COMPUTER SYSTEMS AND ORGANIZATION

C Strings and More

Daniel G. Graham Ph.D

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

ARRAYS NOT QUITE POINTERS

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

int y[4] = {5,6,7,8};

x = y // Not allowed.

//If you want to do this you
will need to a memcpy
(memcp(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
#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;

}
```

```
array.c:7:4: error: array type 'int[4]'
 is not assignable
        x = y;
        ~ ^
1 error generated.
dgg6b@portal07:~/Examples$
```

ARRAYS NOT QUITE POINTERS

Allowed by the language

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

Allowed
pointer = array

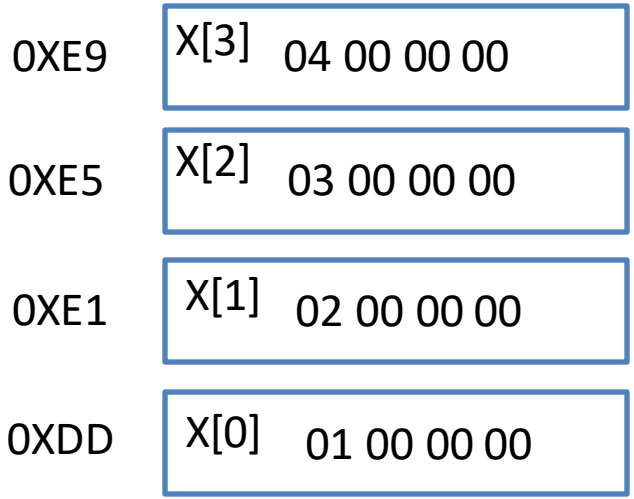
Not allowed by the language

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

Because array types
int[4] is not assignable

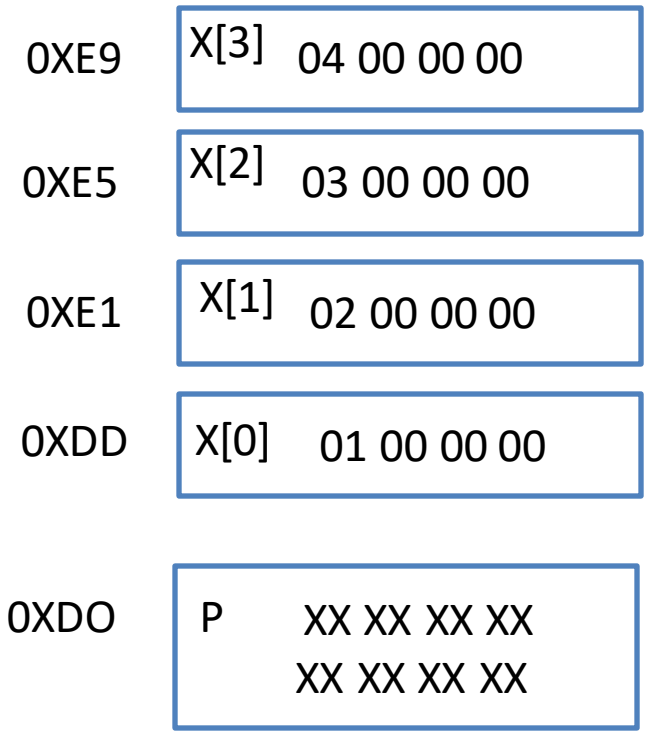
ARRAY VS POINTERS

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



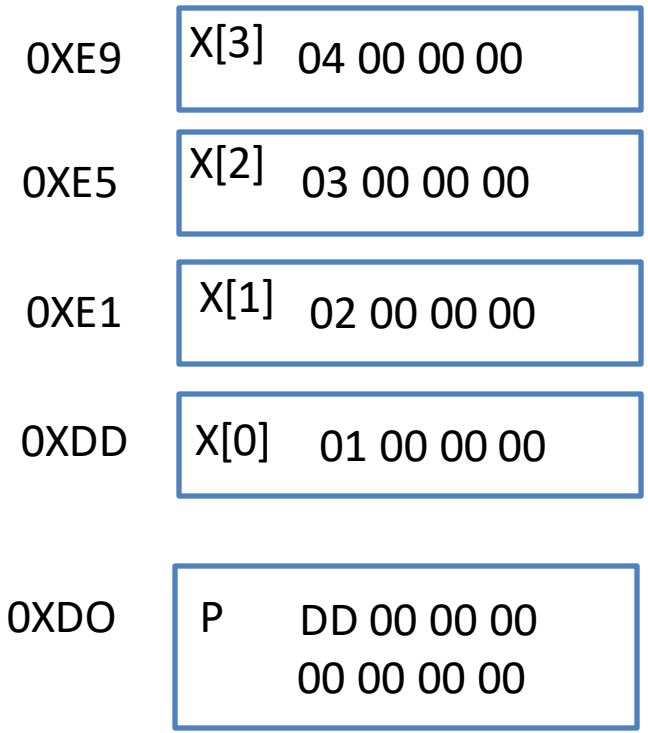
ARRAY VS POINTERS

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

int *p;
p = x; //Same as p=&(x[0])
```



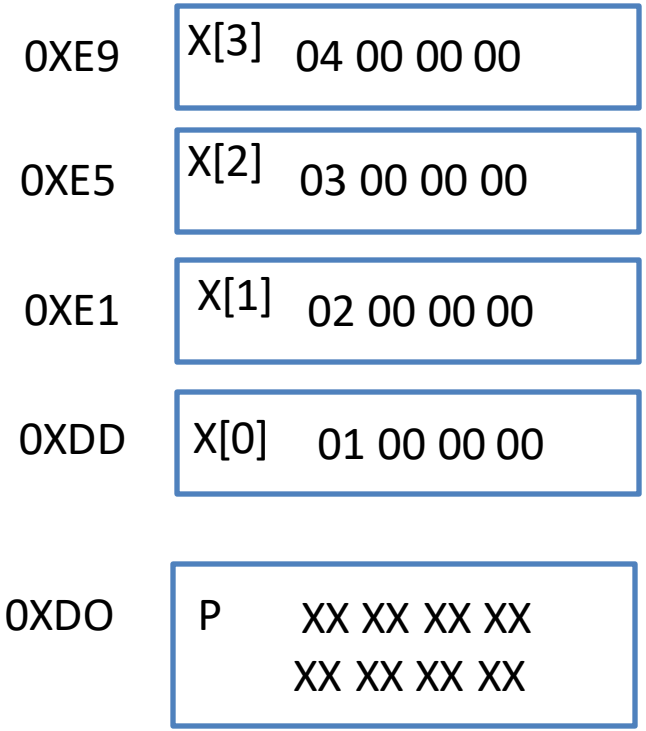
ARRAY VS POINTERS

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

int *p;
p = x; //Same as p=&(x[0])
```

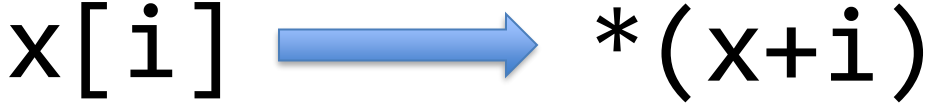


ARRAY VS POINTERS

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



SYNTACTIC SUGAR

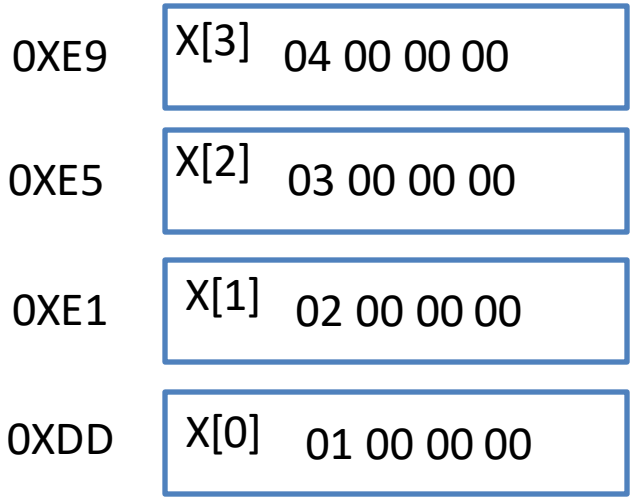


These are equivalent

TALK TO YOUR NEIGHBOR

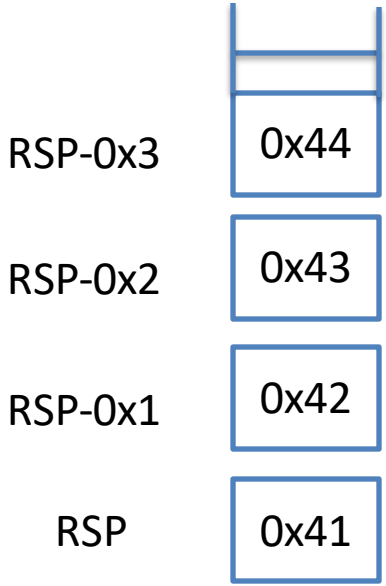
```
int x[4] = {1,2,3,4};
X[2] = *(x + 1);

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

What does this print out?



ARRAY IN C

```
char a[4] = {‘A’, ‘B’, ‘C’, ‘D’};
```



CHAR ARRAY, AND STRING

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

Null-terminating character

CHAR ARRAY, AND STRING

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

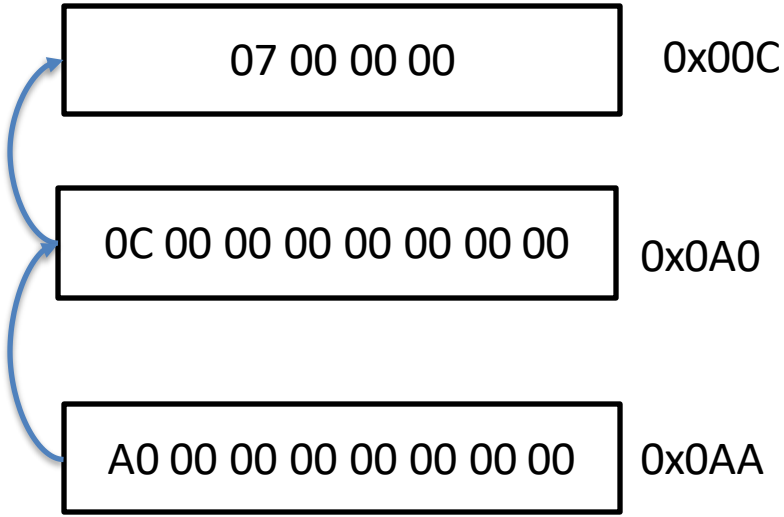
```
char *b = “Daniel”;
```

POINTER TO A POINTER

```
int **x;
```

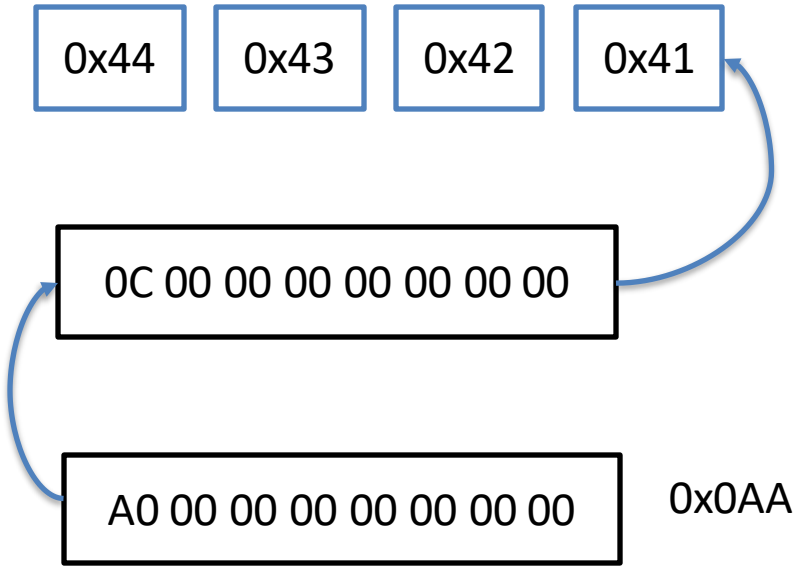
POINTER TO A POINTER

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



POINTER TO POINTER

```
char *abc = “ABCD”;
char **myPhrase = &abc;
```



LET'S IMPLEMENT STRING TOUPPER

Let's write a function that takes in a string and converters 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

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 @)

STRING LITERALS IN C

```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
```

GNU nano 6.3 toupper.c Modified

```
#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
}
```

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

dgg6b@portal07:~$

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 arrayVSpointer.c

```c
#include <stdio.h>

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

Stored as a string in code

[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

GNU nano 6.3 arrayVSpointer.s

```
        .text
        .file   "arrayVSpointer.c"
        .globl  main                                    # -- Beg>
        .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>
```

[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                arrayVSpointer.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                arrayVSpointer.s
        .globl  main                                    # -- Beg
        .p2align        4, 0x90
        .type   main,@function
main:                                   # @main
        .cfi_startproc
# %bb.0:
        subq    $24, %rsp
        .cfi_def_cfa_offset 32
        movabsq $31091192681621864, %rax        # imm =
        movq    %rax, 13(%rsp)
        movabsq $7956005065853857619, %rax      # imm =
        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 functio
        .type   .L__const.main.pointer,@object  # @__con
        .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
```

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

```
[2] 0:nano*                                  "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 > 0){
                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));
        }
}
```

./a.out Hello

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 defines a read only section of memory.

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

NOT REALLY THE SAME AS #DEFINE

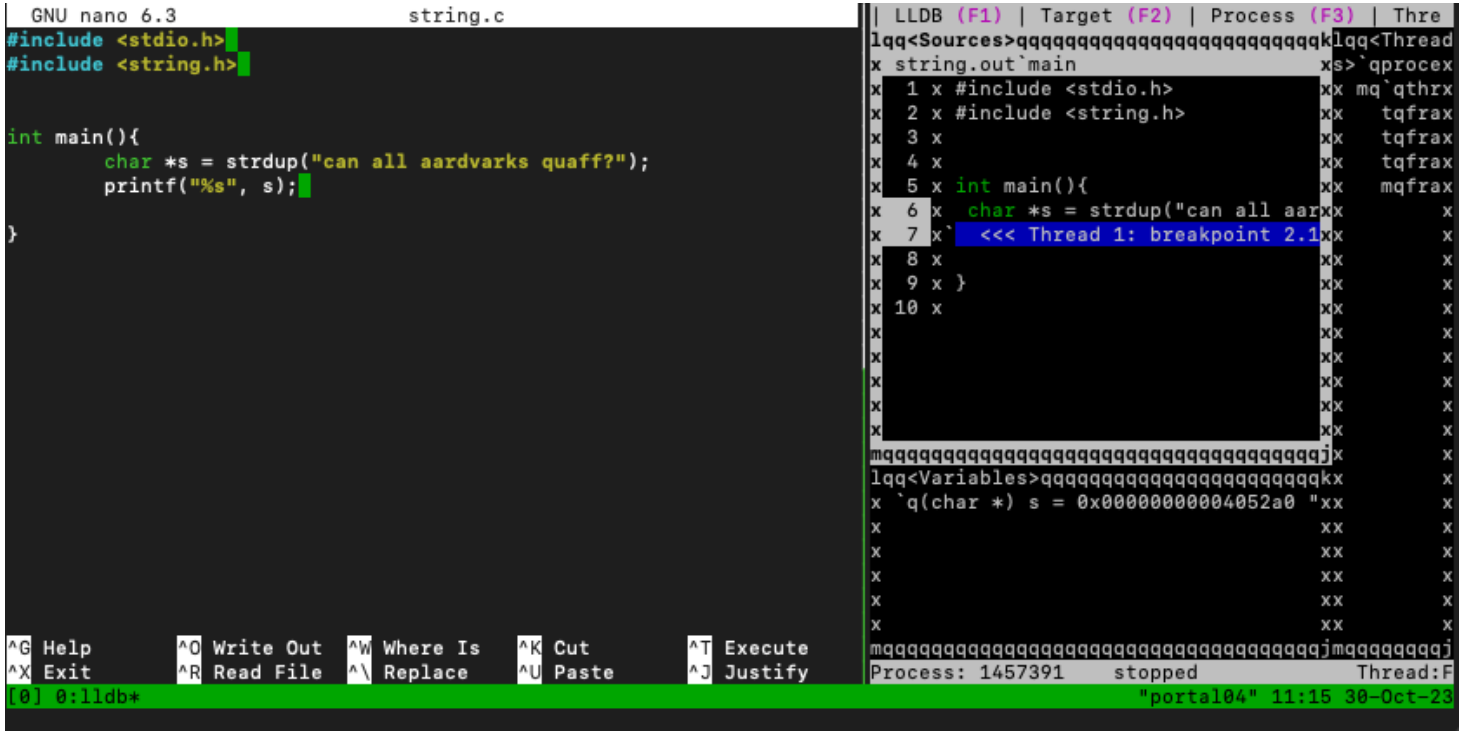
```
const int x  = 10;
```

Type information

```
#define x 10
```

No type information

STRING HELPER FUNCTIONS <STRING.H>



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]);`

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!