Fundamental Concepts of Programming Languages

Control Structures

Lecture 11

conf. dr. ing. Ciprian-Bogdan Chirila

University Politehnica Timisoara
Department of Computing and Information Technology

December 6, 2022

FCPL - 11 - Control Structures

1. Instruction level control structures
   - Sequence
   - Selection
   - Repetition
2. Subprograms
   - Side effects
   - Pseudonyms
3. Exception handling
   - Exceptions in Ada
   - Exceptions in C#
4. Bibliography

Control structures

We focus on mechanisms that allow the programmer to control the flow of actions at:

- Instruction level
- Subunit level

FCPL - 11 - Control Structures

Instruction level control structures

- Specify the order in which the individual program instructions are executed
- Grouped in three categories
  - Sequence
  - Selection
  - Repetition

Sequence

- The most simple control sequence
- Implied by imperative languages
- We do not refer to concurrent languages
- The instructions are executed in the order in which they are written

Sequence

- In many PLs set of instructions can be grouped together to form a composed instruction
- In Python
  - indented by a tab
- In C, Java, C#, JavaScript, TypeScript

```
{
  ...
}
```

Selection

- Allow us to select
  - an alternative between two or more available
  - depending on a logical condition
- In Algol-Pascal like PLs

```
if condition then
 sequence_of_instructions
else
 sequence_of_instructions
end if;
```

Selection in Python

```
if test expression:
 statement(s)

if test expression:
 Body_of_if
else:
 Body_of_else

if test expression:
 Body_of_if
elif test expression:
 Body_of_elif
else:
 Body_of_else
```

Selection between multiple alternatives

- case
  - like in Pascal, Ada, Algol 68
- switch
  - like in C, Java, C#, JavaScript
- the selection is based on a selector value of scalar type
- the programmer must specify the variants and the values for which each value is selected
- there is an option of specifying a variant to be chosen when there is no match

Pascal example

```
if mark <= 10 then
 case mark of
  1,2,3,4 : writeln('failed');
  5,6,7 : writeln('passed');
  8,9 : writeln('good');
  10 : writeln('excellent');
 end
else
 writeln('wrong mark');
```

Ada example

```
case mark of
 when 1..4 => put_line("failed");
 when 5|6|7 => put_line("passed");
 when 8|9 => put_line("good");
 when 10 => put_line("excellent");
 when others => put_line("wrong mark");
end case;
```

C like PLs example

```
switch(mark)
{
 case 1: case 2: case 3: case 4: printf("failed"); break;
 case 5: case 6: case 7: printf("passed"); break;
 case 8: case 9: printf("good"); break;
 case 10: printf("excellent"); break;
 default: printf("wrong mark"); break;
}
```

Python replacement

```
def numbers_to_strings(argument):
 switcher =
 {
  1:"failed",2:"failed",3:"failed",4:"failed",
  5:"passed",6:"passed",7:"passed",
  8:"good",9:"good",
  10:"excellent",
 }
 return switcher.get(argument, "wrong mark")

if __name__ == "__main__":
 argument=5
 print (numbers_to_strings(argument))
```

Repetition

- is the base mechanism for making complex computations
- it means to execute repeatedly an instruction or a set of instructions
- structures are controlled
  - by condition
  - by counter

Condition controlled structures

Condition controlled structures

- Repetition with initial test
  - Pascal: while condition do
  - C, Java, C#, JavaScript, TypeScript:
    `while(condition) instruction;`
- Repetition with final test
  - Pascal: repeat instr_sequence until condition
  - C, Java, C#, JavaScript, TypeScript:
    `do instruction while(condition);`

Counter controlled structures

- The counter evolves
  - from an initial value
  - to a final value
- The general form is

```
for variable:= initial_value to final_value step step_value do
 instruction
```

Counter controlled structures

- In Pl/I and Algol 68

```
for
 variable from initial_value
 by step_value
 to final value
 while condition do
sequence_of_instructions
```

Counter controlled structures

- In C like PLs

```
for(expr1; expre2; expr3)
 instruction;

expr1;
while(expr2)
{
 instruction;
 expr3;
}
```

Exiting the repetition

- Unconditional jump
  - goto
- Specialized jump instruction to exit a loop
  - In Ada
    - exit
  - In C like PLs
    - break
- Only one loop, but not all surrounding ones
  - In Ada
    - exit name
    - Name is the label of the outer loop
  - In C like PLs
    - continue

FCPL - 11 - Control Structures

Control structures at subunit level Subprograms

- Side effects
  - Modifications provoked by a subprogram on an entity that is not local to that subprogram
  - They are troublesome especially in case of functions
  - `v=a+f(a,b)+c;`
- Pseudonyms
  - An object may be referred by two or multiple names
  - It may show up in the case of address parameter transmission mechanism

Pseudonyms example in C++

```
int y;
-----------------
void p(int &x)
{
 x=2*x;
 y=x+y;
};
-----------------
y=1;
p(y);
-----------------
```

Pseudonyms example in C++

- When two or more arguments transmitted by address represent the same object

```
int z;
------------------
void p(int &x,int &y)
{
 x=2*x;
 y=2*y;
}
------------------
z=3;
p(z,z);
```

Pseudonyms example in C++

- When a structure or components of that structures are transmitted as parameters

```
typedef float tab_t[100];
tab_t tab;
procedure pp(int &x, tab_t &tab)
{
  ...
}
--------------------
pp(tab[3],tab);
--------------------
```

Pseudonyms in arrays

- When two arguments are two elements of the very same array
- `p(t[i],t[j]);`
- pseudonyms show up only if `i==j`

FCPL - 11 - Control Structures

Exception handling in Ada

- 5 types of exceptions
- constraint_error
  - Violating the boundaries of a subdomain
  - Referring illegally a field from an article with variants
  - Refering a null pointer
- numeric_error
  - arithmetical overflow
- storage_error
  - exceeding memory space
- select_error, tasking_error
  - concurrency related errors

Exception handling in Ada

- declaring exceptions
  - error, end : exception;
- raising exceptions
  - raise error;

Exception handling example

```
function f(x : float) return float is
negative : exception;
begin
if x<0 then
 raise negative;
else
 return 1/sqrt(x);
end if;
exception
 when numeric_error =>
  return 0; -- return 0 if x is 0
 when negative =>
  return -1; -- return -1 if x is negative
end f;
```

Exception handling example

```
package stack is
 error: exception;
 type stack_type(max_no:integer) is limited private;
 function pop(s:in out stack_type) return integer;
 procedure push(s:in out stack_type; x: integer);
 function top(s:stack_type) return integer;
 procedure init(s:out stack_type);
private
 type stack_type(max_no:integer) is
 record
  tab_st:array(1..max_no) of integer;
  ind:integer;
  end record;
end stack;
```

Exception handling example

```
package body stack is

 function empty(s:stack_type) return boolean is
 begin ... end empty;

 function overflow(s:stack_type) return boolean is
 begin ... end overflow;

 function pop(s: in out stack_type) return integer is
 begin
  if not empty(s) then
   s.ind:=s.ind-1;
    return s.tab_st(s.ind);
   else raise error;
end if;
```

Exception handling example

```
procedure push(s: in out stack_type; x:integer) is
begin
 if not overflow(s) then
  s.tab_st(s.ind):=x;
  s.ind:=s.ind+1;
 else raise error;
 end if;
end push;
```

Exception handling example

```
function top(s:stack_type) return integer is
begin
 if not empty(s) then
  return s.tab_st(s.ind-1);
 else
  raise error;
 end if;
end top;

function init(s: out stack_type) is
begin
 ...
end init;

begin
```

Exception handling example

```
procedure st is
use stack;
stk:stack_type(100);
------------------------
init(stk);
------------------------
push(stk,10);
------------------------
i:=top(stk);
------------------------
```

Exception handling example

```
exception
 when error =>
  put_line("error using stack");
  while not empty(stk) loop
   put(pop(stk));
  end loop;
end st;
```

Exception handling in C#

- is similar to C++ and Java
- exceptions are represented by classes
- Class System.Exception
- all classes denoting exceptions must be derived from predefined class Exception part of the System name space
- classes deriving from Exception are
  - SystemException
    - generated by the execution engine
  - AplicationException
    - generated by program applications
    - can be derived by the programmer creating his own exceptions

Exception classes derived from SystemException

- ArrayTypeMismatchException
  - the assigned type is incompatible with the array type
- DivideByZeroException
  - attempt to devide by zero
- IndexOutOfRangeException
  - the index exceeds the boundaries of the array
- InvalidCastException
  - incorrect cast at run time

Exception classes derived from SystemException

- OutOfMemoryException
  - insufficient memory for new operator allocations
- OverflowException
  - arithmentical overflow
- StackOverflowException
  - exceeding stack capacity

Basic elements in exception handling

- keywords
  - try
  - catch
  - throw
  - finally

How do exceptions work?

- try
  - try blocks contain the error sensitive instructions that must be checked
- throw
  - if an exception occurs then it is thrown
- catch
  - the program can intercept this exception and handles it according to the application requirements
- exceptions are launched
  - automatically by the C# execution engine
  - manually using the throw keyword

How are exceptions handled?

- the code to executed when exiting the block must be places in a finally block
- in order to catch any exception
- of any type
- the catch clause is used with no parameter
- thus, a universal routine is created to intercept and handle all exceptions

Simple example for exception handling

```
using System;
class Example1
{
 public static void Main()
 {
  int [] nums = new int[4];
  try
  {
   Console.WriteLine("Before exception");
   // we generate an exception of invalid index
   nums[7] = 10;
   Console.WriteLine("Message not to be printed");
  }
```

Simple example for exception handling

```
  catch(IndexOutOfRangeException)
  {
   // we intercept the exception
   Console.WriteLine("Index out of bounds");
  }
  Console.WriteLine("After catch");
 }
}
```

Simple example for exception handling

- displayed text:

```
Before exception
Index out of bounds
After catch
```

Manually throwing an exception

```
using System;
class Example2
{
 public static void Main()
 {
  try
  {
   Console.WriteLine("Before throw");
   // launching  exception
   throw new DivideByZeroException();
  }
  catch(DivideByZeroException)
  {
   // intercepting the exception
   Console.WriteLine("Exception intercepted");
  }
  Console.WriteLine("After try/catch");
}
```

Manually throwing an exception

- displayed text

```
Before throw
Exception intercepted
After try/catch
```

FCPL - 11 - Control Structures

Bibliography

1. Horia Ciocarlie – The programming language universe, second edition, Timisoara, 2013.
2. Carlo Ghezzi, Mehdi Jarayeri – Programming Languages, John Wiley, 1987.
3. Ellis Horrowitz – Fundamentals of programming languages, Computer Science Press, 1984.
4. Donald Knuth – The art of computer programming, 2002.