

Pointers

In this section of notes you will learn about another type of variable that stores addresses rather than data

James Tam

Memory: What You Know

- Memory is analogous to a series of slots each of which can store a single piece of information.

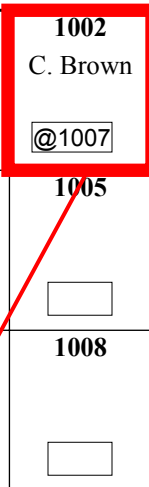
1000 S.F. Adams <input type="text" value="100"/>	1001 S. Bill <input type="text" value="j"/>	1002 C. Brown <input type="text"/>
1003 J. Chan <input type="text" value="4.0"/>	1004 <input type="text"/>	1005 <input type="text"/>
1006 <input type="text"/>	1007 <input type="text"/>	1008 <input type="text"/>

James Tam

Memory: What You Will Learn

- How a memory location can contain the address of another location in memory.

1000 S.F. Adams <input type="text" value="100"/>	1001 S. Bill <input type="text" value="j"/>	1002 C. Brown <input type="text" value="@1007"/>
1003 J. Chan <input type="text" value="4.0"/>	1004 <input type="text"/>	1005 <input type="text"/>
1006 <input type="text"/>	1007 <input type="text" value="1999"/>	1008 <input type="text"/>



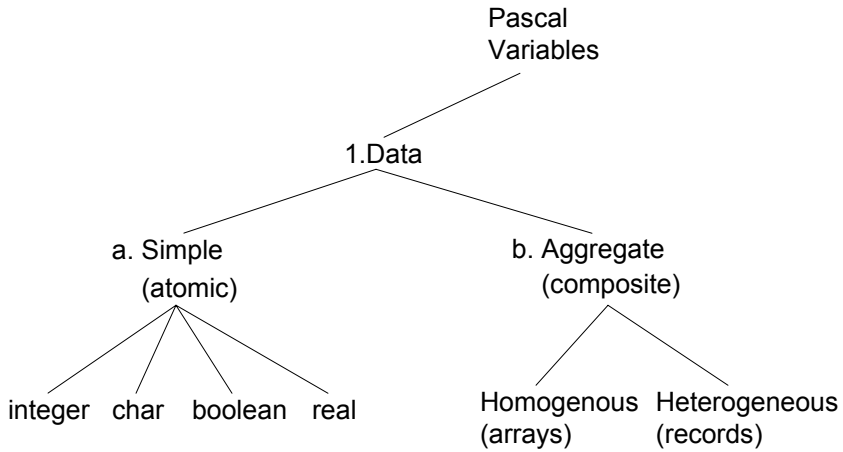
James Tam

Why Bother With Pointers?

The answer to this question will be deferred until the next section of notes (linked lists).

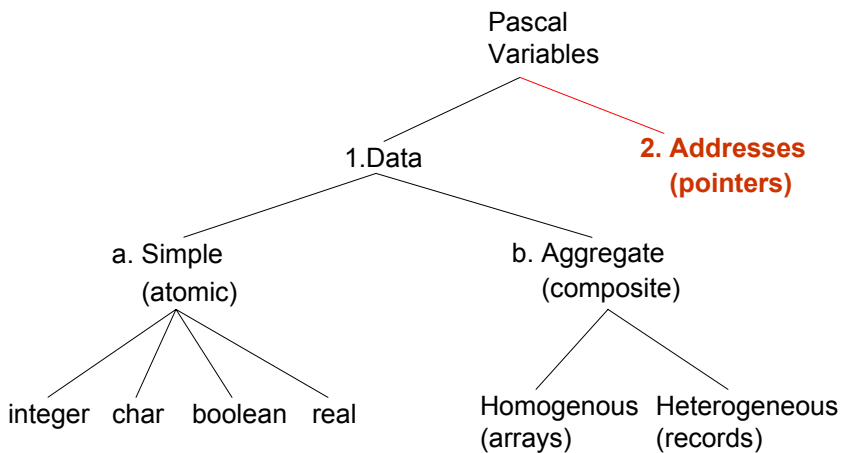
James Tam

Types Of Variables: What You Know



James Tam

Types Of Variables: What You Will Learn



James Tam

Declaration Of Pointer Variables

Format:

```
type
  type name = ^ type pointed to1;
:
:
begin
  var pointer name : type name;
```

Example:

```
type
  IntegerPointer = ^integer;
:
:
begin
  var numPtr1 : IntegerPointer;
```

¹ An alternative is to use the "at-sign" @ instead of the "up-arrow" ^ to declare a pointer variable (*not recommended*)

Allocating Memory For Pointers

- It involves reserving some dynamic memory and having the pointer point to that memory.

Format

```
new (pointer name);
```

Example

```
new (numPtr1);
```

De-Allocating Memory For Pointers

- Returning back the dynamically allocated memory (if it's needed it can then be re-used for something else).

Format

`dispose (pointer name);`

Example

`dispose (numPtr1);`

James Tam

De-Allocating Memory For Pointers: Followup

- Should also be followed by having the pointer no longer point to the memory that has just been de-allocated

Format:

`pointer name := NIL;`

Example:

`numPtr1 := NIL;`

James Tam

Using Pointers

Important! Are you dealing with the pointer or what the pointer is pointing to (allocated memory)?

- Pointer name

- Pointer name ^ (de-reference pointer)

James Tam

Using Pointers

Important! Are you dealing with the pointer or what the pointer is pointing to (allocated memory)?

- Pointer name
pointer

- Pointer name ^ (de-reference pointer)
pointer → variable

James Tam

Accessing Pointers

Format:

(Pointer)

pointer name

(Memory pointed to)

pointer name ^

James Tam

Accessing Pointers (2)

Example:

```
type
  IntegerPointer = ^integer;
:
:
begin
  var numPtr1 : IntegerPointer;
  new(numPtr1);

  (Pointer)
  writeln(numPtr1);

  (Memory pointed to)
  writeln(numPtr1^);
```

James Tam

Accessing Pointers (2)

Example:

```
type
  IntegerPointer = ^integer;
  :
  :
begin
  var numPtr1 : IntegerPointer;
  new(numPtr1);

  (Pointer)
  writeln(numPtr1);

  (Memory pointed to)
  writeln(numPtr1^);
```

James Tam

Accessing Memory Allocated Through A Pointer

Normally memory is accessed through a variable name:

```
var num : integer;
num := 12;
```

A pointer is a variable and the pointer can be accessed through the name of the pointer:

```
type
  IntegerPointer = ^integer;
begin
  var numPtr : IntegerPointer;
  new(numPtr);
```

However the memory allocated through the pointer (referred to by the pointer) can only be accessed through the pointer

James Tam

Accessing Memory Allocated Through A Pointer (2)

- In a similar fashion if you have the address of a location in memory then it may be accessed or modified without using a variable name.
- This is precisely how variable parameters (pass by reference) are implemented!

James Tam

Using Pointers : Allowable Operations

Assignment :=

Relational

• Equality =

• Inequality \neq

For the following slides assume that the following statements:

```
var numPtr1 : ^integer;  
var numPtr2 : ^integer;  
new(numPtr1);  
numPtr1^ := 1;  
new(numPtr2);  
numPtr1^ := 2;
```

James Tam

Using Pointers : Assignment

Format:

(Pointer)

pointer name := pointer name;

(Memory pointed to)

pointer name ^ := expression;

Example:

(Pointer)

numPtr1 := numPtr2;

(Memory pointed to)

numPtr1 ^ := 100;

James Tam

Using Pointers : Allowable Operations (Equality)

Format:

(Pointer)

if (*pointer name 1 = pointer name 2*) then

(Memory pointed to)

if (*pointer name 1 ^ = pointer name 2 ^*) then

Example:

(Pointer)

if (numPtr1 = numPtr2) then

(Memory pointed to)

if (numPtr1 ^ = numPtr2 ^) then

James Tam

Using Pointers : Allowable Operations (Inequality)

Format:

(Pointer)

if (*pointer name 1* <> *pointer name 2*) then

(Memory pointed to)

if (*pointer name 1*^ <> *pointer name 2*^) then

Example:

(Pointer)

if (numPtr1 <> numPtr2) then

(Memory pointed to)

if (numPtr1^ <> numPtr2^) then

James Tam

Pointers : First Example

```
program pointer1 (output);
type
  IntegerPointer = ^integer;
begin
  var num      : integer;
  var numPtr1 : IntegerPointer;
  var numPtr2 : IntegerPointer;
  writeln('Example 1');
  num := 10;
  new(numPtr1);
  new(numPtr2);
  numPtr1^ := 100;
  numPtr2^ := 100;
  writeln('num = ':11, num:3);
  writeln('numPtr1^ = ':11, numPtr1^:3);
  writeln('numPtr2^ = ':11, numPtr2^:3);
```

James Tam

Pointers : First Example (2)

```
if (numPtr1 = numPtr2) then
  writeln('Same memory')
else
  writeln('Separate memory');
if (numPtr1 ^= numPtr2^) then
  writeln('Same data')
else
  writeln('Different data');
(* Not allowed *)
(*writeln('numPtr1=', numPtr1); *)

writeln('Example 2');
num := numPtr1^;
writeln('num = ':11, num:3);
writeln('numPtr1^ = ':11, numPtr1^:3);
num := 33;
writeln('num = ':11, num:3);
writeln('numPtr1^ = ':11, numPtr1^:3);
writeln;
```

James Tam

Pointers: First Example (3)

```
writeln('Example 3');
numPtr2 ^ := 66;
numPtr1 := numPtr2;
if (numPtr1 = numPtr2) then
  writeln('Same memory')
else
  writeln('Separate memory');
numPtr2^ := 33;
writeln('numPtr1^ = ':11, numPtr1^);
writeln('numPtr2^ = ':11, numPtr2^);

dispose(numPtr1);
(* dispose(numPtr2); *)

(* Indicating that neither pointer points to any memory *)
numPtr1 := NIL;
numPtr2 := NIL;
end.
```

James Tam

RAM: How Things Are Divided Up

Memory that's allocated for programs that are running

Executable e.g., a.out (binary, machine language instructions)
Static memory: global constants and variables
Dynamic memory: 'The stack' for parameters and local variables
Dynamic memory: 'The heap' for memory allocated through a pointer

THE HEAP
`new (pointer);`

GLOBALS

```
program test;  
const  
  SIZE= 10;  
var  
  num : integer;  
begin  
end.
```

THE STACK

```
procedure proc (num1 :  
integer);  
var  
  num2 : integer;  
begin  
  var num3 : integer;  
end;
```

James Tam

Pointers And Parameter Passing

Value parameters

- In the call to the module what's passed in is a copy of the value stored in the parameter.
- The header for the module declares the name of the *local identifier/local variable used to store the value stored in the parameter.*

Variable parameters

- In the call to the module what's passed in is the address of the variable.
- The header for the module declares the name of the *local pointer that is used to store the address of the parameter.*
- The pointer is automatically de-referenced (to change the original parameter) whenever the local identifier is accessed.

James Tam

Pointers And Parameter Passing (2)

```
program parameters (output);  
  
procedure proc (   num1 : integer;  
                var num2 : integer);  
begin  
    num1 := 10;  
    num2 := 20;  
end;  
  
begin  
    var num1 : integer;  
    var num2 : integer;  
    num1 := 1;  
    num2 := 2;  
    proc(num1,num2);  
end.
```

James Tam

Parameter Passing: Rules Of Thumb You Should Know For Data Parameters

Value parameters

- Data: What's passed in *cannot* change (changes are made to a local copy).

Variable parameters

- Data: What's passed in *can* change (changes are made to the original parameter)

James Tam

Parameter Passing: Rules Of Thumb You Should Learn For Pointer Parameters

Value parameters (pointer parameter)

- Pointers: What's passed in (a pointer) *cannot* change (changes are made to a local copy of the pointer).

Variable parameters (pointer parameter)

- Pointers: What's passed in (a pointer) *can* change (changes are made to the original pointer parameter)

Value or variable parameters (what the pointer parameter points to)

- Value parameter:
 - A local copy of the pointer is made for the module which contains the address of a data variable.
 - *This allows the data referred to by the pointer to be changed.*
- Variable parameter:
 - The address of the pointer parameter is passed to another local pointer (pointer to a pointer).
 - *Again this allows the data referred to by the pointer to be changed.*

James Tam

Pointers As Value Parameters

Need to define a type for the pointer first!

Format (defining a type for the pointer):

```
type  
  <pointer name> = ^ <type pointed to>;
```

Format (passing pointer):

```
procedure procedure name (pointer name (1) : type of pointer (1);  
  pointer name (2) : type of pointer (2);  
  :  
  :  
  pointer name (n) : type of pointer (n));
```

```
function function name (pointer name (1) : type of pointer (1);  
  pointer name (2) : type of pointer (2);  
  :  
  :  
  pointer name (n) : type of pointer (n));
```

James Tam

Pointers As Value Parameters (2)

Example (defining a type for the pointer)

type

```
CharPointer = ^char;
```

Example (passing pointer):

```
procedure proc1 (aCharPointer : CharPointer );
```

```
begin
```

```
    :           :
```

```
end;
```

James Tam

Pointers As Variable Parameters

Need to define a type for the pointer first!

Format (defining a type for the pointer):

type

```
<pointer name> = ^ <type pointed to>;
```

Format (passing pointer):

```
procedure procedure name (var pointer name (1) : type of pointer (1);
```

```
    var pointer name (2) : type of pointer (2);
```

```
        :           :
```

```
    var pointer name (n) : type of pointer (n));
```

```
function function name (var pointer name (1) : type of pointer (1);
```

```
    var pointer name (2) : type of pointer (2);
```

```
        :           :
```

```
    var pointer name (n) : type of pointer (n));
```

James Tam

Pointers As Variable Parameters

Need to define a type for the pointer first!

Example (defining a type for the pointer)

```
type
  CharPointer = ^char;
```

Example (passing pointer):

```
procedure proc1 (var aCharPointer : CharPointer );
begin
    :      :
end;
```

James Tam

Pointers: Second Example

A full version of this program can be found in Unix under:
/home/231/examples/pointers/pointer2.p

```
program pointer2 (output);
type
  CharPointer = ^char;

procedure proc1 (charPtr : CharPointer);
var
  temp : CharPointer;
begin
  writeln;
  writeln('Proc1');
  new(temp);
  temp^ := 'A';
  charPtr := temp;
  writeln('temp^ = ', temp^);
  writeln('charPtr^ = ', charPtr^);
end;
```

James Tam

Pointers: Second Example (2)

```
procedure proc2 (var charPtr : CharPointer);
var
  temp : CharPointer;
begin
  writeln;
  writeln('Proc2');
  new(temp);
  temp^ := 'A';
  charPtr := temp;
  writeln('temp^ = ', temp^);
  writeln('charPtr^ = ', charPtr^);
end;
```

James Tam

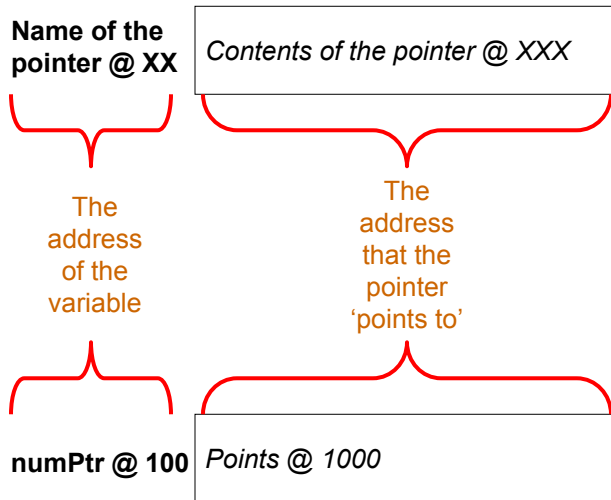
Pointers: Second Example (4)

```
begin          (* Main program *)
  var charPtr : CharPointer;
  new (charPtr);
  charPtr^ := 'a';
  writeln;
  writeln('Main program. ');
  writeln('charPtr^ = ', charPtr^);
  proc1(charPtr);
  writeln('After proc1');
  writeln('charPtr^ = ', charPtr^);
  proc2(charPtr);
  writeln('After proc2');
  writeln('charPtr^ = ', charPtr^);
  writeln;
end.          (* End of main program *)
```

James Tam

Summary: Passing Pointers As Parameters

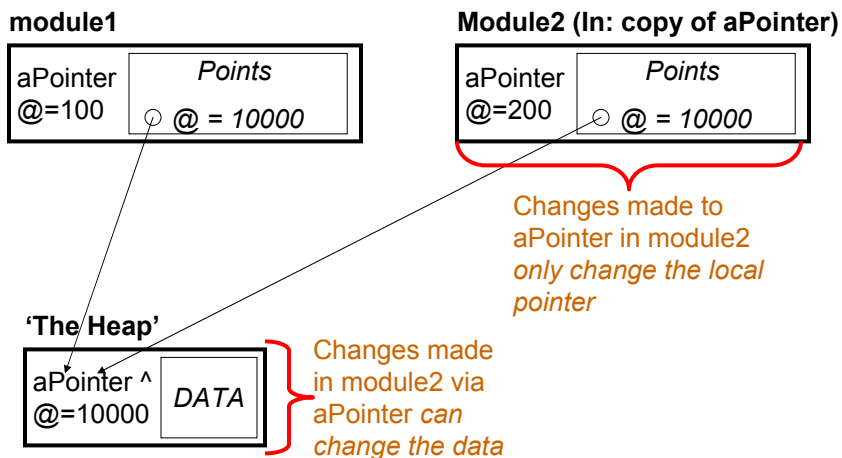
Reminder of the notation



James Tam

Summary: Passing Pointers As Value Parameters

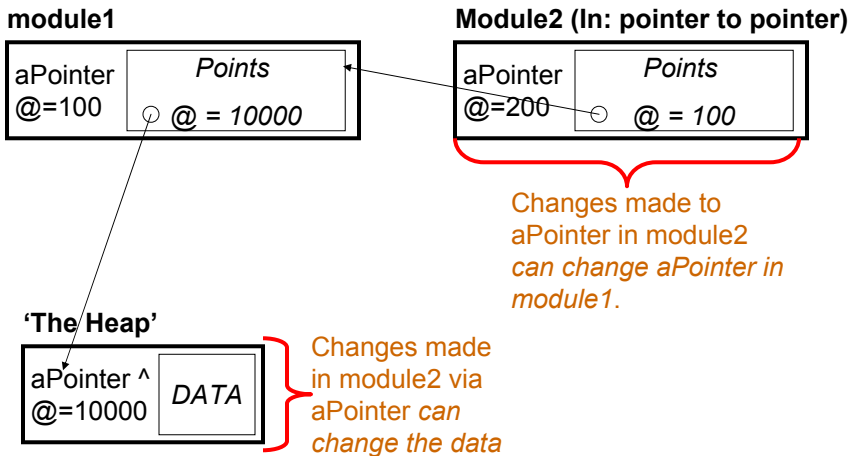
Module1 calls Module2 and passes 'aPointer' as a value parameter e.g., 'procl' in the previous example



James Tam

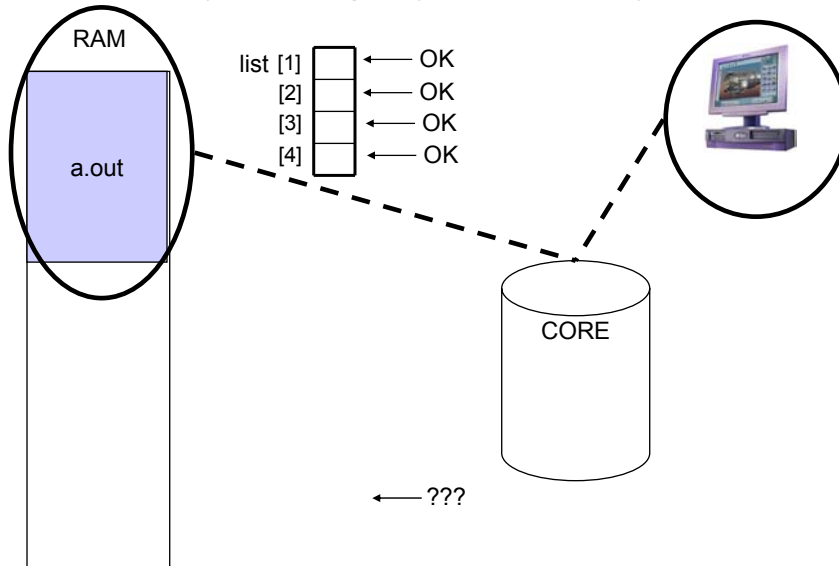
Summary: Passing Pointers As Variable Parameters

Module1 calls Module2 and passes 'aPointer' as a variable parameter e.g., 'proc2' in the previous example



James Tam

What You Know: How Segmentation Faults Are Caused By Indexing Beyond The Array Bounds



James Tam

What You Will Learn: How Segmentation Faults (Possibly Bus Errors) Can Be Caused By Incorrect Pointer Dereferencing

A full version of this program can be found in Unix under:
/home/231/examples/pointers/pointer3.p

program pointer3 (output);

```
type
  IntegerPointer = ^ integer;

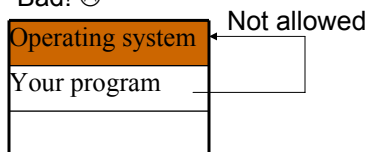
begin
  var numPtr1 : IntegerPointer;
  writeln('1');
  numPtr1^ := 100;
  writeln('2');
  numPtr1 := NIL;
  writeln('3');
  numPtr1^ := 100;
end.
```

James Tam

Segmentation Fault

- A 'memory access violation' (an attempt is made to access a part of memory that is 'forbidden' to a program).
- Can be caused by programs that index beyond the bounds of an array.
- Can also be caused by programs that improperly de-referenced pointers.

Bad! ☹

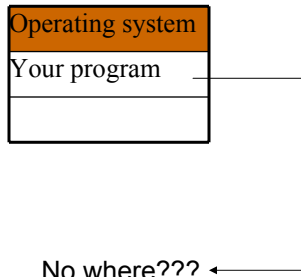


James Tam

Bus Error

- Caused by a ‘faulty memory access’.
- May occur when a program tries to access a non-existent memory address.
- Could also be triggered by accessing beyond the bounds of an array or improperly de-referenced pointers.

Bad!!! ☹



James Tam

You Should Now Know

- How to declare new types that are pointers to data
- How to declare variables that are pointers
- The difference between static and dynamically allocated memory
- How to dynamically allocate memory
- How to de-allocate memory
- Why and when to set pointers to NIL
- How to access a pointer and how to access what the pointer points to
- How to assign values to a pointer and how to assign values to what the pointer points to
- What operations can be performed on pointers and how does each one work
- How to pass pointers as value and variable parameters
- How incorrect pointer usage results in problems with memory accesses such as segmentation faults and bus errors

James Tam