Intro to C, part 2 - Pointers	

Context

The **C programming language** was created by Dennis Ritchie at Bell Labs in the 1970s.
C is notable for several reasons, described in Part I of this activity.
This activity assumes that you are familiar with Java or C#, which are similar to C,
and explores some key features of C that differ from these languages.

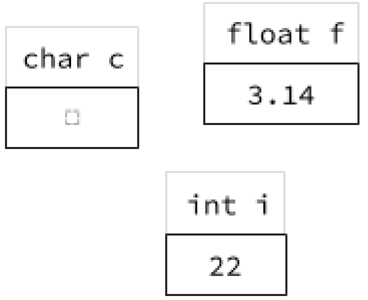
© 2018-2018 by Clif Kussmaul, clif@kussmaul.org. This work is licensed under
a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.
Revised for the purpose of CSCI-201, Spring 2019 by Joanna Klukowska, joannakl@cs.nyu.edu.

A. Memory Addresses	

```
// Each □ stands
// for a missing
// symbol or value

char  c = 'A';
float f = 3.14;
int   i = □;

```



Address	Value
0400	22
...	
0408	'A'
...	
0802	3.14

1. When a program runs, the computer finds a place in memory to store each variable.
When we trace code (left), we sometimes use **diagrams** of variables and values (middle), but each variable is really a memory location with a **numeric address** (right).
(We will use 4-digit addresses, not 10 or more digits used by a modern computer.)

a.	How many variables are declared in the code above?	
b.	What value is stored in variable `f`?	
c.	What value is stored in variable `i`?	
d.	What value is stored at address `0408`?	
e.	What address is used for variable `f`?	
f.	What variable is stored at address `0400`?	
g.	It is easier to write programs that use numeric addresses or variable names?	

2. For **primitive types** (e.g. `char`, `int`, `float`, `...`),
each value uses the same amount of memory For **arrays**, the amount of memory depends on the array size.
The table at right show the typical size of primitive types.
Use them to decide how many bytes are needed for:

type	# bytes
char	1
short	2
int	4
float	4
double	8

a.	an `int` value?	
b.	an array of 10 `float` values?	
c.	an array of 20 characters	

© 2018-2018 by Clif Kussmaul, clif@kussmaul.org. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.
Revised for the purpose of CSCI-201, Spring 2019 by Joanna Klukowska, joannakl@cs.nyu.edu.

B. Pointers

```
char  c;    // declares c to be a character
char *p;    // declares p to be an address of a character
 p = &c;    // gets address of  c, puts it in p
 c = *p;    // gets value from address in p, puts it in c
*p =  c;    // gets value from c, puts it at address in p
```

1. High-level languages often try to hide addresses from programmers.
However, C has special syntax and operators (shown above) to use memory addresses.

a.	In a variable declaration, what character indicates that a variable will contain an address?	
b.	What character gets the address of a variable? This is the **address of operator**.	
c.	What character gets the variable at an address? This is the **indirection operator** or **dereference operator**.	
d.	What character is used in 2 different ways for addresses?	

2. Use the address and indirection operators to show the C code to:

a.	Define variables `f` and `g` for floating point numbers.	
b.	Define variables `q` and `r` for the addresses of floating point numbers.	
c.	Set `q` to be the address of `f`, and `r` to be the address of `g`,	
d.	Set `f` to be half of its former value.	
e.	Set `g` to be half of the value that `q` points to.	
f.	Set the variable that `r` points to, to be half of `f`.	
g.	Set the variable that `q` points to, to be twice the variable that `r` points to.	

3. Why might a variable that contains an address be called a **pointer**?

*© 2018-2018 by Clif Kussmaul, clif@kussmaul.org. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.
Revised for the purpose of CSCI-201, Spring 2019 by Joanna Klukowska, joannakl@cs.nyu.edu.*

4. `*b` is used in both lines at right, but with different meanings. Explain the difference, and how we know which is which.

```
int a  = 2, *b  = &a;
a += 2;
*b +=  2;
```

5. In the code above, what is the final value of a? (Hint: the answer is not 4)

```
01 // Each □ stands for
02 // a missing symbol or value
03
04 int  i =  □,
05      j =  1024;
06 int *p = &□,
07     *q = &i;
08
```

Address	Value	Var
...		
2004	□	j
2008	256	i
2012	2004	p
2016	□	q
...		

6. The value of a pointer is an address of something else in memory.

a.	How many variables are declared in the code above?	
b.	What value is stored at address 2012?	
c.	What is the address of `i`?	
d.	What is the value of `i`?	
e.	What is the address of `p`?	
f.	What address is stored in `p`?	
g.	What variable uses address 2004?	
h.	What variable does `p` point to?	
i.	What should be the value of `q`?	
j.	In line 04 above, what should go in the blank?	
k.	In line 06 above, what should go in the blank?	

© 2018-2018 by Clif Kussmaul, clif@kussmaul.org. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.
Revised for the purpose of CSCI-201, Spring 2019 by Joanna Klukowska, joannakl@cs.nyu.edu.

7. When a program has 2 or more names for the same memory location, each name is called an **alias**. A change using one name also affects the others; this is called **aliasing**. In the code for the previous question, what **alias** is used for `i`? for `j`? Explain why aliasing can lead to potential problems in software.

8. Expressions that combine types (including pointer types) can lead to problems, and the C compiler might not detect or report them. Be careful! Describe the possible type problem(s) in each example below.

	Code	**Possible Problem**
a.	`int i=3.14;`	
b.	`int j=3; int *p=j;`	
c.	`int i=4; float *p=&i;`	

d.	`int j=4, k=5;`
	`int *q=&j; *q=&k;`	

9. The table below shows lines of C code with pointers, with a column for each variable. **Trace** the code to complete the table and show how variables change. For **pointers p, q, r**, show the address operator and the variable that is pointed to, not a numeric address.

	C code **variables:**	**a**	**b**	**c**	**p**	**q**	**r**
a.	`int a = 1;     int *p = &a;`	1	---	---	&a	---	---
b.	`int b = 2;     int *q = &b;`						
c.	`int c = a+b;   int *r =  p;`						
d.	`    a = *q + 5;     p = &c;`						
e.	`   *q = *r;         r =  q;`						

10. In the trace above, `p` and `q` are both pointers. In complete sentences, explain the difference between `p = q` and `*p = *q`.

© 2018-2018 by Clif Kussmaul, clif@kussmaul.org. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License. Revised for the purpose of CSCI-201, Spring 2019 by Joanna Klukowska, joannakl@cs.nyu.edu.

C. Arrays	

Java and C#	C

`char[] c = new char[10];`
`char[] d = { 'c','a','t' };`
`d[0] = d[2];`
	`char  c[10];`
	`char  d[] = { 'c','a','t' };`
	`d[0] = d[2];`

1. C has arrays, and the syntax is slightly different from Java and C#.
Use the examples above to answer the questions below about C arrays.

a.	What characters mark a set of values to be stored in an array?	
b.	What characters declare an array variable?	
c.	What characters mark an index in an array?	
d.	Declare array `fdata` of 10 floating point values.	
e.	Declare array `idata` with the integers 0 to 4.	

© 2018-2018 by Clif Kussmaul, clif@kussmaul.org. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License. Revised for the purpose of CSCI-201, Spring 2019 by Joanna Klukowska, joannakl@cs.nyu.edu.

```
01 // Each □ stands for
02 // a missing symbol or value
03
04 int a[] = {□,□,□};
05 int *p  = &a[□];
06
```

Address	Value	Var
...		
4012	4212	a
4016	4220	p
...		
4208	128	
4212	256	
4216	512	
4220	1024	
4224	2048	

2. In C (and most other languages), an **array** is a set of adjacent locations in memory.
In C, an array variable contains the address of the first (0th) value in the array.

a.	How many variables are declared in the code above?	
b.	What value is stored at address 4220?	
c.	What is the address of `a`?	
d.	What address is in `a`?	
e.	What is the address of `a[0]`?	
f.	What is the address of `a[1]`?	
g.	What value is in `a[1]`?	
h.	What value is in `a[-1]`?	
i.	Which position in `a` does `p` point to?	
j.	What is the **difference** in the addresses between one array element and the next?	
k.	If each address refers to 1 byte (8 bits), how many bits does each array element use?	
l.	In the above code, what should replace the three blanks on line 04?	
m.	In the above code, what should replace the blank on line 05?	

*© 2018-2018 by Clif Kussmaul, clif@kussmaul.org. This work is licensed under
a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.
Revised for the purpose of CSCI-201, Spring 2019 by Joanna Klukowska, joannakl@cs.nyu.edu.*

```
01 char  d[] = { 'c','a','t' };        // create array
02 char *e = d,  *f = &d[2];           // set pointers to array positions
03 e[0] = 'b'; f[0] = 'g';             // use pointers to change array
04 e[1] = 'o'; f[-1] = 'i';
```

3. In C, we can use pointers and arrays to do things that might be difficult in other languages.

a.	What position in `d` does `e` point to?	
b.	What position in `d` does `f` point to?	
c.	What will `d` contain after line 3 above?	
d.	What will `d` contain after line 4 above?	

4. The table below shows lines of C code with pointers, with a column for each variable. Trace the code to complete the table and show how variables change. For **pointers p, q**, show the address operator and the variable that is pointed to, not a numeric address.

	C statement	a[0]	a[1]	a[2]	c	p	q
a.	`char  a[] = {'a','b','c'};`	'a'	'b'	'c'			
b.	`char  c   = 'x';`						
c.	`a[1]      = c;`						
d.	`char *p   = a;`						
e.	`char *q   = &a[1];`						
f.	`p[2]      = c+1;`						
g.	`q[-1]     = c-1;`						

5. Refer to the C statements in the previous question, and list all **aliases** for:

a.	`a`	
b.	`c`	
c.	`a[2]`	

© 2018-2018 by Clif Kussmaul, clif@kussmaul.org. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.
Revised for the purpose of CSCI-201, Spring 2019 by Joanna Klukowska, joannakl@cs.nyu.edu.

D. Strings

```
char  text1[] = { 'H','e','l','l','o','\0' };
char  text2[] = "Goodbye";
char *text3   = text2[4];
```

1. Most programs use text for input, output, user instructions, error messages, etc. Thus, most programming languages have special syntax for character **strings**. In C, a string is a character array that ends with the **null character** (`'\0'`). Thus, we say that C uses **null-terminated strings**.

a.	Explain why C must use 8 characters (not 7) for the string “Goodbye”.	
b.	Explain why a function to get the length of a string is O(N), not O(1). (N is the length of the number of characters in the string.)	

Java and C# each have a `String` class with useful methods for strings.
The C string library (`#include <string.h>`) defines a variety of useful functions.
A few are listed above, but there are many others.

`char c; int n;`
`char *s,*t,*u;`	

`n = strlen(s);`	return the **length** (# of chars) of `s` (without null char)
`n = strcmp(s,t);`	**compare** strings `s` and `t` (alphabetically), return 0 if they are the same, <0 if (`s<t`), >0 if (`s>t`)
`u = strchr(s,c);`	**search** `s` and return pointer to first instance of `c`
`u = strstr(s,t);`	**search** `s` and return pointer to first instance of `t`
`u = strcpy(s,t);`	**copy** `t` into start of `s`, return `s`
`u = strcat(s,t);`	**concatenate** `t` to end of `s`, return `s`
`u = strncpy(s,t,n);`	**copy** up to `n` chars from `t` into start of `s`, return `s`
`u = strncat(s,t,n);`	**concatenate** up to `n` chars from `t` to end of `s`, return `s`

© 2018-2018 by Clif Kussmaul, clif@kussmaul.org. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.
Revised for the purpose of CSCI-201, Spring 2019 by Joanna Klukowska, joannakl@cs.nyu.edu.

E. Pointer Arrays

3. An array of non-primitive values (e.g. character strings) is really an array of pointers to its values.
The table at right shows the memory used by:

```
        char *p[] = { □, □ };
```

Use it to answer the questions below:

a.	What is the address of `p`?	
b.	What address is in `p`?	
c.	What is the address of `p[0]`?	
d.	What address is in `p[0]`?	
e.	What address is in `p[1]`?	
f.	What character does `p[0]` point to?	
g.	What character string does `p[0]` point to?	
h.	What character does `p[1][1]` point to?	

Address	Value	Var
...		
8024	8120	p
...		
8064	'd'	
8066	'o'	
8068	'g'	
8070	'\0'	
...		
8116	8204	
8120	8202	
8124	8064	
8128	8200	
...		
8200	's'	
8202	'c'	
8204	'a'	
8206	't'	
8208	'\0'	
...		

© 2018-2018 by Clif Kussmaul, clif@kussmaul.org. This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.
Revised for the purpose of CSCI-201, Spring 2019 by Joanna Klukowska, joannakl@cs.nyu.edu.