# COMPUTER SYSTEMS AND ORGANIZATION
## Type Def and Structs

Daniel G. Graham Ph.D

UNIVERSITY *of* VIRGINIA | ENGINEERING

# Contents

1. Exam Review question
2. Sizeof (Correction)
3. Array of Strings
4. Two-Dimensional Array
5. Type Def
6. Structs
7. Array of Structs

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]);`

<span style="color:red">y, h, i!</span>

# CORRECTION ON SIZEOF COMMENT

Sizeof() – returns the total number of bytes in the array

```
int x[4] = {1,2,3,4};
int totalNumberOfBytes = sizeof(x); //4*4 = 16


char x[4] = {'A','B','C','D'};
int totalNumberOfBytes = sizeof(x); //1*4 = 4
```

# GENERAL FORMAT OF ARRAY TYPE

int x[4];

___ __[__];

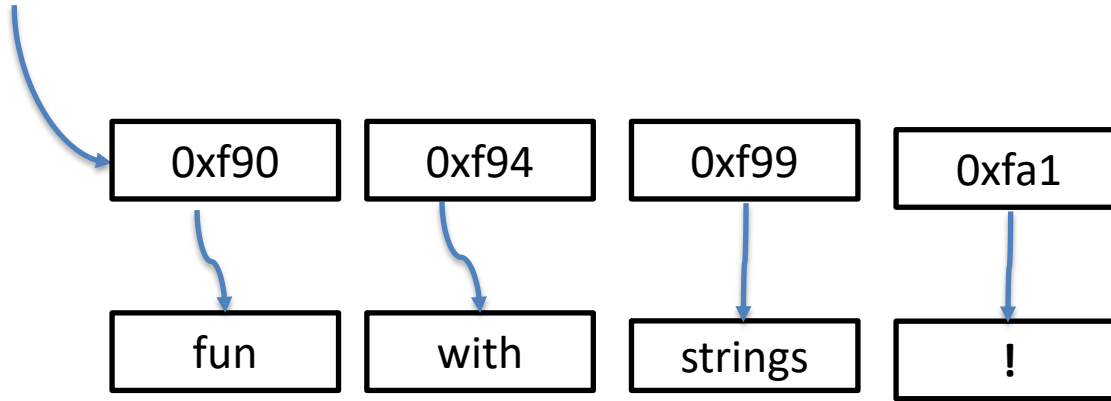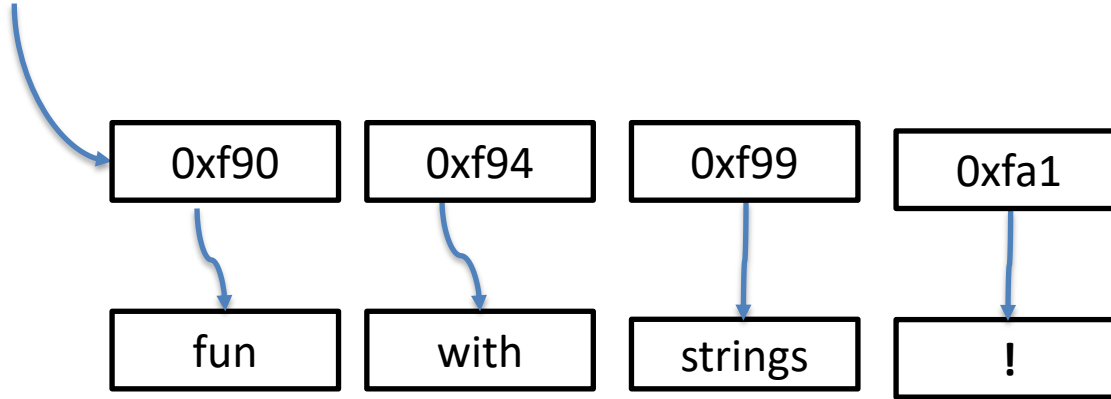type    Variable    Size
        Name

# ARRAY OF STRINGS
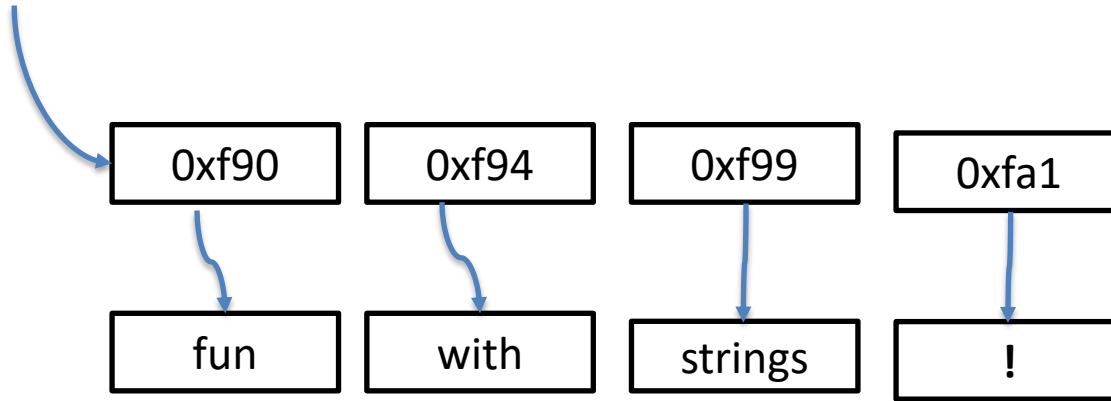
`char * strings[4] = {"fun", "with", "strings", "!"};`

| 0xf90 | 0xf94 | 0xf99 | 0xfa1 |
|---|---|---|---|
| fun | with | strings | ! |

```
char * strings[4] = {"fun", "with", "strings", "!"};
```

| 0xf90 | 0xf94 | 0xf99 | 0xfa1 |
|-------|-------|-------|-------|

| fun | with | strings | ! |
|-----|------|---------|---|

Get the third letter of the second word

```
strings[1][2] ⟷ (*(strings + 1))[2] ⟷ *((*(strings + 1)) + 2)
```

```
char * strings[4] = {"fun", "with", "strings", "!"};
```

| 0xf90 | 0xf94 | 0xf99 | 0xfa1 |
|-------|-------|-------|-------|

| fun | with | strings | ! |
|-----|------|---------|---|

Get the third letter of the second word

```
char **x = strings;
```
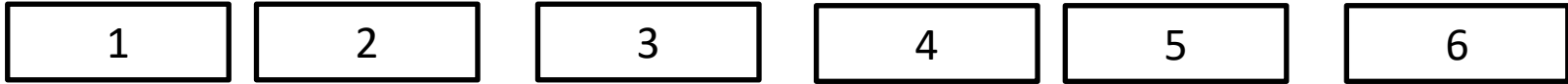
$$(*(x + 1))[2] \longleftrightarrow *((*(x + 1)) + 2)$$

UNIVERSITY *of* VIRGINIA | ENGINEERING

# TWO-DIMENSIONAL ARRAYS

```
int matrix[2][3] = { {1, 2, 3}, {4, 5, 6} };
```

|         | Column 0 | Column 1 | Column 2 |
|---------|----------|----------|----------|
| Row 0   | 1        | 2        | 3        |
| Row 1   | 4        | 5        | 6        |

ENGINEERING

# TWO-DIMENSIONAL ARRAYS

```
int matrix[2][3] = { {1, 2, 3}, {4, 5, 6} };
```

| 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|

"Row-Major" ordering of all elements in memory

# TWO-DIMENSIONAL ARRAYS

```
int count1 = { 1, 2, 3};
int count1 = { 4, 5, 6 };

int * matrix= { count1, count2 };
```
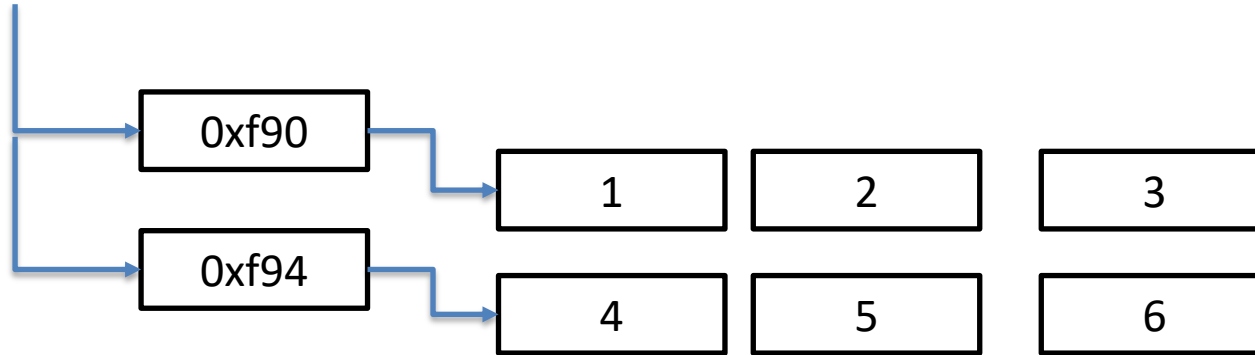
# TYPE DEF

C allows us to give types a new name using the type def keyword.

typedef _____ _____;

Type info        New name

# TYPE DEF

C allows us to give types a new name using the type def keyword.

typedef `unsigned char` `byte`;

Type info                    New name

# TYPE DEF EXAMPLES

```
typedef int dimension;

dimension height = 3;
dimension width = 4;

dimension area = width * height;

printf("%d\n", area);
```

# TYPE DEF EXAMPLES

```
typedef char * string;

string name = "Daniel";
string  AI = "maybe";

printf("%s\n", name);
```

# STRUCTS

In C, a struct (short for "structure") is a composite data type that allows you to group together variables of different data types under a single name.

```c
struct struct_name {
    data_type member1;
    data_type member2;
    // more members
}
```

# STRUCTS

```
struct student{
    int year;
    float grade;
};
```

Tag is optional
↓

```
struct _____{
    ____
    ____
};
```

Don't forget the semicolon ;

# STRUCT USAGE EXAMPLE

```
struct student{
    int year;
    float grade;
};
```

The composite type is: `struct student`

So we declare variables that are of this struct type we need to include both struct and student.

# STRUCTS USAGE EXAMPLE

```
struct student{
    int year;
    float grade;
}


struct student daniel;
```

declaring a variable of
type struct student.

ENGINEERING

# STRUCTS USAGE EXAMPLE

```
struct student{
    int year;
    float grade;
};

struct student daniel;
daniel.year = 4;
daniel.grade = 77.7;
```

Accessing members of the struct using dot syntax

# WHAT IF WE DON'T WANT TO WRITE STRUCT NAME EVERY TIME?

# TYPE DEF TO THE RESCUE

C allows us to give types a new name using the type def keyword.

```
typedef _____ _____;
```

Type info        New name

# TYPE DEF AND STRUCTS

```
struct student{
    int year;
    float grade;
};

typedef struct student studentType;
```

# USING OUR NEW TYPE

```
struct student{
    int year;
    float grade;
};

typedef struct student studentType;
studentType daniel;
studentType jane;
daniel.year = 5;
```

ENGINEERING

# WE CAN COMBINE TYPE DEF AND STRUCT DEFINITION

ENGINEERING

# TYPE DEF

C allows us to give types a new name using the type def keyword.

$$\texttt{typedef} \ \underline{\hspace{2cm}} \ \underline{\hspace{2cm}};$$

Type info    New name

# STRUCTS

```
typedef struct student{
    int year;
    float grade;
} studentType;
```

# STRUCTS

Tag is optional

```
typedef struct student{          struct _____{
    int year;                        ____
    float grade;                     ____
} studentType;                   };
```

Since we are not going to need the tag anymore and it is option we can choose to omit it.

UNIVERSITY of VIRGINIA | ENGINEERING

# STRUCTS

Tag is optional

```
typedef struct {                    struct _____{
    int year;                           ____
    float grade;                        ____
} studentType;                      };
```

Since we are not going to need the tag anymore and
it is option we can choose to omit it.

# STRUCTS

```c
typedef struct {
    char name[50];
    float grade;
} studentType;


studentType daniel;
daniel.grade = 77.7;
strcpy(daniel.name, "Daniel");
printf("name %s ", daniel.name);
Printf("grade %0.2f \n", daniel.grade);
```

# STRUCTS AND POINTERS

```
typedef struct {
    char name[50];
    float grade;
} studentType;

studentType daniel;
studentType *pointer = &daniel;
```

# STRUCTS AND POINTERS

```
typedef struct {
    char name[50];
    float grade;
} studentType;

studentType daniel;
studentType *pointer = &daniel;

(*pointer).grade = 77.7;
float grade = (*pointer).grade;
printf("grade %0.2f", grade);
```

UNIVERSITY *of* VIRGINIA | ENGINEERING

# SYNTACTIC SUGAR

```
(*pointer).member
```

```
pointer->member
```

# STRUCTS AND POINTERS

```c
typedef struct {
    char name[50];
    float grade;
} studentType;

studentType daniel;
studentType *pointer = &daniel;

pointer->grade = 77.7;
float grade = pointer->grade;
printf("grade %0.2f", grade);
```

# STRUCTS AREN'T REFERENCES

```
struct rec {
    int a[4];
    size_t i;
    struct rec *next;
};
```

r

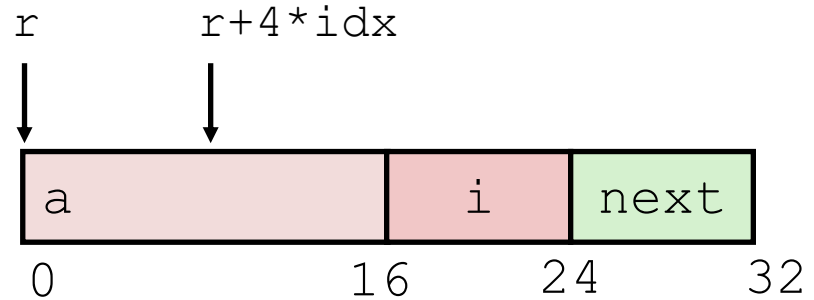| a | i | next |
|---|---|------|

0                16      24      32
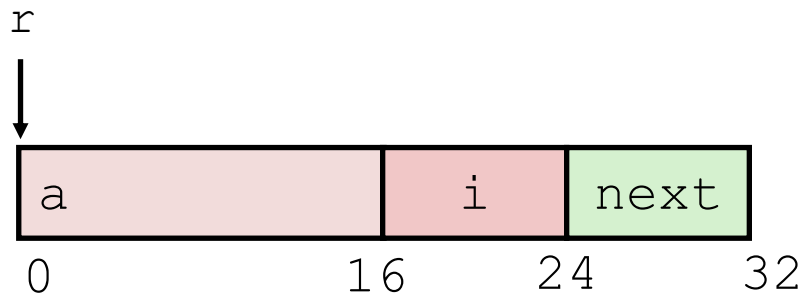
In bytes

Structs are represented as a block of memory big enough to hold all the fields

Fields are order according to order they appear in code

Machine level program has no understanding of structures. (The type is lost)
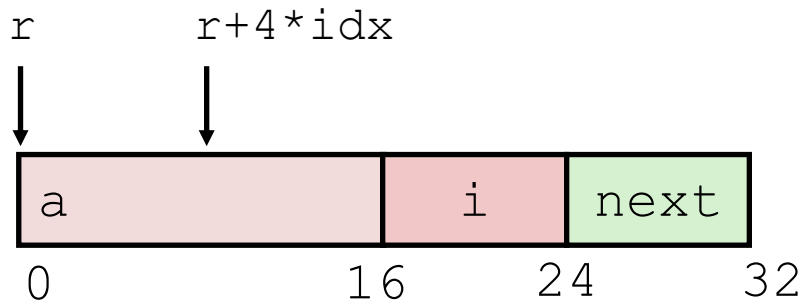
# STRUCTS AREN'T REFERENCES

```
struct rec {
    int a[4];
    size_t i;
    struct rec *next;
};
```



r          r+4*idx

| a | i | next |
0         16    24   32

```
int *get_ap(struct rec *r, size_t idx)
{
  return &r->a[idx];
}
```

# STRUCTS AREN'T REFERENCES

```
struct rec {
    int a[4];
    size_t i;
    struct rec *next;
};
```

r          r+4*idx

| a | | i | next |
|---|---|---|---|
| 0 | 16 | 24 | 32 |

```
int *get_ap(struct rec *r, size_t idx)
{
  return &r->a[idx];
}
```

```
# r in %rdi, idx in %rsi
leaq  (%rdi,%rsi,4), %rax
ret
```

ENGINEERING

# UNDEFINED BEHAVIOR

ENGINEERING

# NEXT TIME

1. Discuss Malloc and Free

2. Implement a binary tree in C

3. Do a basic in order traversal

UNIVERSITY of VIRGINIA | ENGINEERING