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

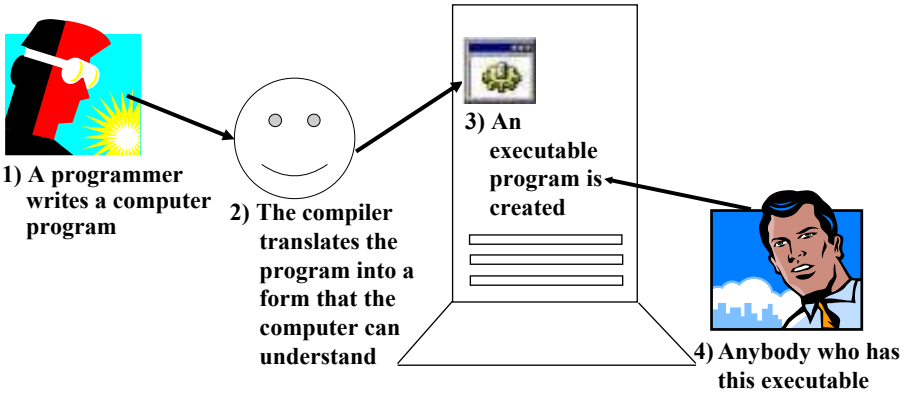
Pascal operators

Common programming errors

Introduction to program design

Computer Programs

Binary is the language of the computer



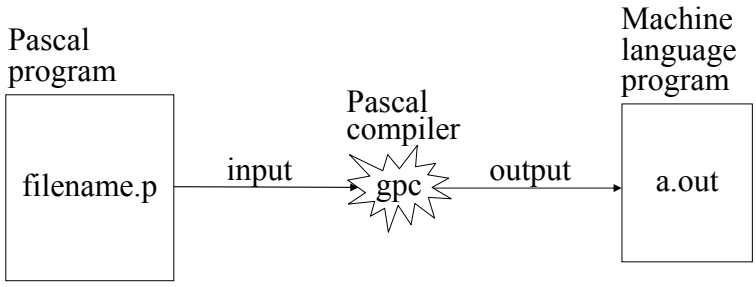
Translators

Convert computer programs to machine language

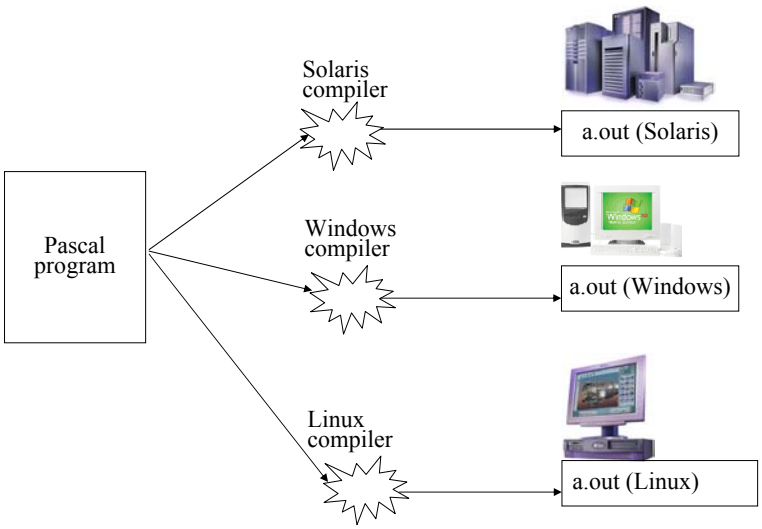
Types

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 right 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.

Compiling Programs: Basic View



Compiling Programs On Different Operating Systems Produces Different Executables



Basic Structure Of Pascal Programs

Program name.p (Pascal source code)

Part I: Header

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

Part II: Declarations

```
const
:
```

Part III: Statements

```
begin
  :
end.
```

Details Of The Parts Of A Pascal Program

Part I: Header

- 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
     - Keyword: program, 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.
*)

program taxIt (input, output);
```

Documentation

Heading

Details Of The Parts Of A Pascal Program (2)

Part II: Declarations

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

Part III: 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

The Smallest Pascal Program

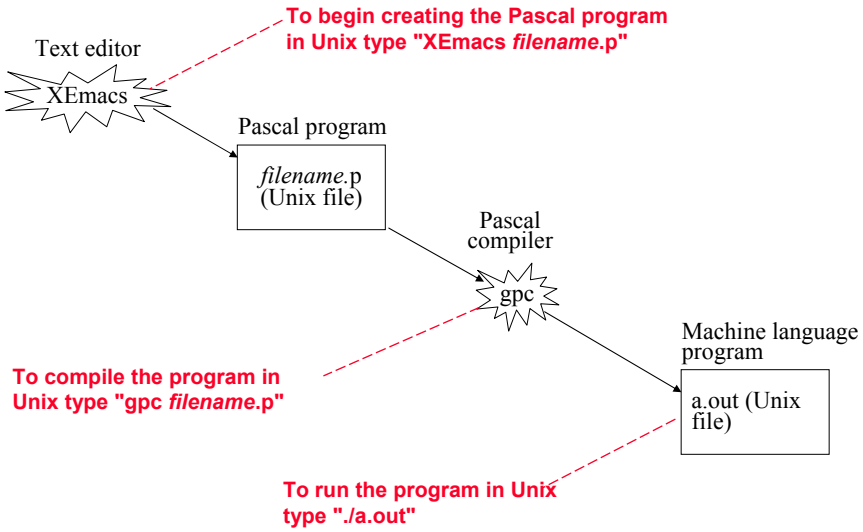
```
program smallest;

begin

end.
```

Note: The name in the header "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").

Creating And Compiling Programs: On The Computer Science Network



Source Code Vs. Executable Files

Source code
- A file that contains the Pascal program code.
- It must end with a 'dot-p' suffix (program name.p).
- Can be viewed and edited.
- Cannot be executed.

```
program smallest;
begin
   :       :
end.
```

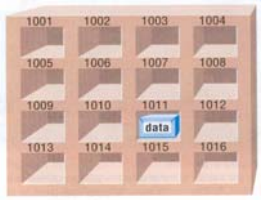
Executable code
- A file that contains machine language (binary) code.
- By default this file will be called "a.out".
- It cannot be directly viewed or edited (meaningless).
- It can be executed.

```
ELF^A^B^A^@^@^
@^@^@^@^@^@^
@^@^B^@^B^@^@
^@^A^@^A^Zh^@^
@^@4^@^B\263\37
0^@^@^@^@^@4^
@
^@^E^@(^@^]^@^Z
^@^@^@^F^@^@^\
   :       :
```


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

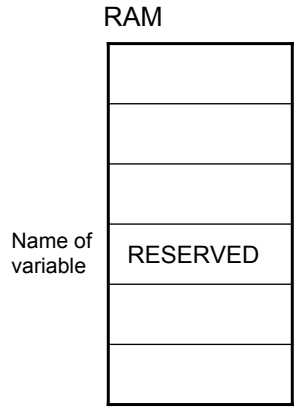
Usage (must be done in this order!)
- Declaration
- Accessing or assigning values to the variables

Picture from Computers in your future by Pfaffenberger B

Declaring Variables

Sets aside memory

Memory locations are addressed through the name of the variable



Declaring Variables

Declare variables between the 'begin' and 'end'.

Part I: Header

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

Part II: Declarations

```
const
:
```

Part III: Statements

```
begin
    Declare variables here
end.
```

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;
```


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.

Variable Naming Conventions (2)

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

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

For more information on reserved words go to the url: http://www.gnu-pascal.de/gpc/index.html

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

For more information on standard identifiers go to the url: http://www.gnu-pascal.de/gpc/index.html

Standard Identifiers (2)

Predefined functions

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

For more information on standard identifiers go to the url: http://www.gnu-pascal.de/gpc/index.html

James Tam

Standard Identifiers (3)

Predefined procedures

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

For more information on standard identifiers go to the url: http://www.gnu-pascal.de/gpc/index.html

Accessing Variables

Can be done by referring to the name of the variable

Format:

name of variable

Example:

num

Assigning Values To Variables

Format:

Destination := Source; [1]

Example:

```
var grade     : integer;
var median    : integer;
var age       : integer;
var principle : real;
var rate      : real;
var principle : real;
var interest  : real;
var initial   : char;

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

1 The source can be any expression (constant, variable or mathematical formula)

Assigning Values To Variables (2)

Avoid assigning mixed types:

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

   num1 := 12;
   num2 := 12.5;
   num2 := num1;
   num1 := num2;

end.
```

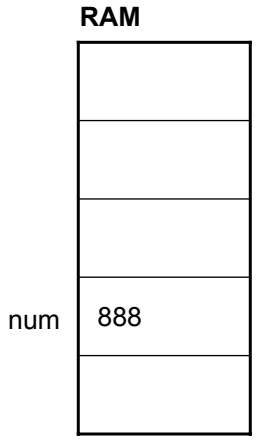
Rare

Not allowed!

Reminder: Variables Must First Be Declared Before They Can Be Used! (The Right Way)

Correct:

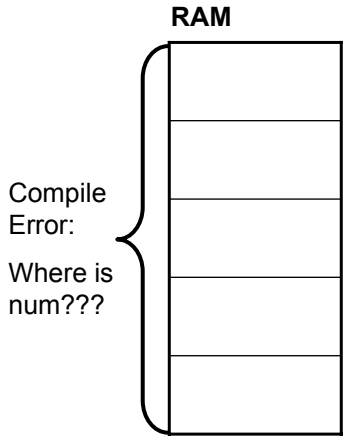
```
var num : integer;
num := 888;
```



Reminder: Variables Must First Be Declared Before They Can Be Used! (The Wrong Way)

Incorrect:

```
num := 888;
var num : integer;
```



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 generally apply 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;
```

Declaring Named Constants

Named constants are declared in the declarations section

Part I: Header

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

Part II: Declarations

```
const
    Declare constants here
```

Part III: Statements

```
begin
   :   :
end.
```


Purpose Of Named Constants

1) Makes the program easier to understand

```
populationChange := (0.1758 – 0.1257) * currentPopulation;
```

Vs.

Magic Numbers (avoid whenever possible!)

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

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.

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 (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

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'…);
```


Output (2)

Example:

```
program simple (output);
begin
    writeln(‘This it it.’);
end.
```

Output (3)

Example:

```
program outputExample (output);
begin
 var num : integer;
 num := 10;
 writeln('line1');
 write('line2A');
 writeln('line2B');
 writeln(num);
 writeln('num=', num);
end.
```


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);
```

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 the number of decimal places less than the actual number of decimal places – number will be rounded up.
- Example One:

```
num1 := 123.4567;
writeln (num1:6:2);
```

- Set the number of decimal places greater than the actual number of decimal places – number will be padded with zeros.
- Example Two:

```
num1 := 123.4567;
writeln(num1:6:6);
```

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

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 (2)

Examples:

```
program inputExampleOne (input);
begin
    var num1 : integer;
    var num2 : integer;
    read (num1);
    read (num2);
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: 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('Enter a number: ');
  read(num1);
  write('Enter a number: ');
  read(num2);
  writeln('You entered these numbers: '
          '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('Enter a number: ');
  readln(num1);
  write('Enter a number: ');
  readln(num2);
  writeln('You entered these numbers: '
          'First: ', num1, '  Second: ', num2);
end.
```

Another Use For Readln

As an input prompt

e.g.,

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


Input Testing

```
program inputChecking (input, output);

begin

   var num : integer;

   var ch  : char;

   write('Enter number and a character: ');

   read(num,ch);

   writeln('num:', num, '-ch:', ch, '-');

end.
```

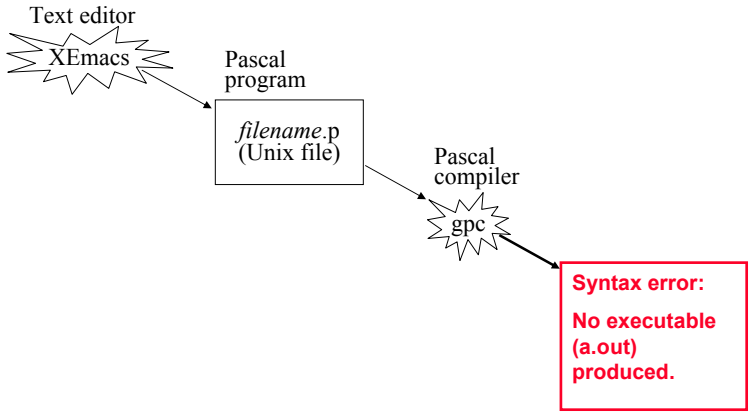
Performing Calculations

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

Common Programming Errors

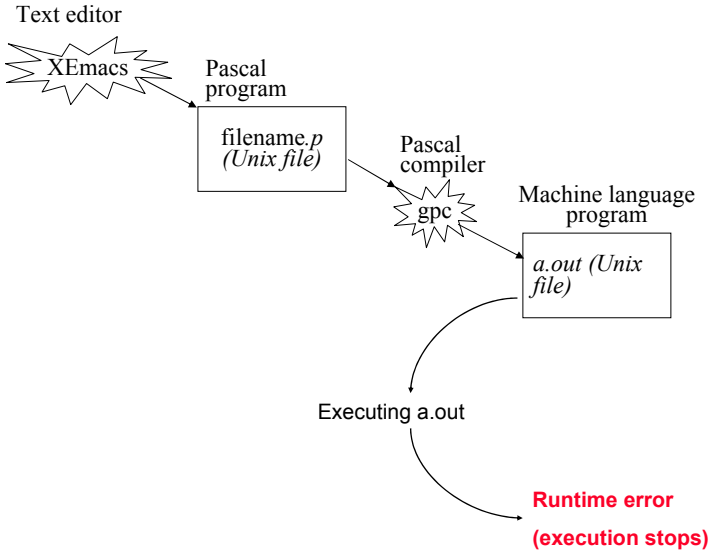
1. Syntax/compile errors
2. Runtime errors
3. Logic errors

1. Syntax/Compile Errors



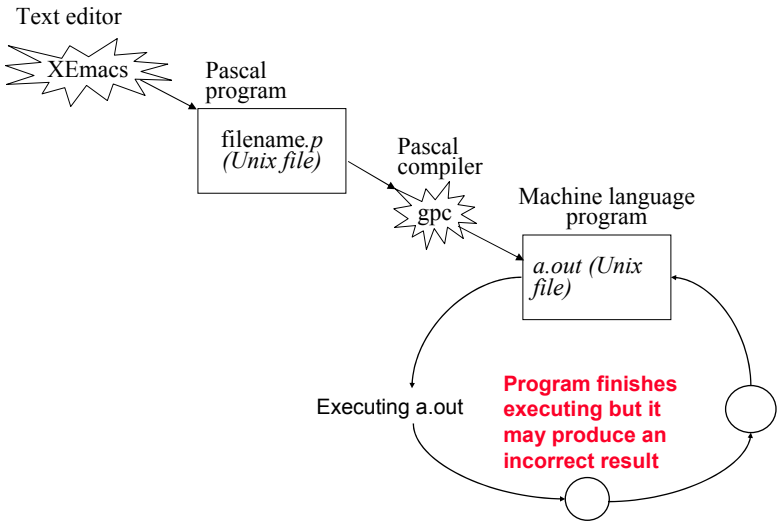
James Tam

2. Runtime Errors



James Tam

3. Logic Errors

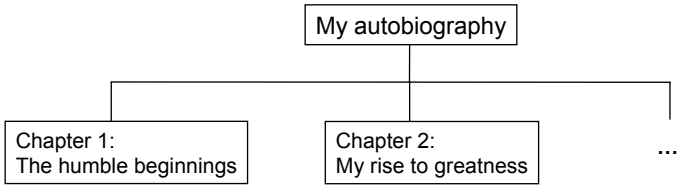


Approaches To Program Design

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.

Top Down Design

1. Start by outlining the major parts (structure)



2. Then implement the solution for each part

Chapter 1: The humble beginnings

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

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 “Tim Horton’s” in Balzac..

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 a variable through a declaration
- How to access and change their values
- Conventions for naming variables

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 or constants 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
- How to perform input checking

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.