

# Pointers, characters and grouping

## Lecture B06



Lecture notes section B06

2002-01-14

CSE1 303 Part B lecture notes

1

## Last time

- Byte order
- bitwise operations
  - AND, OR, NOT, XOR
  - masking
- shifting
  - relationship to multiplication and division
  - right shifts

2002-01-14

CSE1 303 Part B lecture notes

2

## In this lecture

- Pointers
  - value which is the address of some other value
- Aggregation of data
  - arrays
  - structs
- Characters
  - ASCII
- Strings

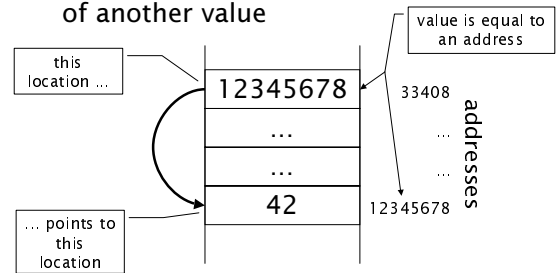
2002-01-14

CSE1 303 Part B lecture notes

3

## Pointers

- Pointer is a value that is an address of another value



2002-01-14

CSE1 303 Part B lecture notes

4

## Pointers

- Pointer is an address
  - addresses are unsigned integers
  - pointers are unsigned integers too
  - pointer takes up as many bytes as needed to represent any possible address
    - 32 bits (4 bytes) in MIPS (addresses range from 0 to 4294967295)
  - can print the address a pointer contains in C using `printf("%p", ptr)`

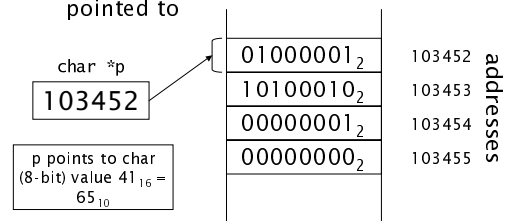
2002-01-14

CSE1 303 Part B lecture notes

5

## Pointers

- Pointers need a type
  - to be able to interpret thing being pointed to



2002-01-14

CSE1 303 Part B lecture notes

6

## Pointers

- Pointers need a type
  - to be able to interpret thing being pointed to

unsigned short \*p

103452

p points to unsigned short (16-bit) value  
A241<sub>16</sub> = 41537<sub>10</sub>  
(little endian)

01000001 <sub>2</sub>	103452	addresses
10100010 <sub>2</sub>	103453	
00000001 <sub>2</sub>	103454	
00000000 <sub>2</sub>	103455	

2002-01-14 CSE1303 Part B lecture notes 7

## Pointers

- Pointers need a type
  - to be able to interpret thing being pointed to

long \*p

103452

p points to long (32-bit) value  
0001A241<sub>16</sub> = 107073<sub>10</sub>  
(little endian)

01000001 <sub>2</sub>	103452	addresses
10100010 <sub>2</sub>	103453	
00000001 <sub>2</sub>	103454	
00000000 <sub>2</sub>	103455	

2002-01-14 CSE1303 Part B lecture notes 8

## Pointer arithmetic

- Can perform addition and subtraction on pointers in C

char \*p

103453

p (103453) points here

p + 1 (103454) points here

01000001 <sub>2</sub>	103452	addresses
10100010 <sub>2</sub>	103453	
00000001 <sub>2</sub>	103454	
00000000 <sub>2</sub>	103455	

p - 1 (103452) points here

p + 2 (103455) points here

this works even if p points to a type other than char (offset is scaled)

2002-01-14 CSE1303 Part B lecture notes 9

## Pointers and arrays

- Pointer arithmetic is related to arrays

char \*p

103452

\*p = \*(p+0) = p[0] = 0xA1

\*(p+1) = p[1] = 0xA2

\*(p+2) = p[2] = 0x01

\*(p+n) = p[n]

01000001 <sub>2</sub>	103452	addresses
10100010 <sub>2</sub>	103453	
00000001 <sub>2</sub>	103454	
00000000 <sub>2</sub>	103455	

2002-01-14 CSE1303 Part B lecture notes 10

## Arrays

- A group of values
  - each has the same type and size
- Elements accessed by index
  - index is integer
  - lowest index is 0
  - highest index is size - 1
- Elements stored consecutively in memory
  - lowest index at lowest address
  - highest index at highest address
  - addresses of adjacent elements differ by size of single element

2002-01-14 CSE1303 Part B lecture notes 11

## Arrays

- long a[5] = {7, 256, -1, 0, 107073};

To avoid clutter, memory often drawn with 1 word per box (here 4 bytes, 401960 to 401963). Actual byte values depend on byte order.

a[0] =	0x00000007	401960	addresses
a[1] =	0x00000100	401964	
a[2] =	0xFFFFFFFF	401968	
a[3] =	0x00000000	401972	
a[4] =	0x0001A241	401976	

2002-01-14 CSE1303 Part B lecture notes 12

## Structures

- A group of values
  - each member can be different type
- Members accessed by name
  - names and types assigned during declaration of structure data type
  - compiler uses declaration to determine for each member
    - offset from start of structure
    - size

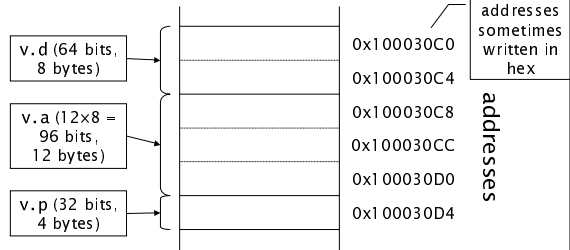
2002-01-14

CSE1 303 Part B lecture notes

13

## Structures

- `struct { double d; char a[12]; char *p; } var;`



2002-01-14

CSE1 303 Part B lecture notes

14

## Characters

- Single printable symbol
  - digits
  - letters
  - LETTERS
  - punctuation
  - space
- Other characters
  - line feed ("newline" in Unix)
  - tab
  - end-of-string (null) character '\0'
  - control characters

2002-01-14

CSE1 303 Part B lecture notes

15

## ASCII

- Assign every unique character a distinct number from 0 to 127

character	number	character	number
end-of-string	0	A	65
space	32	a	97
*	42	{	123
0	48	delete	127

the complete version of this table is the American Standard Code for Information Interchange (ASCII)

2002-01-14

CSE1 303 Part B lecture notes

16

## Strings

- Sequence of characters
  - start indicated by pointer to first character
  - end indicated by end-of-string (null) character after final character
  - usually stored in an array of char

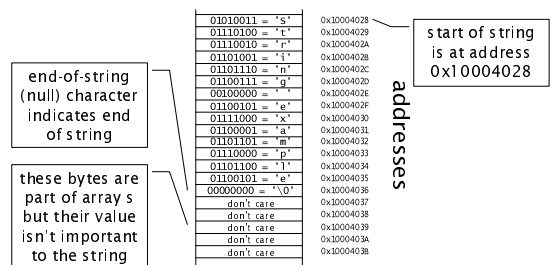
2002-01-14

CSE1 303 Part B lecture notes

17

## Strings

- `char s[20] = "String example";`



2002-01-14

CSE1 303 Part B lecture notes

18

## Covered in this lecture

- Pointers
  - contain address of some other value
- Aggregation of data
  - arrays
  - structs
- Characters
  - ASCII
- Strings

2002-01-14

CSE1303 Part B lecture notes

19

## Going further

- Union data types
  - similar to structures
  - dynamic typing in C
- Padding and alignment
  - wasting a little space to improve memory access speed

2002-01-14

CSE1303 Part B lecture notes

20

## Next time

- Compilers
  - code generation
- Assembly language



Reading:  
Lecture notes section B07

2002-01-14

CSE1303 Part B lecture notes

21

## Copyright

Copyright © 2001 Deborah Pickett.  
No part of this presentation may be  
duplicated without permission from  
the author.

2002-01-14

CSE1303 Part B lecture notes

22