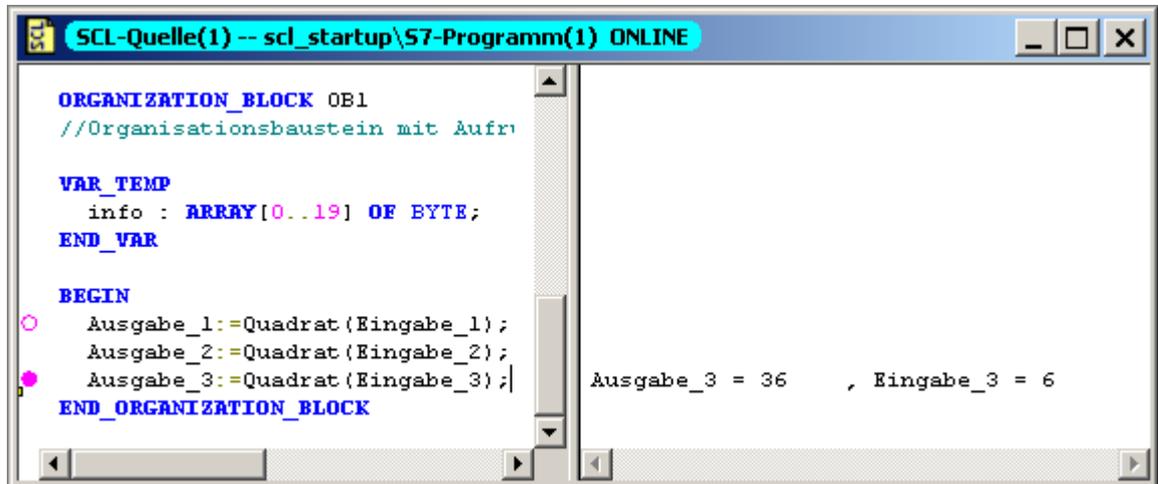
Buttons: OK, Abbrechen, Hilfe

Deactivate here!



13. Now, select the command 'Continue' . The program will be processed to the next active stop point. With the function 'To next instruction' , the values of the current line are displayed.

(→  → )



The screenshot shows a window titled "SCL-Quelle(1) -- scl_startup\S7-Programm(1) ONLINE". The code on the left is as follows:

```

ORGANIZATION_BLOCK OB1
//Organisationsbaustein mit Auftr

VAR_TEMP
  info : ARRAY[0..19] OF BYTE;
END_VAR

BEGIN
  Ausgabe_1:=Quadrat(Eingabe_1);
  Ausgabe_2:=Quadrat(Eingabe_2);
  Ausgabe_3:=Quadrat(Eingabe_3);
END_ORGANIZATION_BLOCK
    
```

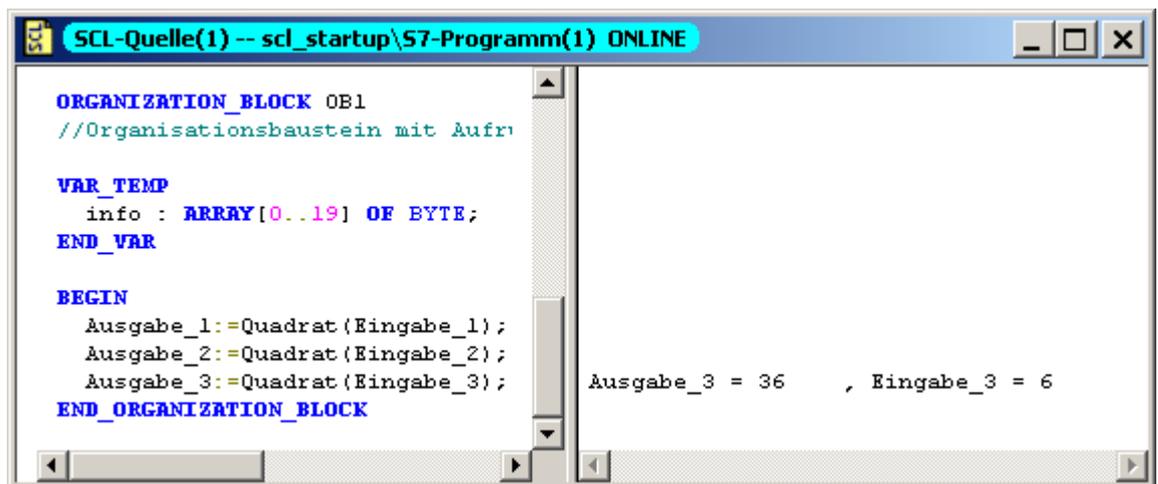
On the right side of the window, the current execution state is displayed: "Ausgabe_3 = 36 , Eingabe_3 = 6". A pink circle with a right-pointing arrow is positioned on the line "Ausgabe_3:=Quadrat(Eingabe_3);".



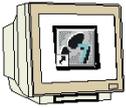
Note: With the function 'Continue' , the program is executed up to and including the marked line. However, the display is not updated. It is updated with the function 'To next instruction' .



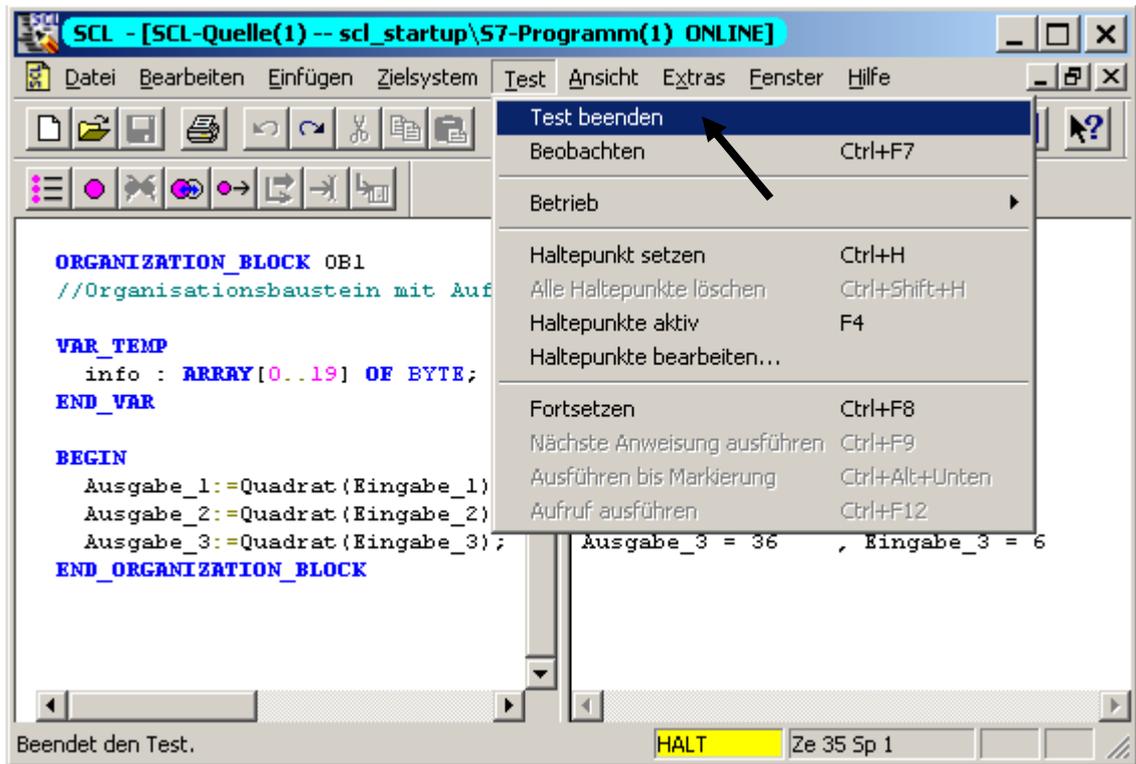
14. If you are switching off  and clearing  the stop points, you are still in the test mode. The partial window for the current values is still displayed. (→  → )



The screenshot shows the same window as in step 13. The code is identical. The current execution state on the right is still "Ausgabe_3 = 36 , Eingabe_3 = 6". However, the pink circle with a right-pointing arrow is no longer present on the code, indicating that the stop point has been cleared.



15. The test function has to be closed in the menu 'Test'. (→ Test → Close Test)



Note: As long as the test functions are active, no changes can be made in the text editor. Only after the test is completed is it possible to edit the SCL source again.