

COMPUTER SYSTEMS AND ORGANIZATION

C Strings and More

Daniel G. Graham Ph.D



UNIVERSITY
of VIRGINIA

ENGINEERING



1. Question from last time
2. Char Array in C
3. Demo of debugging
4. String in C
5. Const keyword
6. Two-dimensional arrays

ARRAY NOT QUITE POINTS

```
int x[4] = {1,2,3,4};
```

```
int y[5] = {1,2,3,4,5};
```

```
x = y // Not allowed.
```

```
//If you want to do this you  
will need to a memcpy  
(memcpy(x,y, sizeof(x)));
```

Arrays are of type
int [n] and language doesn't
allow these types to be
assigned

ARRAY TYPES NOT ASSIGNABLE

GNU nano 6.3 array.c	
<pre>#include <stdio.h> #include <stdlib.h> int main(){ int x[4] = {1,2,3,4}; int y[7] = {1,2,3,4,5,6,7}; x = y; }</pre>	<pre>array.c:7:4: error: array type 'int[4]' is not assignable x = y; ~ ^ 1 error generated. dgg6b@portal07:~/Examples\$</pre>

ARRAYS NOT QUITE POINTERS

Allowed the language

```
int x[4] = {1,2,3,4};  
int *p;  
p = x; //Same as p=&(x[0])
```

Allowed
pointer = array

Not allow by the language

```
int x[4] = {1,2,3,4};  
int *p;  
x = p //Not allowed ☹️
```

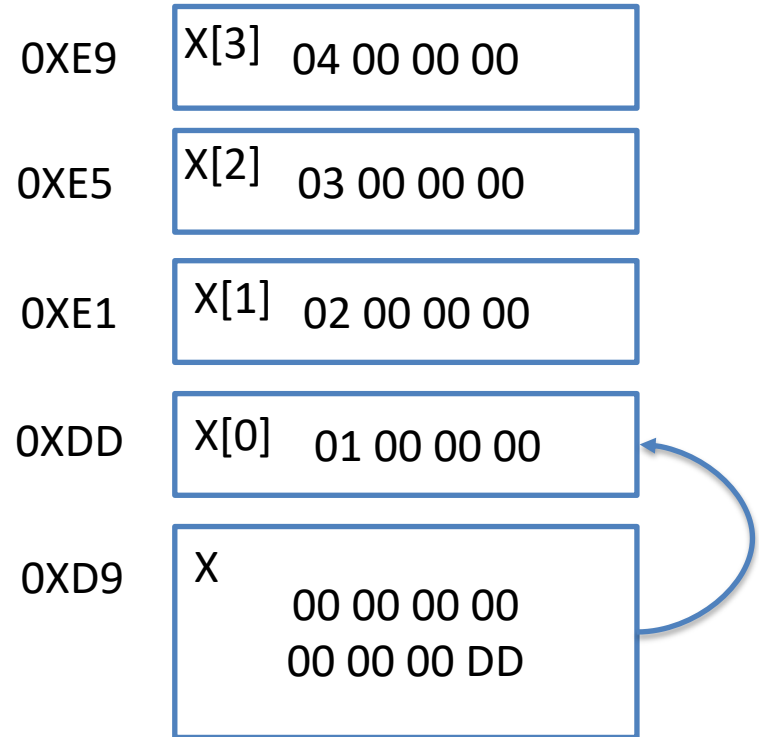
Because array types
int[4] is not assignable

LET'S LOOK AT SOME TRICKY EXAMPLES

TALK TO YOUR NEIGHBOR

```
int x[4] = {1,2,3,4};  
  
*(x + 1) = *x + *(x + 1);  
  
printf("value: %d", x[1]);
```

What does this print out?



SYNTACTIC SUGAR

$x[i] \longrightarrow *(x+i)$

These are equivalent

TALK TO YOUR NEIGHBOR

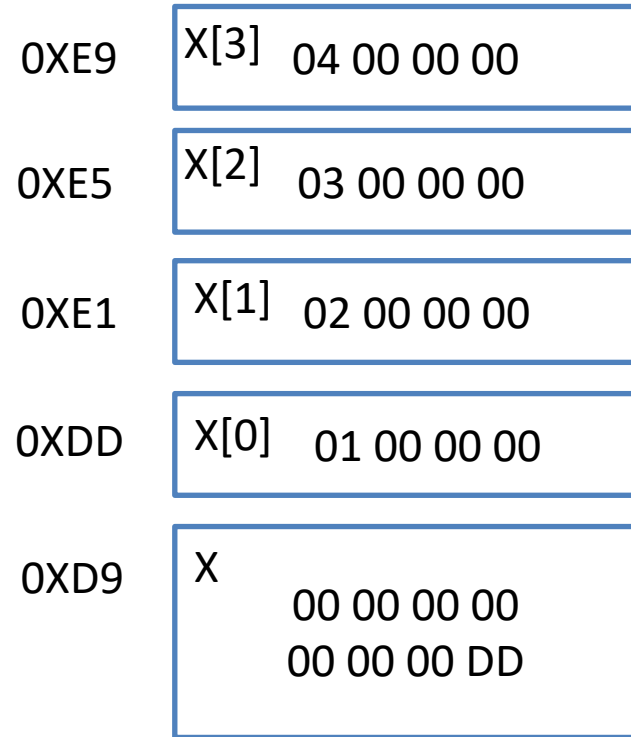
```
int x[4] = {1,2,3,4};
```

```
x = x + 1;
```

```
printf("value: %d", x[1]);
```

What does this print out?

We get an error. Array types cant be assigned.



LET'S SWITCH TO TERMINAL
A LOOK AT SOME CODE
ALSO PRACTICE USING THE DEBUGGER

ARRAY IN C

8 bits (1 byte) wide

```
char a[4] = {'A', 'B', 'C', 'D'}
```

RSP-0x3

0x44

RSP-0x2

0x43

RSP-0x1

0x42

RSP

0x41

CHAR ARRAY, AND STRING

```
char b[7] = {'D', 'a', 'n', 'i', 'e', 'l', '\0'}
```



Null terminal character

Terminate strings

CHAR ARRAY, AND STRING

```
char b[7] = {'D', 'a', 'n', 'i', 'e', 'l', '\0'};
```

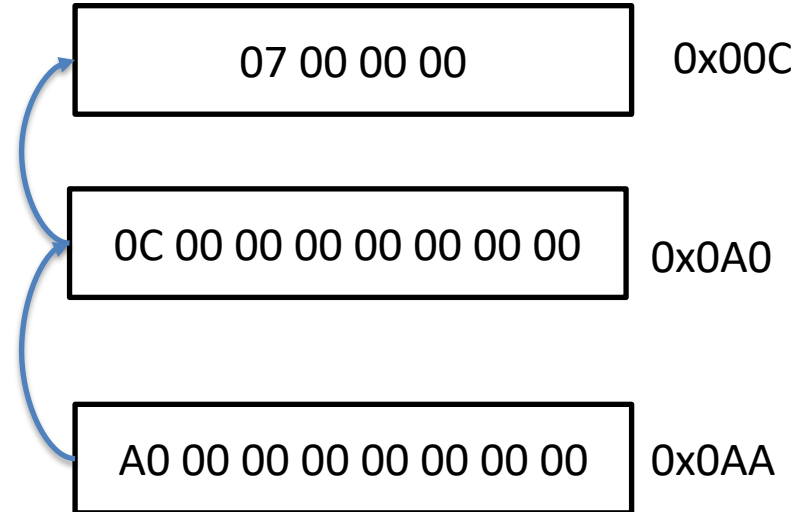
```
char *b = "Daniel";
```

POINTER TO A POINTER

```
int **x;
```

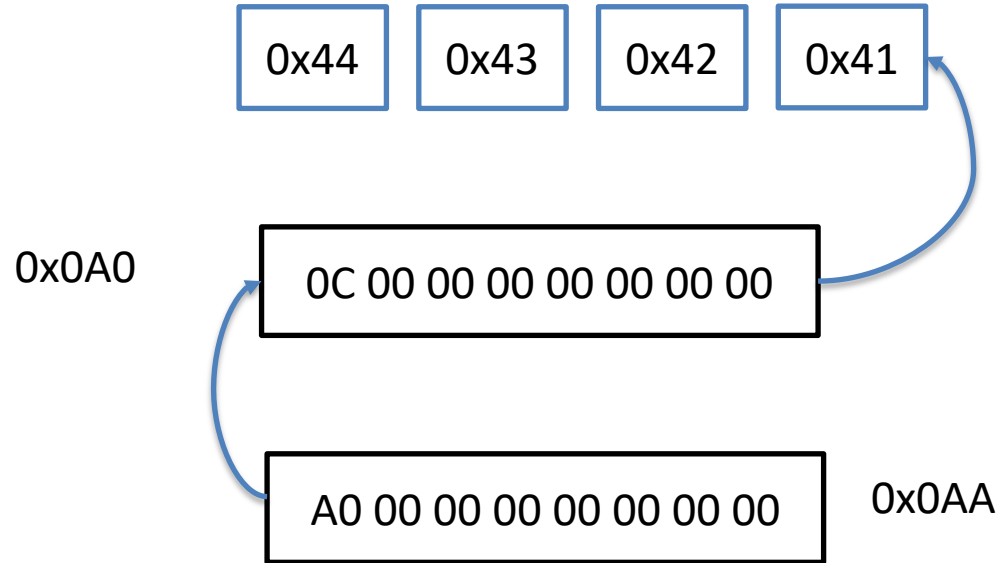
POINTER TO A POINTER

```
int variable = 7;  
int *pointer = &variable  
int **pointer2pointer = &pointer;
```



POINTER TO POINTER

```
char *abc = "ABCD";  
char **myPharse = &abc
```



LET'S IMPLEMENT STRING TOUPPER

Let's write a function that takes in a string and converts it uppercase

```
#include <stdio.h>

int main(){
    char *input = "lowercase";
    _toUpper(&input);
    printf("%s", input);
}
```

ASCII TABLE SNIPPET

Dec	Char	Dec	Char
64	@	96	`
65	A	97	a
66	B	98	b
67	C	99	c
68	D	100	d
69	E	101	e
70	F	102	f
71	G	103	g
--	--	---	---

We could just **subtract** 32 to our chars. (Need to add cases to ignore space and special characters like @)

```
#include <stdio.h>
```

```
void _toUpper(char *input){  
    int index = 0;  
    while(*(input+index) != '\0'){  
        //sub 32  
        *(input + index) = *(input + index) - 32;  
        index++;  
    }  
}
```

```
int main(){  
    char input[] = "lowercase";  
    _toUpper(input);  
    printf("%s \n", input);  
  
    return 0; //Optional  
}
```

STRING LITERALS IN C

```
#include <stdio.h>

void _toUpper(char *input){
    int index = 0;
    while(*(input+index) != '\0'){
        //add 32
        *(input + index) = *(input + index) - 32;
        index++;
    }
}

int main(){
    char *input = "lowercase";
    _toUpper(input);
    printf("%s \n", input);

    return 0; //Optional
}
```

```
Home directory usage for /u/dgg6b: 1%
You have used 1.29G of your 100G quota
```

```
dgg6b@portal07:~$ c
```

STRING LITERALS IN C

```
char *b = "Daniel";
```

```
char b[] = "Daniel";
```

These are not the
same thing in c

Let's look at the assembly to see what
happening

CHAR *B = "DANIEL" STORED AS A STRING IN CODE

GNU nano 6.3 arrayVSpoiner.c

```
#include <stdio.h>

int main(){
    char *pointer = "SomethingFun";
    printf("%s\n", pointer);
}
```

Stored as a string in
code



GNU nano 6.3 arrayVSpoiner.s

```
.text
.file "arrayVSpoiner.c"
.globl main                                # -- Begin
.p2align 4, 0x90
.type main,@function                       # @main

main:
.cfi_startproc
# %bb.0:
pushq %rax
.cfi_def_cfa_offset 16
movl $.L.str, %edi
callq puts@PLT
xorl %eax, %eax
popq %rcx
.cfi_def_cfa_offset 8
retq

.Lfunc_end0:
.size main, .Lfunc_end0-main
.cfi_endproc                                # -- End function

.type .L.str,@object                       # @.str
.section .rodata.str1.1,"aMS",@progbits,1

.L.str:
.asciz "SomethingFun"
.size .L.str, 13

.ident "clang version 14.0.6 (https://github.co>
```

[Wrote 6 lines]

^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute
^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify

[Read 29 lines]

^G Help ^O Write Out ^W Where Is ^K Cut
^X Exit ^R Read File ^\ Replace ^U Paste

GNU nano 6.3 arrayVSpoiner.c

```
#include <stdio.h>

int main(){
    char pointer[] = "SomethingFun";
    printf("%s\n", pointer);
}
```

Also stored code
section but a copy
Moved onto the stack
(so we can modify it)

GNU nano 6.3 arrayVSpoiner.s

```
# -- Begin function
.globl main
.p2align 4, 0x90
.type main,@function
main:
    .cfi_startproc
# %bb.0:
    subq $24, %rsp
    .cfi_def_cfa_offset 32
    movabsq $31091192681621864, %rax # imm = 31091192681621864
    movq %rax, 13(%rsp)
    movabsq $7956005065853857619, %rax # imm = 7956005065853857619
    movq %rax, 8(%rsp)
    leaq 8(%rsp), %rdi
    callq puts@PLT
    xorl %eax, %eax
    addq $24, %rsp
    .cfi_def_cfa_offset 8
    retq
.Lfunc_end0:
    .size main, .Lfunc_end0-main
    .cfi_endproc
# -- End function

.type .L__const.main.pointer,@object # __const.main.pointer
.section .rodata.str1.1,"aMS",@progbits,1
.L__const.main.pointer:
    .asciz "SomethingFun"
    .size .L__const.main.pointer, 13
```

[Wrote 6 lines]

^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute
^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify

[2] 0:nano*

^G Help ^O Write Out ^W Where Is ^K Cut
^X Exit ^R Read File ^\ Replace ^U Paste

"portal07" 02:23 30-Oct-23

THE DIFFERENCES

```
char *p = "Daniel";
```

p is a pointer

p and &p are NOT the same

```
char a[] = "Daniel";
```

a is an array

a and &a ARE the same

COMMAND LINE ARGUMENTS

This is a command-line argument



```
./a.out Hello
```

COMMAND LINE ARGUMENTS

This is a command-line argument



```
.clang hello.c
```

READING COMMAND LINE ARGUMENTS IN C

```
#include <stdio.h>
```

```
int main(int argc, char **argv){  
    if(argc > 1){  
        printf("argument was %s", *argv);  
    }  
}
```

./a.out Hello
prints a.out (Not Hello)

Get the first element in the array
just like in python argument is
name of the program itself

READING COMMAND LINE ARGUMENTS IN C

```
#include <stdio.h>
```

```
int main(int argc, char **argv){  
    if(argc > 0){  
        printf("argument was %s", *(argv + 1));  
    }  
}  
  
    prints hello
```

READING COMMAND LINE ARGUMENTS IN C

```
#include <stdio.h>

int main(int argc, char **argv){
    if(argc > 0){
        printf("argument was %s", argv[1]);
    }
}
```

CONST KEY WORD

Const keyword define a read only section of memory.

```
const int x = 10;
```

NOT REALLY THE SAME AS #DEFINE

`const int x = 10;`

Type information

`#define x 10`

No type information

STRING HELPER FUNCTIONS <STRING.H>

```
GNU nano 6.3 string.c
```

```
#include <stdio.h>  
#include <string.h>
```

```
int main(){  
    char *s = strdup("can all aardvarks quaff?");  
    printf("%s", s);  
}
```

```
Lldb | Target(F2) | Process(F3) | Thread(F4)  
lqq<Sources>qqqqqqqqqqqqqqqqqqqqqqqqqqqqqlqq<Thread  
x string.out`main xs> `aproces  
x   1 x #include <stdio.h> xx mq`qthr  
x   2 x #include <string.h> xx tqfraz  
x   3 x                        xx tqfraz  
x   4 x                        xx tqfraz  
x   5 x int main(){          xx mqfraz  
x   6 x     char *s = strdup("can all aarxx  
x   7 x         <<< Thread 1: breakpoint 2.1xx  
x   8 x                                     xx  
x   9 x }                     xx  
x  10 x                       xx  
  
xxxxxxxxxxxxxxx  
mqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqjxx  
lqq<Variables>qqqqqqqqqqqqqqqqqqqqqqqqqqx  
x `q(char *) s = 0x00000000004052a0 "xx  
x                                       xx  
x                                       xx  
x                                       xx  
x                                       xx  
mqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqmqqqqqqqq  
Process: 1457391 stopped Thread:  
[0] 0:lldb+ portal04 11:15 30-Oct-20
```


STRING HELPER FUNCTIONS

```
size_t strlen(const char *str)
```



- size_t - integer the size of a pointer (unsigned)
- ssize_t - integer the size of a pointer (signed)

STRING HELPER FUNCTIONS

const keyword prevents the value
The pointer points to from being reassigned



```
size_t strlen(const char *str)
```



- size_t - integer the size of a pointer (unsigned)
- ssize_t - integer the size of a pointer (signed)

8. [12 points] Consider the following C code:

```
char first[5] = {'f', 'y', 'i', '!', '\0'};  
char *second = strdup("hello");  
char *both[2] = {first, second};
```

What is printed for each of the following lines? If the program would crash or seg fault, write **crash**. *Hint: printf("%c", x); means "print the char stored in variable x."*

A. `printf("%c", (*both)[1]);`

B. `printf("%c", *(both[1]));`

C. `puts(&both[0][2]);`

y, h, i!

