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 and problem solving

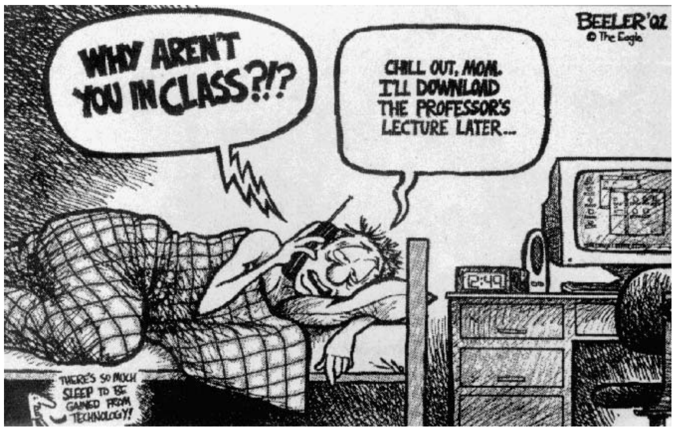
Reminder: About The Course Textbook

•It's recommended but not a required purchase.

•However the course notes are required for this course

Reminder: How To Use The Course Resources

- They are provided to support and supplement this class.
- Neither the course notes nor the text book are meant as a substitute for regular attendance to lecture and the tutorials.


BEELER'02 © The Eagle
WHY AREN'T YOU IN CLASS?!?
CHILL OUT, MOM. I'LL DOWNLOAD THE PROFESSOR'S LECTURE LATER...
THERE'S SO MUCH SLEEP TO BE GAINED FROM TECHNOLOGY!
12:40

James Tam

Reminder: How To Use The Course Resources (2)

```
procedure add (var head        : NodePointer;
               var newNode : NodePointer);
var
  temp : NodePointer;
begin
  if (head = NIL) then
    head := newNode
  else
  begin
    temp := head;
    while (temp^.next <> NIL) do
      temp := temp^.next;
    temp^.next := newNode;
  end;
  newNode^.next := NIL;
end;
```

James Tam

Reminder: How To Use The Course Resources (2)

```
procedure add (var head       : NodePointer;
               var newNode : NodePointer);
var
   temp : NodePointer;
begin
   if (head = NIL) then
      head := newNode
   else
   begin
      temp := head;
      while (temp^.next <> NIL) do
         temp := temp^.next;
      temp^.next := newNode;
   end;
   newNode^.next := NIL;
end;
```

If you miss a class make sure that you catch up on what you missed (get someone's class notes)

…when you do make it to class make sure that you supplement the slides with your own notes (cause you aint gonna remember it in the exams if you don't)

But Once You've Made An Attempt To Catch Up

- Ask for help if you need it
- There are no dumb questions

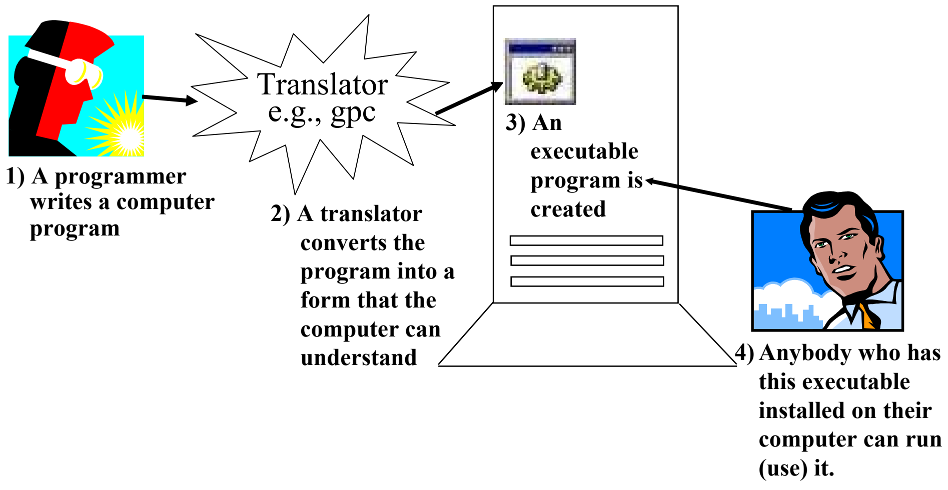
Image from "The Simpsons" © Fox

Don’t Forget: How To Succeed In This Course

1. Practice things yourself
2. Make sure that you keep up with the material
3. Look at the material before coming to lecture
4. Start working on things early

Computer Programs

Binary is the language of the computer


Translator
e.g., gpc
1) A programmer writes a computer program
2) A translator converts 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.

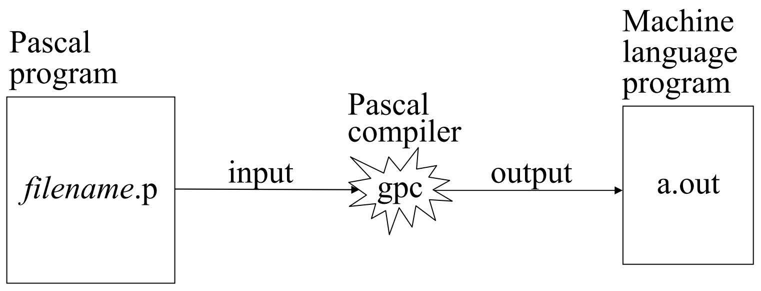
Translators

Convert computer programs to machine language

Types

1) Interpreters
   - Each time that 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

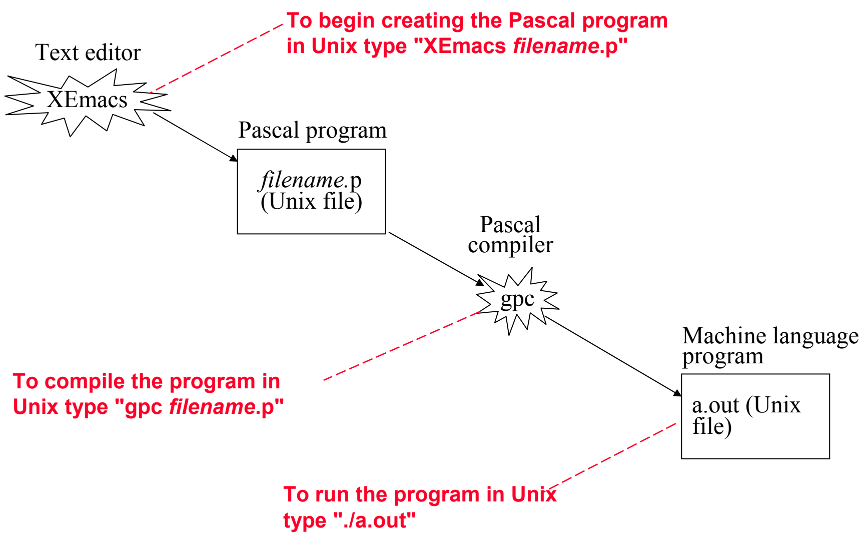


The Smallest Pascal Program

```
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


To begin creating the Pascal program in Unix type "XEmacs *filename*.p"
Text editor
XEmacs
Pascal program
filename.p (Unix file)
Pascal compiler
gpc
Machine language program
a.out (Unix file)
To compile the program in Unix type "gpc *filename*.p"
To run the program in Unix type "./a.out"

Source Code Vs. Executable Files

Source code (e.g., smallest.p file)

- 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 (often it's the "a.out" file)

- 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^@^@^\
   :        :
```

James Tam

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.
```

James Tam

Details Of The Parts Of A Pascal Program

Part I: Header

- Parts:
  1) Program documentation
     - 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.
 *  This program will only allow you to complete a Canadian tax return.     Documentation
*)


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

Program Documentation

Program documentation: Used to provide information about a computer program to another *programmer*:

- Often written inside the same file as the computer program (when you see the computer program you can see the documentation).
- The purpose is to help other programmers understand how the program code was written: how it works, what are some of it's limitations etc.

User manual: Used to provide information about how to use a program to *users* of that program:

- User manuals are traditionally printed on paper but may also be electronic but in the latter case the user manual typically takes the form of electronic help that can be accessed as the program is run.
- The purpose is to help users of the program use the different features of the program without mention of technical details.

Program Documentation (2)

- It doesn't get translated into binary.
- It doesn't contain instructions for the computer to execute.
- It is for the reader of the program:
  - What does the program do e.g., tax program.
  - What are it's capabilities e.g., it calculates personal or small business tax.
  - What are it's limitations e.g., it only follows Canadian tax laws and cannot be used in the US.
  - What is the version of the program
    - If you don't use numbers for the different versions of your program then consider using dates.
  - How does the program work.
    - This is often a description in English (or another high-level) language that describes the way in which the program fulfills its functions.
    - The purpose of this description is to help the reader quickly understand how the program works

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

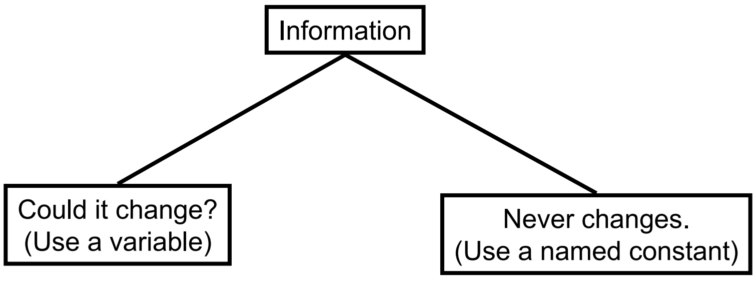
Part III: Statements

- 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 "**;**"
- Example statements: display a message onscreen, prompt the user for input, open a file and write information to that file etc.
- Much more to come later throughout the rest of the term regarding this section

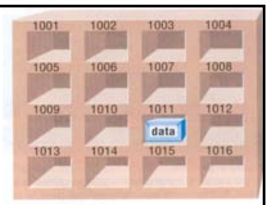
Performing Calculations

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

Storing Information



Variables



Set aside a location in memory
- This location can store one 'piece' of information

Used to store information (temporary)
- *At most* the information will be accessible as long as the program runs

Types:
- **integer** – whole numbers
- **real** – whole numbers and fractions
- **char** – a single character: alphabetic, numeric and miscellaneous symbols (in UNIX type "man ascii")
- **boolean** – a true or false value

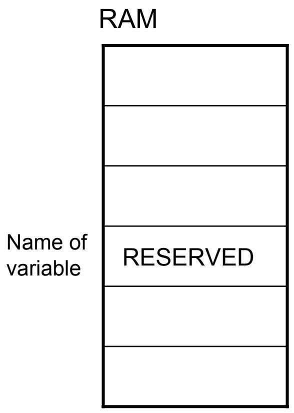
Usage (must be done in this order!)
- Declaration
- Accessing or assigning values to the variables

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 (just after the ‘begin’

end**.**

Declaring Variables (3)

Format:

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

Example:

The full example can be found in UNIX under: /home/231/examples/intro/variableExample1.p (variableExample1 for the compiled version).

```
program variableExample1;
begin
  var height  : real;
  var weight : real;
  var age     : integer;
end.
```

Variable declaration

Global Variables

- Variables declared outside of the begin-end pair.

```
program anExample;
var num1 : integer;
begin
    var num2 : integer;
end.
```

Global variable: DON’T DO IT THIS WAY

Non-global variable (local variable): DO IT THIS WAY

- For now avoid doing this (additional details will be provided later in the course): generally this is regarded as bad programming style.

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.

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

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

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

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:

The full example can be found in UNIX under: /home/231/examples/intro/variableExample2.p (variableExample2 for the compiled version).

```
program variableExample2;
begin
  var height  : real;
  var weight : real;
  ~~var age      : integer;~~
  ~~weight := height * 2.2;~~
end.
```

NO!

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

Assigning Values To Variables (2)

```
program variableExample2;
begin
  var height  : real;
  var weight : real;
  var age      : integer;
  height := 69;
  weight := height * 2.2;
end.
```

$\left.\begin{array}{l}\text{height := 69;}\\\text{weight := height * 2.2;}\end{array}\right\}$ **A better approach**

Important lesson: **ALWAYS** initialize your variables to some default starting value before using them.

Assigning Values To Variables (3)

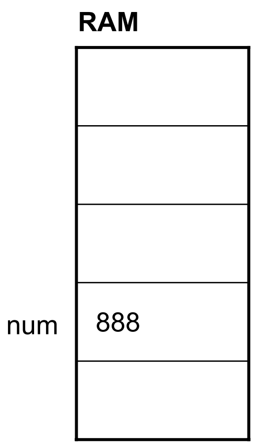
Avoid assigning mixed types:

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

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

    num1 := num2;    Not allowed!

end.
```

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

Correct:

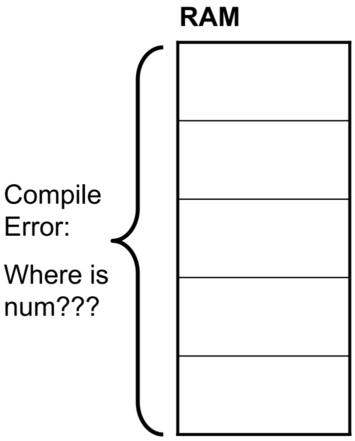
```
program anExample;
begin
  var num : integer;
  num := 888;
end.
```



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

Incorrect:

```
program anExample;
begin
    num := 888;
    var num : integer;
end.
```



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.
```

Named Constants: A Compilable Example

```
program anExample;
const
    TAX_RATE = 0.25;
    SAMPLE_SIZE = 1000;
    YES = True;
    NO = False;
    MY_FIRST_INITIAL = ‘J’;
begin
    var grossIncome : real;
    var afterTaxes     : real;
    grossIncome := 100000;
    afterTaxes := grossIncome – (grossIncome * TAX_RATE);
end.
```

Purpose Of Named Constants

1) Makes the program easier to understand

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

Magic Numbers (avoid whenever possible!)

Vs.

```
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.
```

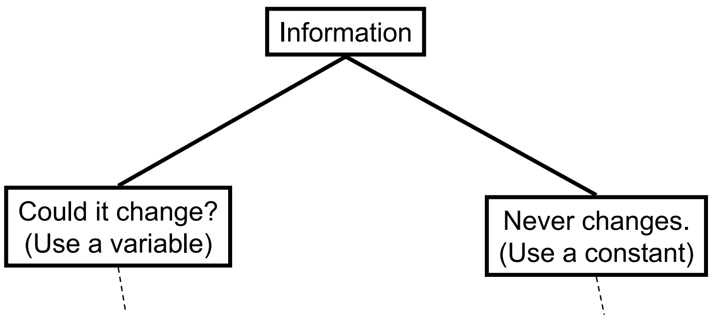
Purpose Of Named Constants (3)

```
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.
```

Storing Information


Information
Could it change? (Use a variable)
Never changes. (Use a constant)

- Fractional numbers: use a “real”
- Whole numbers: use an “integer”
- Single character: use a “char”
- Only true or false: use a “boolean”

- Fractional numbers: use a “real”
- Whole numbers: use an “integer”
- Single character: use a “char”
- Only true or false: use a “boolean”

James Tam

Output

•Displaying information onscreen

•Done via the write and writeln statements
- Write: displays the output and nothing else (the cursor remains on the line)
- Writeln: displays the output followed by a newline (the cursor moves to the next line)

Format (literal string of characters):

```
write (‘a message');
          or
writeln(‘a message');
```

James Tam

Output (2)

Example (literal string of characters):
The complete example can be found in UNIX under:
/home/231/examples/intro/outputExample1.p (outputExample1 for the compiled version).

```
program outputExample1 (output);
begin
   write('line1');
   writeln('line2');
   write('line3');
end.
```

Style convention

James Tam

Output Of The Contents Of Variables And Constants

Format:

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

James Tam

Output Of The Contents Of Variables And Constants (2)

Example:

The complete example can be found in UNIX under: /home/231/examples/intro/outputExample2.p (outputExample2 for the compiled version).

```
program outputExample2 (output);
const
    ACONSTANT = 888;
begin
    var num : integer;
    num := 7;
    writeln(ACONSTANT);
    writeln(num);
end.
```

Mixed Output

It’s possible to display literal strings of characters and the contents of variables and constants with a single write or writeln statement.

Format:

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

Mixed Output (2)

Example:

The complete example can be found in UNIX under: /home/231/examples/intro/outputExample3.p (outputExample3 for the compiled version).

```
program outputExample3 (output);
const
  ACONSTANT = 888;
begin
  var num : integer;
  num := 7;
  writeln('ACONSTANT: ', ACONSTANT);
  writeln('num=', num);
end.
```

Output: How Do You Make It Look Nice?

P1: How to make output line align/justify from line-to-line?
A1: Set the field width parameter

P2: How to specify the number of places of precision for the output of real numbers?
A2: Set the parameter for the number of places of precision (only works for real numbers)

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/floating point form.

Manually formatting of output:

Format:

write or writeln (<data>: <*Field width for data*[1]>: <*Number decimal places for real data*[1]>);

Examples:

```
var num : real;
num := 12.34;
writeln(num);
writeln(num:5:2);
```

1 These values can be set to any non-negative integer (zero or greater).

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 integer variables **but not** for other types of data.
- Examples:

```
var num : integer;
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:

```
var num : integer;
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.

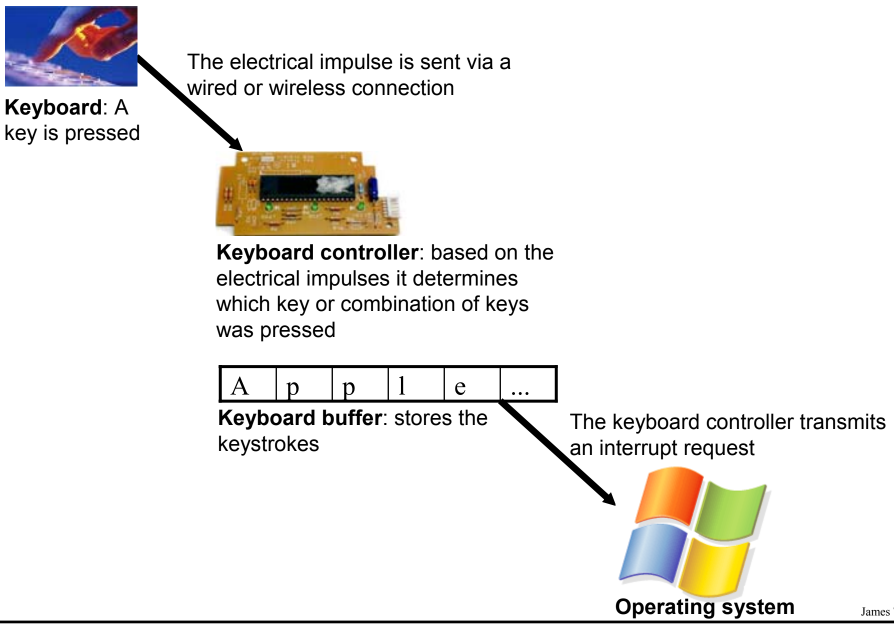
- Set the number of decimal places less than the actual number of decimal places – the number will be rounded up.
- Example One:

```
var num : real;
num := 123.4567;
writeln (num:6:2);
```

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

```
var num : real;
num := 123.4567;
writeln(num:6:6);
```

Recall: How Keyboard Input Works



Keyboard: A key is pressed

The electrical impulse is sent via a wired or wireless connection

Keyboard controller: based on the electrical impulses it determines which key or combination of keys was pressed

A	p	p	l	e	...

Keyboard buffer: stores the keystrokes

The keyboard controller transmits an interrupt request

Operating system

Recall: How Keyboard Input Works

Operating system:

Q: Is the key combination a (an operating) system level command e.g., <alt>-<ctrl>-<del>?

Yes

Execute operating system instruction

No

Pass the key combination onto current application

James Tam

Input

The computer program getting information from the user

Done via the read and readln statements

Format:

read (<*name of variable to store the input*>);

or

readln (<*name of variable to store the input*>);

James Tam

Input (2)

Example:

```
program inputExampleOne (input, output);
begin
    var num : integer;
    write('Enter an integer: ');
    readln (num);
end.
```

A common style convention

Input: Read Vs. Readln

Both:

- Reads each value entered and matches it to the corresponding variable.
  - e.g., read (num)
  - If num is an integer then the read statement will try to read an integer value from the user's keyboard input.

Read

- If the user inputs additional values before hitting enter, the additional values will remain in the buffer.

Readln

- Any additional values entered before (and including) the enter key will be discarded.

Read: Effect On The Keyboard Buffer

Pascal program

```
program getInput (input, output);
begin
   var num : integer;
   write('Enter an integer: ');
   read(num);
end.
```

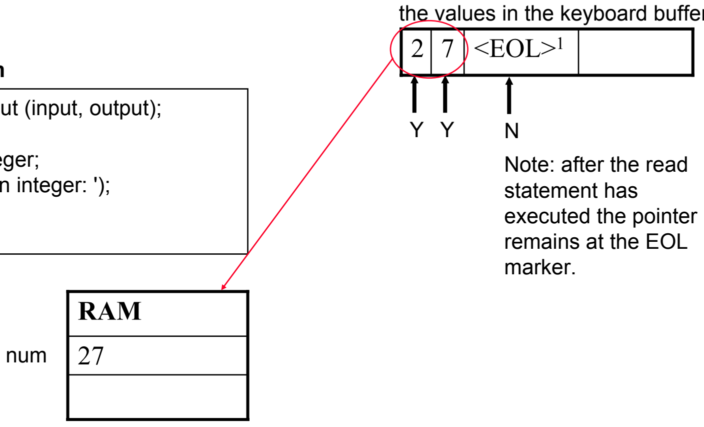
Keyboard: user types in 27 and hits enter

Read: Effect On The Keyboard Buffer (2)

Keyboard controller: determines which keys were pressed and stores the values in the keyboard buffer

Pascal program

```
program getInput (input, output);
begin
   var num : integer;
   write('Enter an integer: ');
   read(num);
end.
```



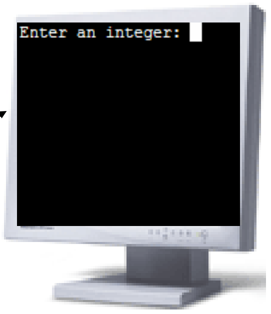
Note: after the read statement has executed the pointer remains at the EOL marker.

1 When the user presses the enter key it is stored as the EOL (end-of-line) marker. The EOL marker signals to the Pascal program that the information has been typed in and it will be processed.

Readln: Effect On The Keyboard Buffer

Pascal program

```
program getInput (input, output);
begin
   var num : integer;
   write('Enter an integer: ');
   readln(num);
end.
```



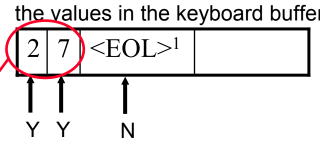
```
Enter an integer:
```

Keyboard: user types in 27 and hits enter

James Tam

Readln: Effect On The Keyboard Buffer (2)

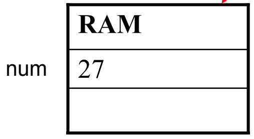
Keyboard controller: determines which keys were pressed and stores the values in the keyboard buffer



Note: Unlike read, the readln will move the pointer past the EOL marker (input buffer is emptied and ready for new input).

Pascal program

```
program getInput (input, output);
begin
   var num : integer;
   write('Enter an integer: ');
   readln(num);
end.
```



1 When the user presses the enter key it is stored as the EOL (end-of-line) marker. The EOL marker signals to the Pascal program that the information has been typed in and it will be processed.

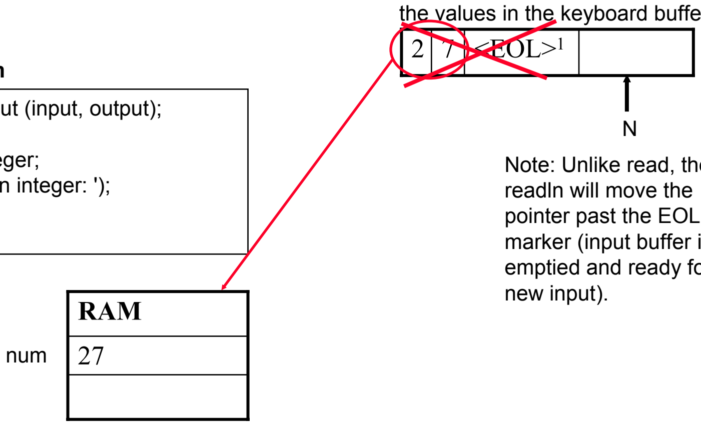
James Tam

Readln: Effect On The Keyboard Buffer (2)

Keyboard controller: determines which keys were pressed and stores the values in the keyboard buffer

Pascal program

```
program getInput (input, output);
begin
   var num : integer;
   write('Enter an integer: ');
   readln(num);
end.
```



Note: Unlike read, the readln will move the pointer past the EOL marker (input buffer is emptied and ready for new input).

1 When the user presses the enter key it is stored as the EOL (end-of-line) marker. The EOL marker signals to the Pascal program that the information has been typed in and it will be processed.

Read Vs. Readln

•If no input is read in by the program after a 'read' or 'readln' statement then both approaches appear identical (the effect of the pointer staying or moving past the EOL marker has no visible effect).

```
program getInput (input, output);
begin
   var num : integer;
   write('Enter an integer: ');
   readln(num);
end.
```

After this readln the program ends and the keyboard buffer is emptied.

•**Caution!** If the 'read' or 'readln' statement is followed by another read or readln then the effect of the extra input remaining in the keyboard buffer can have unexpected consequences!

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 num : integer;
   var ch  : char;
   write('Enter a number: ');
   read(num);
   write('Enter a character: ');
   read(ch);
   writeln('You entered num: ', num, '  ch: ', ch);
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 num : integer;
   var ch  : char;
   write('Enter a number: ');
   readln(num);
   write('Enter a character: ');
   readln(ch);
   writeln('You entered num: ', num, '  ch: ', ch);
end.
```

General Rule Of Thumb: Use Readln!

When getting input from the user unless there's a compelling reason you should use 'readln' rather than 'read'.

(This is an important point: forget at your own peril!)

General Rule Of Thumb

The prompt that requests user input should take the form of a write rather than a writeln:

```
var num : integer;
write('Enter your age: ');
readln(age);
```

Vs.

```
var num : integer;
writeln ('Enter your age: ');
readln(age);
```

Another Use For Readln

As an input prompt

e.g.,

```
  writeln('To continue press enter');
  readln;
  writeln('The rest of the program continues..');
```

When this statement is reached the program will pause and wait for input from the user.

Testing Inputs

```
program inputChecking (input, output);
begin
  var num : integer;
  var ch    : char;
  write('Enter a number and a character: ');
  read(num, ch);
  writeln('num:', num, '-ch:', ch, '-');
end.
```

Common Programming Errors

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

1. Syntax/ Compilation Errors

Each language has rules about how statements are to be structured.

English sentence is structured by the grammar of the English language:

- The cat sleeps the sofa.

Grammatically incorrect: missing the preposition to introduce the prepositional phrase 'the sofa'

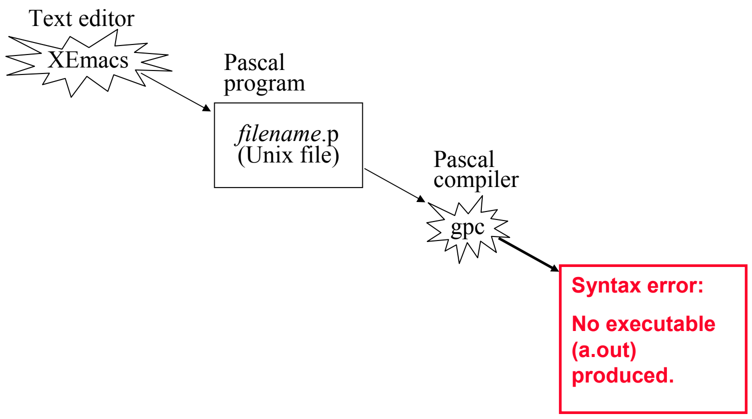
Pascal statements are structured by the syntax of the programming language:

- $\underbrace{5 :=}$ num

Syntactically incorrect: the left hand side of an assignment statement cannot be a literal constant.

1. Syntax/Compile Errors (2)

They occur as the program is being compiled

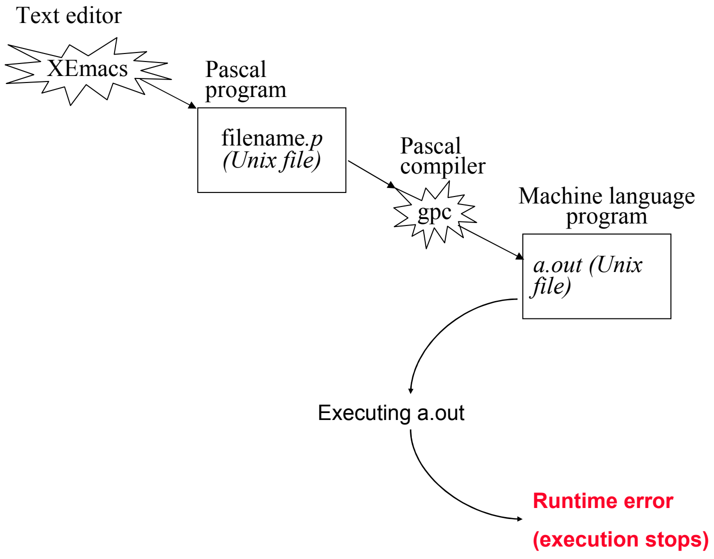


Some Common Syntax Errors

- Missing or mismatching quotes for a write or writeln statement
- Forgetting to separate statements with a semi-colon
- Forgetting the name of the program in the header
- Forgetting the period at the end of the program
- Using identifiers (such as variables or constants) before they've been declared
- Forgetting keywords such as 'program', 'begin' and 'end'

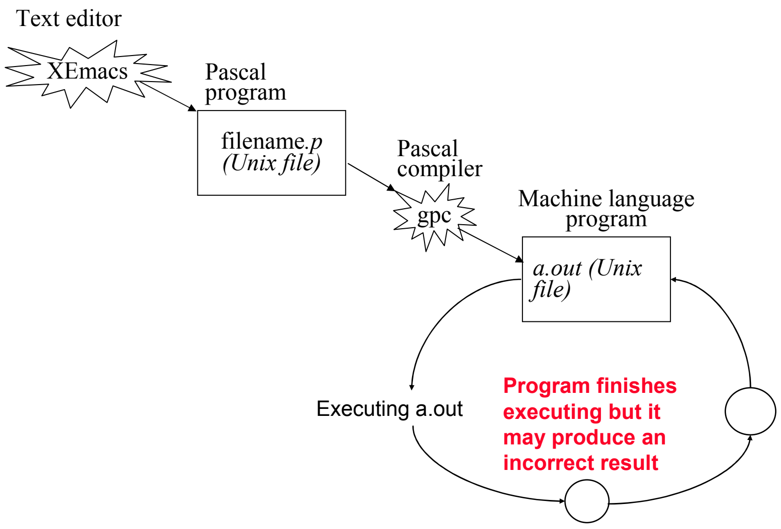
2. Runtime Errors

They occur as the program is running and cause execution to stop



3. Logic Errors

They occur as the program is running, the program doesn't abruptly end.



Problem Solving Example: Making Change

(Paraphrased from the book "Pascal: An introduction to the Art and Science of Programming" by Walter J. Savitch.

Problem statement:

Design a program to make change. Given an amount of money, the program will indicate how many quarters, dimes and pennies are needed. The cashier is able to determine the change needed for values of a dollar and above.

Actions that may be needed:

- Action 1: Prompting for the amount of money
- Action 2: Computing the combination of coins needed to equal this amount
- Action 3: Output: Display the number of coins needed

Program Design: An Example Problem

- However Action 2 (computing change) is still quite large and may require further decomposition into sub-actions.
- One sensible decomposition is:
  - Sub-action 2A: Compute the number of quarters to be given out.
  - Sub-action 2B: Compute the number of dimes to be given out.
  - Sub-action 2C: Compute the number of pennies to be given out.

Determining What Information Needs To Be Tracked

1. Amount of change to be returned
2. Number of quarters to be given as change
3. Number of dimes to be given as change
4. Number pennies to be given as change
5. The remaining amount of change still left (changes as quarters, dimes and pennies are given out)

How To Come Up With A Solution

1. If you are truly stuck then STEP AWAY from the computer!
2. Try to picture things in terms of something that you can relate to (i.e., not Pascal code) but something in the real world.
    a. Make sure that you understand what the problem truly entails by describing it in terms of what you know e.g., draw pictures, write text descriptions (English), use physical analogies.
    b. Try to work out a solution to the problem in terms of concepts that you are familiar with e.g., draw pictures, write text descriptions (English), use physical analogies.
    c. Then try to translate your solution to program code.
    d. (If you are having trouble going from (b) to (c)) then try to describe the solution in as much detail as possible using a human language. If your solution is detailed enough then it’s often just a matter of working out the syntax when you write program code.

Making Change: Solution

DO NOT LOOK AT THIS SOLUTION BEFORE CLASS!
The full version of this program can be found in UNIX under:
/home/231/examples/intro/change.p

```
program change (input, output);
begin
   var amount      : integer;
   var quarters    : integer;
   var dimes       : integer;
   var pennies     : integer;
   var amountLeft : integer;

   write ('Enter the amount of change from 1 to 99 cents: ');
   readln (amount);
```

Making Change: Solution (2)

```
   (* Quarters *)
   quarters := amount DIV 25;
   amountLeft := amount MOD 25;

   (* Dimes *)
   dimes := amountLeft DIV 10;
   amountLeft := amountLeft MOD 10;

   (* Pennies *)
   pennies := amountLeft;
```

Making Change: Solution (3)

```
  (* Display the results. *)
  writeln ('Original amount: ', amount, ' pennies');
  writeln ('No quarters: ', quarters);
  writeln ('No dimes: ', dimes);
  writeln ('No pennies: ', pennies);
end.
```

Testing The Solution

- What should be tested? (What inputs should be used)
  - Running the program with all possible inputs (time-consuming?)
  - Running the program with a subset of the possible inputs (try to catch all reasonable cases)?
- Not testing the programming or performing minimal testing.
  - This may work for small programs
  - With anything but a trivial sized program, finding the logical errors may be next to impossible unless each portion has undergone a reasonable amount of testing.

You Should Now Know

What is the difference between the two types of translators: compilers and interpreters.

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 the value of a variable
- 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 named constant
- What are the benefits of using a named constant

How are common mathematical operations performed in Pascal.

Output:

- How to display text messages or the value of a memory location (variable or constant) onscreen with write and writeln
- How to format the output of a Pascal 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)

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

An approach for solving simple problems.