
CS 241

Control Structures

Christopher A. Gantz

SPS Undergraduate Program
Regis University
cgantz@regis.edu

Lecture Topic #11

CS 241: Subprograms

Christopher A. Gantz
School of Professional Studies
Regis University
cgantz@regis.edu

Overview

- Reading
 - Nance textbook Pages 319-353
- Procedures
 - Subprograms & Modules
 - Why Modular Programs?
 - Procedures
 - Procedure Interfaces

Subprograms

- A subprogram is a program within a program
- Each subprogram should represent and implement some task that is either simple or complex
- Subprograms are the mechanisms utilized to realize a stepwise refinement subtask
- Subprograms are also implementations of the modules that are the result of a top-down design and stepwise refinement

Subprograms (Cont.)

- A subprogram is a module implementation that contains definitions, declarations and corresponding executable statements of some task
- A program that is implemented with subprograms is said to be modular in structure.

Why Modular Programs?

- Modular Structured programs are easier to
 - Test
 - Debug
 - Correct
- Modularity within implementation structure also facilitates “Bottom-up Testing”
- “Bottom-up Testing” is testing of modules independently before they collected together as a solution.

Structured Programming

- The process of developing/implementing a program where the focus of the development effort is in the interfacing of independent modules
- An interface is simply a formal definition of what information is to be passed and returned from a module
- Interfacing in Pascal is realized by defining/declaring parameters to/for a subprogram/module
- This methodology is very well suited for large team program/solution development

Subprogram Types

- There are two types of subprograms available in standard Pascal

- I. Procedures

- II. Functions

-

Procedures

- A Pascal procedure is a subprogram that has been designed to perform a specific task as a component of a larger encompassing program
- Procedures are most useful for
 - I. Implementing top-down design subtasks as modules
 - II. Avoiding writing repeated segments of code
-

Procedures (Cont.)

- Some common procedural tasks are as follows:
 - Collecting Input
 - Processing Data
 - Displaying standard formatted output
-

Procedure Syntactical Properties and Form

- Procedures contain the same declaration and executable subsections as Pascal programs
- A Procedure consist of the following sections:
 - Procedure heading
 - Optional declaration section
 - Executable section
- The format of the procedure heading is:
PROCEDURE <procedure name>

Procedure Syntactical Properties and Form (Cont.)

- Examples of procedure headings without parameters (i.e. an interface)

PROCEDURE CollectInput;

PROCEDURE ProcessData;

PROCEDURE DisplayOutput;

-

Procedure Section Constraints

- The declaration section is identical in format and syntax to the declaration section of a program
- The procedure declaration section can contain:
 - CONST subsection
 - VAR subsection
- The procedure executable section is also identical to the program executable section
- However, the terminating END must have a semi-colon ':'

Procedure (without parameters) Template

PROCEDURE <procedure name>

{Declaration Section}

CONST {

{ list of constant definitions }

VAR

{list of variable declarations }

{Executable Section}

BEGIN {Body of Procedure}

END;

Procedure definition/declaration constraints/rules

- Procedure definitions/declarations are located in the VAR subsection of the program
 - See example on page 323 of text
- Remember to liberally utilize comments, blank lines and indenting to enhance the readability of procedure definitions/declarations
 - See procedure writing style example on pages 323 and 324 of text
-

Procedure Invocation

- Invocation of a procedure occurs when the procedure name is used within the executable section of a program
 - See Example 7.3 on page 328 & 329 of Text

-

Procedures with Parameters

- Defines a procedure with a parameter list which defines the module interface

General Syntactical Form

```
PROCEDURE <name> (<parameter list>);
```

```
{Declaration Section}
```

```
BEGIN {Executable Section}
```

```
    {Body of Procedure}
```

```
END;
```

Parameter list Specifications

- Parameters are utilized to describe the interface to the procedure
- Parameters allow values to be passed from the calling program to the procedure
- Parameters transmit values from the main program to the procedure

Types of Parameters

- Value parameters
 - Values transmitted/passed into procedures
- Variable parameters
 - Values transmitted/passed into procedures
 - Values transmitted/returned back to the program
-

Rules of Parameter Usage

- Rules of parameter usage
 1. Number & order of parameters must match between definition & invocation
 2. Types of parameters must match
 3. Parameter types are declared in the procedure heading
- Parameters defined/declared within the procedure definition are referred to as formal parameters
- Parameters defined at the time of procedure invocation are referred to as actual parameters

Rules of Parameter Usage (Cont.)

- Definition/Declaration Example

```
PROCEDURE PrintNum(N1, N2 :integer; N3 :real);
```

- Invocation Example

```
PrintNum(Num1, Num2, Num3);
```

-

Variable Parameters

- Implementation of interface semantic referred to as pass/call by reference

- Definition/Declaration Example

```
PROCEDURE PrintNum(VAR N1, N2 :integer; N3 :  
    real);
```

- Invocation Example

```
PrintNum(Num1, Num2, Num3);
```

-

Side Effects

- Intentional/Unintentional change in the value of a variable resulting from some action executed in the program
- Example

```
FOR Count := 1 TO 25 DO
```

```
BEGIN
```

```
    Total := Total + Count;
```

```
    Count := Count + 1; { Side Effect }
```

```
END;
```

Side Effects (Cont.)

- Unintentional side effects are often caused by the misuse of variable parameters
- Therefore be very careful of variable parameters