

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

Common programming errors

James Tam

Computer Programs

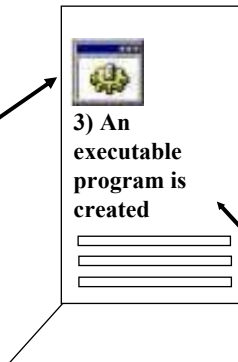
Binary is the language of the computer



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



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

James Tam

Translators

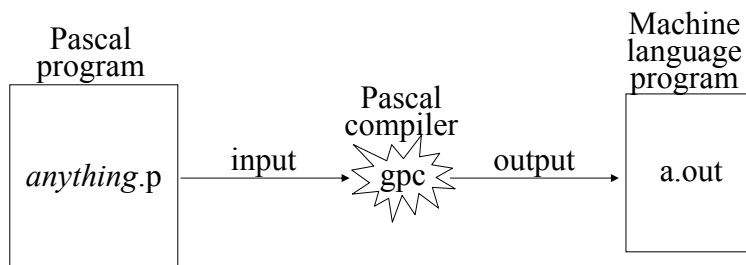
Convert computer programs to machine language

Types

- 1) Interpreters
 - Translate the program as it's executed (a part at a time).
- 2) Compilers
 - Translate the program before it's executed (all at once).

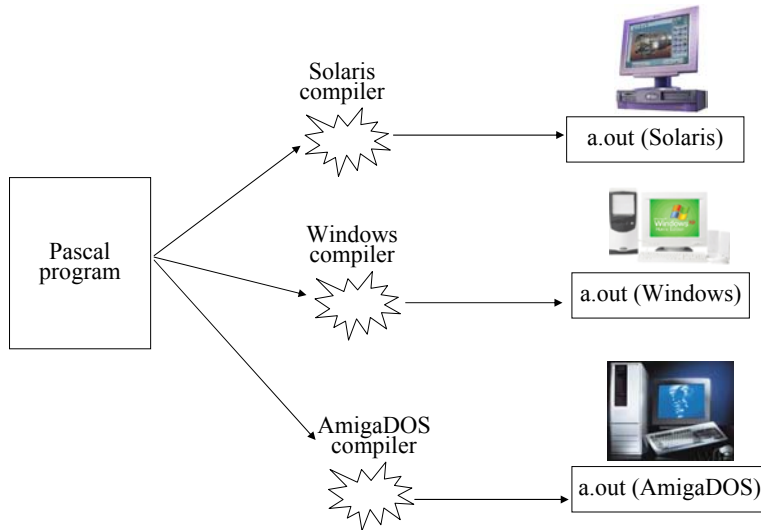
James Tam

Compiling Programs: Basic View



James Tam

Compiling Programs On Different Operating Systems



James Tam

Basic Structure Of Pascal Programs

Header

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

Declarations

```
const  
:
```

Statements

```
begin  
:  
end.
```

James Tam

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.
 - *)

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

James Tam

Details Of The Parts Of A Pascal Program (2)

Declarations

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

Statements

- The instructions in the program that actually gets stuff done
- They tell the computer what to do as the program is running
- Each statement is separated by a semicolon ";"
- Much more to come later in the course

James Tam

The Smallest Pascal Program

program smallest;

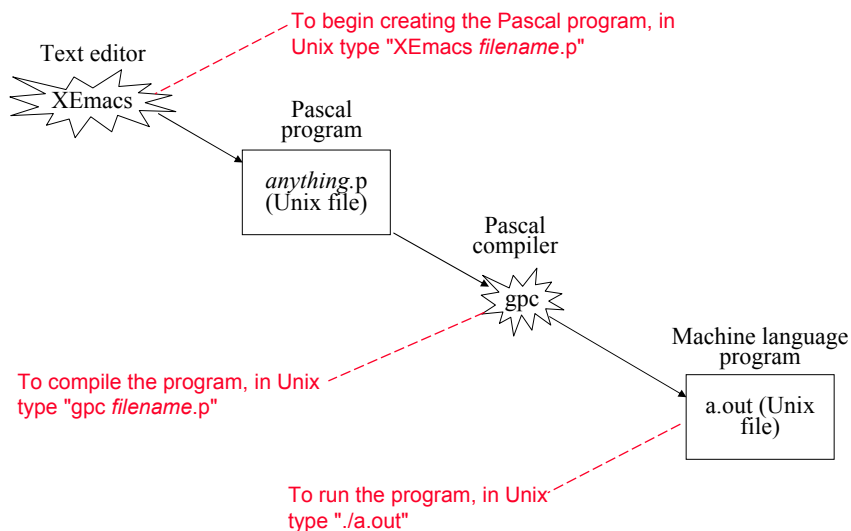
begin

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").

James Tam

Creating And Compiling Programs: On The Computer Science Network



James Tam

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
- char – alphabetic, numeric and miscellaneous symbols
- boolean – true or false values

Usage:

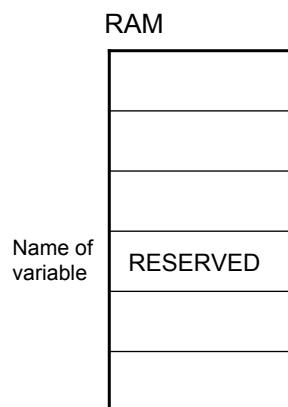
- Declaration
- Accessing or assigning values to the variables

James Tam

Declaring Variables

Sets aside memory

Memory locations addressed through the name



James Tam

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.

James Tam

Declaring Variables (3)

Syntax:

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;

James Tam

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.

James Tam

Variable Naming Conventions (2)

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

James Tam

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>

James Tam

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 identifier go to the url: <http://www.gnu-pascal.de/gpc/index.html>

James Tam

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 identifier 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 identifier go to the url: <http://www.gnu-pascal.de/gpc/index.html>

James Tam

Accessing Variables

Can be done by referring to the name of the variable

Syntax:

name of variable

Example:

num

James Tam

Assigning Values To Variables

Syntax:

Destination := Source; ¹

Example:

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

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

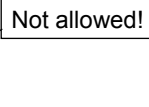
James Tam

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.
```

Not allowed!



James Tam

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).

Syntax:

```
const
```

```
  NAME OF FIRST CONSTANT = value of first constant;
```

```
  NAME OF SECOND CONSTANT = value of second constant;
```

```
  etc.
```

James Tam

Named Constants (2)

Examples:

```
const
```

```
TAX_RATE = 0.25;
```

```
SAMPLE_SIZE = 1000;
```

```
YES = True;
```

```
NO = False;
```

James Tam

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;
```

James Tam

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.

James Tam

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 (BIRTH_RATE > DEATH_RATE) then
    writeln('Growing population')
  else if (BIRTH_RATE < DEATH_RATE) then
    writeln('Shrinking population')
end.
```

James Tam

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 (BIRTH_RATE > DEATH_RATE) then
    writeln('Growing population')
  else if (BIRTH_RATE < DEATH_RATE) then
    writeln('Shrinking population')
end.
```

James Tam

Purpose Of Named Constants (3)

```
program population (output);
const
  BIRTH_RATE = 0.5;
  DEATH_RATE = 0.01;
begin
  var populationChange : real;
  var currentPopulation : real;
  populationChange := (BIRTH_RATE - DEATH_RATE) * currentPopulation;
  if (BIRTH_RATE > DEATH_RATE) then
    writeln('Growing population')
  else if (BIRTH_RATE < DEATH_RATE) then
    writeln('Shrinking population')
end.
```

James Tam

Output

Displaying information onscreen

Done via the write and writeln statements

Syntax:

```
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'...);
```

James Tam

Output (2)

Example:

```
program simple (output);  
begin  
    writeln('The beginning and the end.');
```

```
end.
```

James Tam

Output (3)

Examples:

Begin

```
var num : integer;  
num := 10;  
writeln('line1');  
write('line2A');  
writeln('line2B');  
writeln(num);  
writeln('num=', num);
```

James Tam

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:

Syntax:

```
write or writeln (data: Field width for data: Number decimal places for data);
```

Examples

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

James Tam

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);
```

James Tam

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);
```

James Tam

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.
```

James Tam

Input

The computer program getting information from the user

Done via the read and readln statements

Syntax:

```
read (name of variable);
    or
readln (name of variable);
```

James Tam

Input (2)

Examples:

begin

var num1 : integer;

var num2 : integer;

read (num1);

read (num2);

James Tam

Input: Read Vs. Readln

Both:

- Reads each value inputted and matches it to the corresponding variable.

Read

- If the user inputs additional values before hitting return they will remain

Readln

- Any additional values inputted before the return will be discarded

James Tam

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):

```
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);
```

James Tam

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)

```
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);
```

James Tam

Another Use For Readln

As an input prompt

e.g.,

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

James Tam

Another Input Example

For the complete version of this program look in Unix under:
/home/231/examples/intro/read3.p (or read3 for the compiled version)

```
var ch1 : char;  
var in1 : integer;  
var re1 : real;  
write('Enter an integer, a character and a real number on one line (no spaces): ');  
read(in1);  
write(in1, '-');  
read(ch1);  
write(ch1, '-');  
read(re1);  
writeln(re1);
```

James Tam

Common Programming Errors

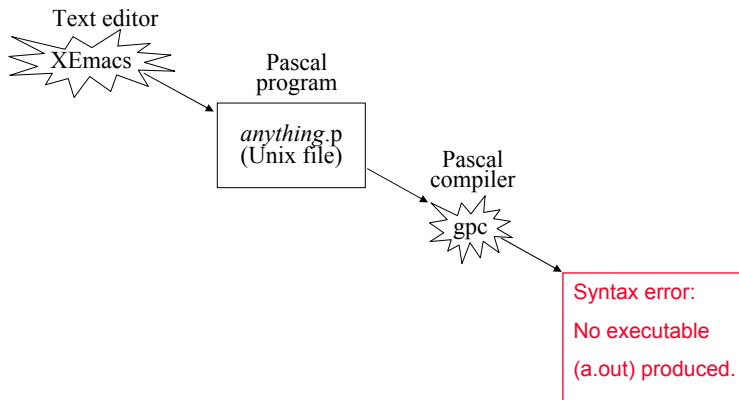
Syntax/compile errors

Runtime errors

Logic errors

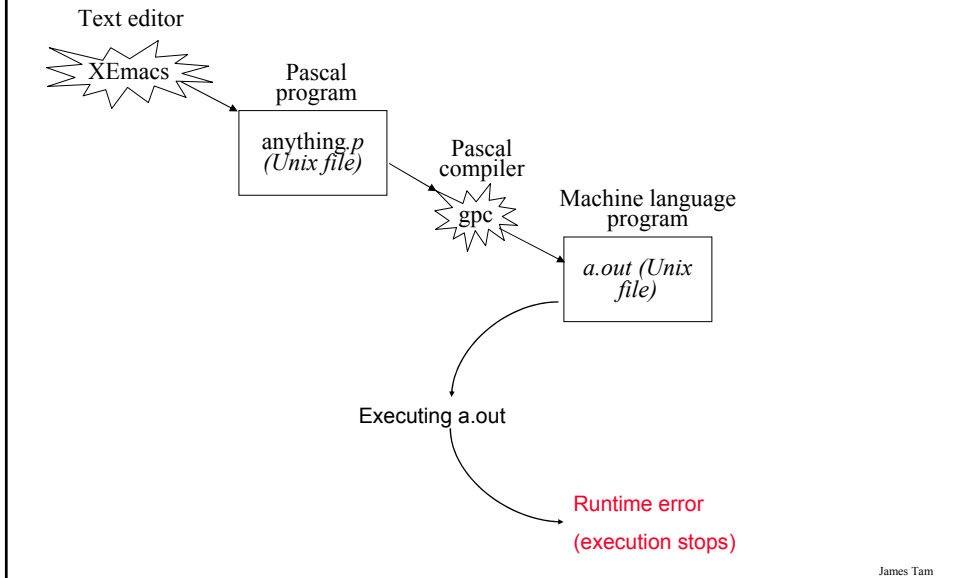
James Tam

Syntax/Compile Errors

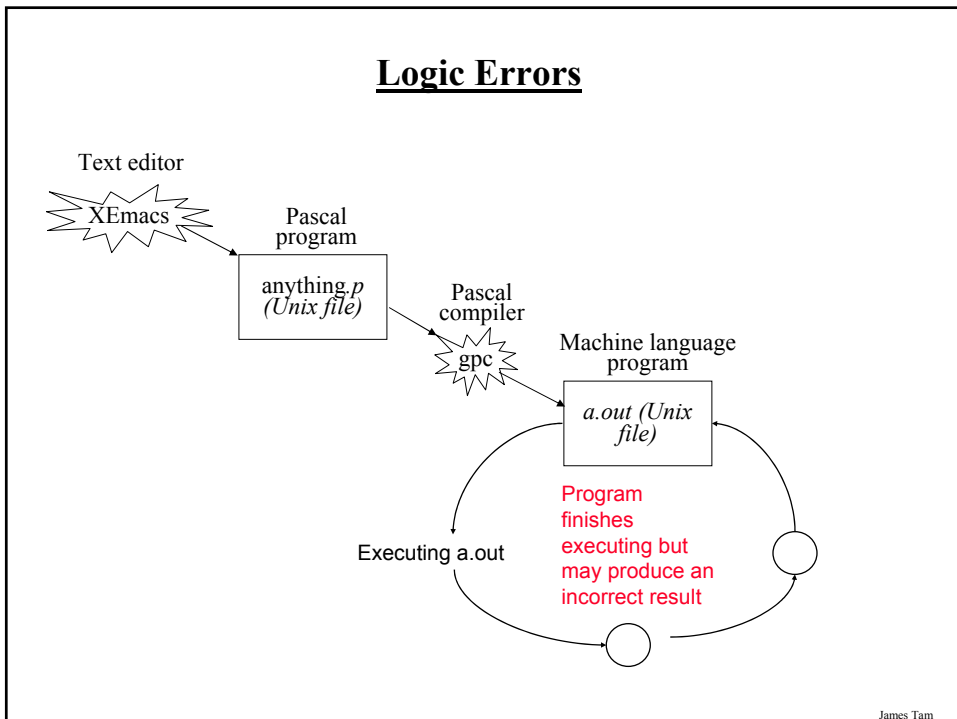


James Tam

Runtime Errors



Logic Errors



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

James Tam

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
- How does the reading of multiple inputs to a computer work

James Tam

You Should Now Know (3)

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