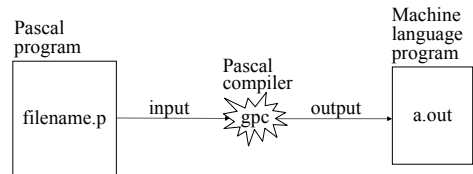# Getting Started With Pascal Programming

How are computer programs created

What is the basic structure of a Pascal Program

Variables and constants

Input and output

Pascal operators

Common programming errors

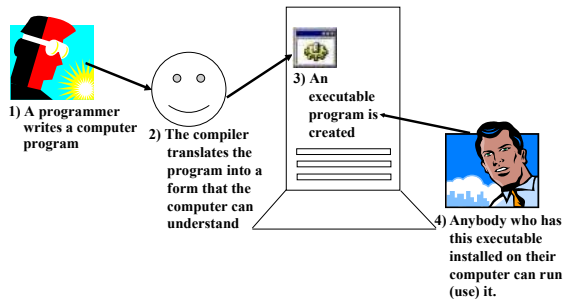**Introduction to program design**
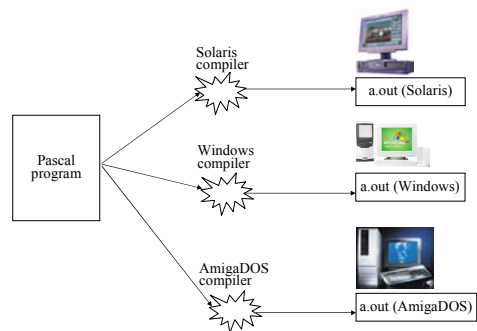

Compiling Programs: Basic View

Computer Programs

Binary is the language of the computer

1) A programmer writes a computer program

2) The compiler translates the program into a form that the computer can understand

3) An executable program is created

4) Anybody who has this executable installed on their computer can run (use) it.

Compiling Programs On Different Operating Systems

Solaris compiler → a.out (Solaris)

Windows compiler → a.out (Windows)

AmigaDOS compiler → a.out (AmigaDOS)

Pascal program

Translators

Convert computer programs to machine language

Types

1) Interpreters
   - As the program is run the interpreter translates the program (translating a part at a time).
   - If there are any errors during the process of interpreting the program, the program will stop running when the error is encountered.
2) Compilers
   - Before the program is run the compiler translates the program (compiling it all at once).
   - If there are *any errors* during the compilation process, no machine language executable will be produced.
   - If there are *no errors* during compilation then the translated machine language program can be run.

Basic Structure Of Pascal Programs

Header

Program documentation
program *name* (input, output)*;*

Declarations

const
:

Statements

begin
:
end**.**

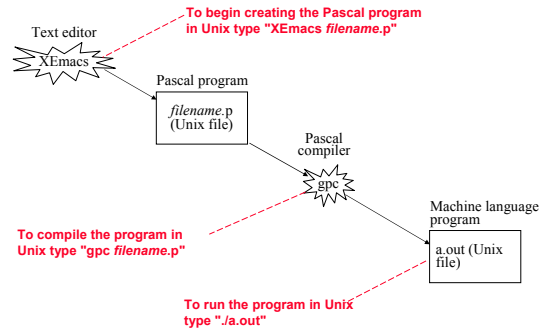## Details Of The Parts Of A Pascal Program

Headers
- Parts:
  - 1) Program documentation
    - What does the program do, author(s), version number, date of last modification etc.
    - Comments for the reader of the program (and not the computer)
      - (*   Marks the beginning of the documentation
      - *)   Marks the end of the documentation
  - 2) Program heading
    - Name of program, if input and/or output operations performed by the program.

- Example
  ```
  (*
   * Tax-It v1.0: This program will electronically calculate your tax return.
   *)
  ```
  Documentation

  ```
  program taxIt (input, output);
  ```
  Heading

Creating And Compiling Programs: On The Computer Science Network

To begin creating the Pascal program in Unix type "XEmacs *filename*.p"

Text editor
XEmacs

Pascal program
filename.p (Unix file)

Pascal compiler
gpc

To compile the program in Unix type "gpc *filename*.p"

Machine language program
a.out (Unix file)

To run the program in Unix type "./a.out"

Details Of The Parts Of A Pascal Program (2)

Declarations
- List of constants
- More to come later during this term regarding this section

Statements
- The instructions in the program that actually gets things done
- They tell the computer what to do as the program is running
- Statement are separated by semicolons "**;**"
- Much more to come later throughout the rest of the term regarding this section

---

Variables

Set aside a location in memory

Used to store information (temporary)

Types:
- integer – whole numbers
- real – whole numbers and fractions
  - Can't start or end with a decimal (must be a digit)
- char – alphabetic, numeric and miscellaneous symbols (type "man ascii")
- boolean – true or false values

Usage:
- Declaration
- Accessing or assigning values to the variables

The Smallest Pascal Program

program smallest;

begin

end.

Note: The name "smallest" should match the filename "smallest.p". You can find an online version of this program in the Unix file system under /home/231/examples/intro/smallest.p (the compiled version is called "smallest").

Declaring Variables

Sets aside memory

Memory locations are addressed through the name of the variable

RAM

Name of variable   RESERVED
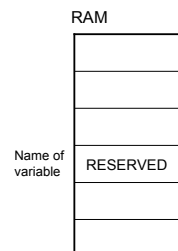
Declaring Variables (2)

The declaration occurs between the begin and the end statements.

Header

| Program documentation |
| program *name* (input and output operations); |

Declarations

| const |
| : |

Statements

| begin |
| **Declare variables here** |
| end. |

Variable Naming Conventions (2)

• Okay:
  - tax_rate
  - firstName
• Not Okay (violate Pascal syntax)
  - 1abc
  - test.msg
  - good-day
  - program
• Not okay (bad style)
  - x
  - wrintln

Declaring Variables (3)

Format:

var *name of first variable* : *type of first variable*;

var *name of second variable*: *type of second variable*;

Examples:

var height: real;

var weight: real;

var age: integer;

Reserved Words

Have a predefined meaning in Pascal that **cannot** be changed

and	array	begin	case	const	div	do	downto	else
end	file	for	forward	function	goto	if	in	label
mod	nil	not	of	or	packed	procedure	program	record
repeat	set	then	to	type	until	var	while	while

Variable Naming Conventions

• Should be meaningful
• Any combination of letters, numbers or underscore (*can't* begin with a number and *shouldn't* begin with an underscore)
• Can't be a reserved word (see the "Reserved Words" slide)
• Avoid using predefined identifiers (see the "Standard Identifiers" slides)
• Avoid distinguishing variable names only by case
• For variable names composed of multiple words separate each word by capitalizing the first letter of each word (save for the first word) or by using an underscore.

Standard Identifiers

Have a predefined meaning in Pascal that **SHOULD NOT** be changed

Predefined constants
• false
• true
• maxint

Predefined types
• boolean
• char
• integer
• real
• text

Predefined files
• input
• output

Standard Identifiers (2)

Predefined functions

abs	arctan	chr	cos	eof	eoln
exp	ln	odd	ord	pred	round
sin	sqr	sqrt	succ	trunc	

Assigning Values To Variables

Format:
Destination := Source; [1]

Example:
grade := 100;
age := median;
interest := principle * rate;
initial := 'j';

Standard Identifiers (3)

Predefined procedures

dispose	get	new	pack	page
put	read	readln	reset	rewrite
unpack	write	writeln		

Assigning Values To Variables (2)

Avoid assigning mixed types:

```
program variableExample;
begin
    var num1 : integer;
    var num2: real;

    num1 := 12;                    Rare
    num2 := 12.5;
    num2 := num1;              Not allowed!
    num1 := num2;

end.
```

Accessing Variables

Can be done by referring to the name of the variable

Format:
name of variable

Example:
num

Named Constants

A memory location that is assigned a value that cannot be changed

Declared in the constant declaration ("const") section

The naming conventions for choosing variable names also applies to constants but the name of constants should be all UPPER CASE. (You can separate multiple words with an underscore).

Format:

const

NAME_OF_FIRST_CONSTANT = value of first constant;

NAME_OF_SECOND_CONSTANT = value of second constant;

etc.

Named Constants (2)

Examples:

```
const
    TAX_RATE = 0.25;
    SAMPLE_SIZE = 1000;
    YES = True;
    NO = False;
```

Purpose Of Named Constants (2)

2) Makes the program easier to maintain
- If the constant is referred to several times throughout the program, changing the value of the constant once will change it throughout the program.

Location Of Named Constants

The declaration occurs in the declaration section.

Header

Program documentation
program *name* (input and output operations);

Declarations

const
Declare constants here

Statements

begin
end.

Purpose Of Named Constants (3)

```
program population (output);
const
    BIRTH_RATE = 0.1758;
    DEATH_RATE = 0.1257;
begin
    var populationChange : real;
    var currentPopulation : real;
    populationChange := (BIRTH_RATE - DEATH_RATE) * currentPopulation;
    if (populationChange > 0) then
        writeln('Births: ', BIRTH_RATE, ' Deaths:', DEATH_RATE, ' Change:',
            populationChange)
    else if (populationChange < 0) then
        writeln('Births: ', BIRTH_RATE, ' Deaths:', DEATH_RATE, ' Change:',
            populationChange)
end.
```

Purpose Of Named Constants

1) Makes the program easier to understand

populationChange := (0.1758 – 0.1257) * currentPopulation;

Vs.

Magic Numbers (avoid!)

```
const
    BIRTH_RATE = 0.1758;
    DEATH_RATE = 0.1257;
begin
    populationChange := (BIRTH_RATE – DEATH_RATE) *
                        currentPopulation;
```

Purpose Of Named Constants (3)

```
program population (output);
const
    BIRTH_RATE = 0.5;
    DEATH_RATE = 0.1257;
begin
    var populationChange : real;
    var currentPopulation : real;
    populationChange := (BIRTH_RATE - DEATH_RATE) * currentPopulation;
    if (populationChange > 0) then
        writeln('Births: ', BIRTH_RATE, ' Deaths:', DEATH_RATE, ' Change:',
            populationChange)
    else if (populationChange < 0) then
        writeln('Births: ', BIRTH_RATE, ' Deaths:', DEATH_RATE, ' Change:',
            populationChange)
end.
```

Purpose Of Named Constants (3)

```
program population (output);
const
  BIRTH_RATE = 0.1758;
  DEATH_RATE = 0.01;
begin
  var populationChange : real;
  var currentPopulation : real;
  populationChange := (BIRTH_RATE - DEATH_RATE) * currentPopulation;
  if (populationChange > 0) then
     writeln('Births: ', BIRTH_RATE, ' Deaths:', DEATH_RATE, ' Change:',
             populationChange)
  else if (populationChange < 0) then
     writeln('Births: ', BIRTH_RATE, ' Deaths:', DEATH_RATE, ' Change:',
             populationChange)
end.
```

Output (3)

Examples:

```
var num : integer;

num := 10;

writeln('line1');

write('line2A');

writeln('line2B');

writeln(num);

writeln('num=', num);
```

Output

Displaying information onscreen

Done via the write and writeln statements

Format:

```
write ('text message');
        or
writeln('text message');


write(<name of variable> or <constant>);
        or
writeln (<name of variable> or <constant>);


write('message', <name of variable>, 'message'…);
        or
writeln('message', <name of variable>, 'message'…);
```

Formatting Output

Automatic formatting of output
- Field width: The computer will insert enough spaces to ensure that the information can be displayed.
- Decimal places: For real numbers the data will be displayed in exponential form.

Manually formatting of output:

Format:
   write or writeln (<data>: <Field width for data>: <Number decimal places for real data>);

Examples

```
num := 12.34;

writeln(num);

writeln(num:5:2);
```

Output (2)

Example:
```
program simple (output);
begin
   writeln('This it it.');
end.
```

Formatting Output (2)

If the field width doesn't match the actual size of the field
- Field width too small – extra spaces will be added for numerical variables **but not** for other types of data.
- Examples:
   ```
   num := 123456;
   writeln(num:3);
   writeln('123456':3);
   ```

- Field width too large – the data will be right justified (extra spaces will be put in front of the data).
- Examples:
   ```
   num := 123;
   writeln(num:6);
   writeln('123':6);
   ```

Formatting Output (3)

If the number of decimal places doesn't match the actual number of decimal places.

- Set number of decimal places less than the actual number of decimal places
  – number will be rounded up.
- Example:
  ```
  num1 := 123.4567;
  writeln (num1:6:2);
  ```

- Set number of decimal places greater than the actual number of decimal places – number will be padded with zeros.
- Example:
  ```
  num1 := 123.4567;
  writeln(num1:6:6);
  ```

Input (2)

Examples:

```
program inputExampleOne (input);

begin

    var num1 : integer;

    var num2 : integer;

    read (num1);

    read (num2);

end.
```

Formatting Output: A Larger Example

For the complete program and executable look under
/home/231/examples/intro/out1.p (out1 for the compiled version)

```
program out1 (output);

begin

  var num1 : integer;

  var num2 : real;

  num1 := 123;

  num2 := 123.456;

  writeln('Auto formatted by Pascal', num1, num2);

  writeln('Manual format':13, num1:3, num2:7:3);

  writeln('Manual not enough':13, num1:2, num2:6:3);

  writeln('Manual too much':16, num1:4, num2:8:4);

end.
```

Input: Read Vs. Readln

Both:
- Reads each value entered and matches it to the corresponding variable.

Read
- If the user inputs additional values before hitting return, the additional values will remain on the 'input stream'.

Readln
- Any additional values entered before the return will be discarded.

Input

The computer program getting information from the user

Done via the read and readln statements

Format:
```
  read (<name of variable>);
              or
  readln (<name of variable>);
```

Input: Read Vs. Readln (An Example)

For the complete version of this program look in Unix under:
/home/231/examples/intro/read1.p (or read1 for the compiled version):

```
program read1 (input, output);
begin
  var num1 : integer;
  var num2 : integer;
  write('Type in an integer: ');
  read(num1);
  write('Type in an integer: ');
  read(num2);
  writeln('You typed in the following numbers:');
  writeln('First: ', num1, ' Second: ', num2);
end.
```

Input: Read Vs. Readln (An example (2))

For the complete version of this program look in Unix under:
/home/231/examples/intro/read2.p (or read2 for the compiled version)

```
program read2 (input, output);
begin
  var num1 : integer;
  var num2 : integer;
  write('Type in an integer: ');
  readln(num1);
  write('Type in an integer: ');
  readln(num2);
  writeln('You typed in the following numbers:');
  writeln('First: ', num1, '  Second: ', num2);
end.
```

Common Programming Errors

Syntax/compile errors

Runtime errors

Logic errors

Another Use For Readln

As an input prompt

e.g.,

```
writeln('To continue press return');

readln;
```
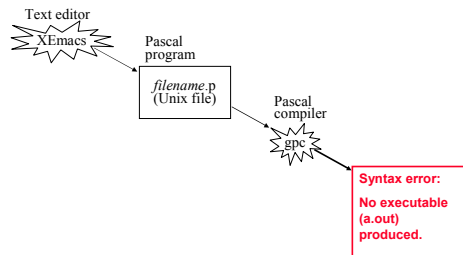

Syntax/Compile Errors

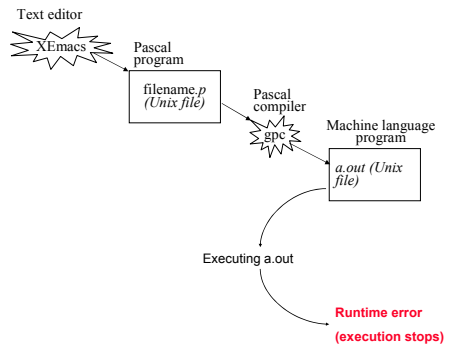Text editor
XEmacs

Pascal program
filename.p
(Unix file)

Pascal compiler
gpc

Syntax error:

No executable (a.out) produced.

Performing Calculations

Operation	Symbol (Operator)
Addition	+
Subtraction	-
Multiplication	*
Real number division	/
Integer division	DIV
Remainder (modulo)	MOD

Runtime Errors

Text editor
XEmacs

Pascal program
filename.*p*
(Unix file)

Pascal compiler
gpc

Machine language program
a.out (Unix file)

Executing a.out

Runtime error

(execution stops)

Logic Errors

Text editor

XEmacs

Pascal program

filename.*p*
(Unix file)

Pascal compiler

gpc

Machine language program

a.out (Unix file)

Executing a.out

Program finishes executing but it may produce an incorrect result

---
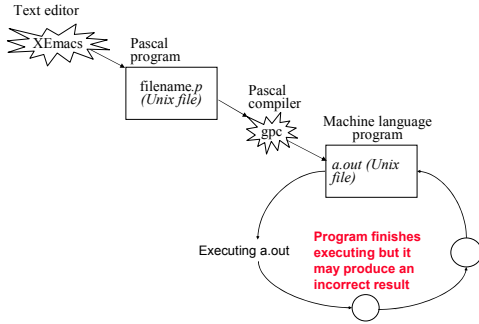
Bottom Up Design

1. Start implementing a solution without creating a structure or plan.

   Here is the first of my many witty anecdotes, it took place in a Paris cafe…

Approaches To Program Design

1. Top down
   - Plan out your approach prior to working on the details of your solution.

2. Bottom up
   - Immediately start working on details of your solution without any sort of structure for your approach.

You Should Now Know

What are different the types of translators and the differences between them

What is the basic structure of a Pascal program

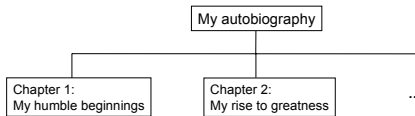How to create, compile and run Pascal programs on the Computer Science network

Variables:
- What are they and what are they used for
- How to set aside memory for one through a declaration
- How to access and change their values
- Conventions for naming variables

Top Down Design

1. Start by outlining the major parts (structure)

   My autobiography

   Chapter 1: My humble beginnings

   Chapter 2: My rise to greatness

   ...

2. Then implement the solution for each part

   Chapter 1: My humble beginnings

   It all started seven and one score years ago with a log-shaped work station…

You Should Now Know (2)

Constants:
- What are named constants and how do they differ from variables
- How to declare a constant
- What are the benefits of using constants

Output:
- How to display text messages or the value of variables onscreen with write and writeln
- How to format the output of a program

Input:
- How to get a program to acquire and store information from the user of the program
- What is the difference between read and readln

You Should Now Know (3)

How are common mathematical operations performed in Pascal.

What are the three common programming errors, when do they occur and what is the difference between each one.

What is the difference between top down and bottom up design.