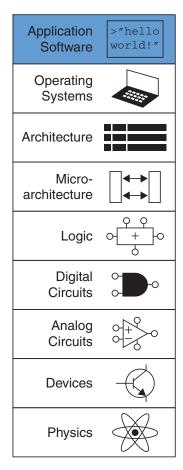


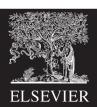
Lecture 13

- Structures
- Memory
- Pointers
- Memory Allocation
- Example: Variable Size Matrices





Digital Design and Computer Architecture: ARM[®] Edition © 2019



Structures

- Store a collection of related information
- General format:

struct name {
 type1 element1;
 type2 element2;



};

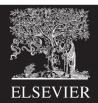


Structures

```
struct contact {
   char name[30];
   int phone;
   float height; // in meters
};
```

```
struct contact c1;
strcpy("Ben Bitdiddle", c1.name);
c1.phone = 7226993;
c1.height = 1.82;
```

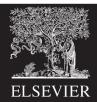




Memory

- Variables are stored in memory
- Each data type has a size
 - char 1 byte
 - short 2 bytes
 - long 4 bytes
 - int native word size of machine
 - (4 bytes on 32-bit computer)
 - float4 bytes
 - double
 8 bytes
- Arrays stored in multiple consecutive locations





Typedef

- If you're using lots of the same structure, you can shorten your typing by using typedef.
- typedef *type name*;

```
typedef struct contact {
   char name[30];
   int phone;
   float height; // in meters
} contact; // defines contact as shorthand for "struct contact"
```

contact c1; // now we can declare the variable as type contact





Structure Examples

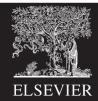
```
typedef struct point {
    int x;
    int y;
} point;
point p1;
p1.x = 42; p1.y = 9;
```

```
typedef struct rect {
  point ll;
  point ur;
  int color;
} rect;
rect r1;
r1.color = 1;
r1.ll = p1;
r1.ur.x = r1.ll.x + width;
r1.ur.y = r1.ll.y + height;
```



Digital Design and Computer Architecture: ARM[®] Edition © 2019

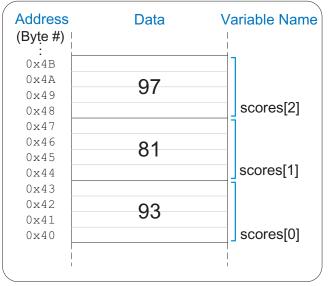
Lecture 13 <7>



Memory Example: Array

C Code Example eC.21 ARRAY INITIALIZATION AT DECLARATION USING { }

long scores[3]={93, 81, 97}; // scores[0]=93; scores[1]=81; scores[2]=97;



Address (Byte #)	Data	Variable Name
0x4B	0x00]
0x4A	0x00	
0x49	0x00	
0x48	0x61	scores[2]
0x47	0x00	1
0x46	0x00	
0x45	0x00	
0x44	0x51	scores[1]
0x43	0x00]
0x42	0x00	
0x41	0x00	
0x40	0x5D	scores[0]
] []

Figure eC.4 scores array stored in memory

Memory

Memory

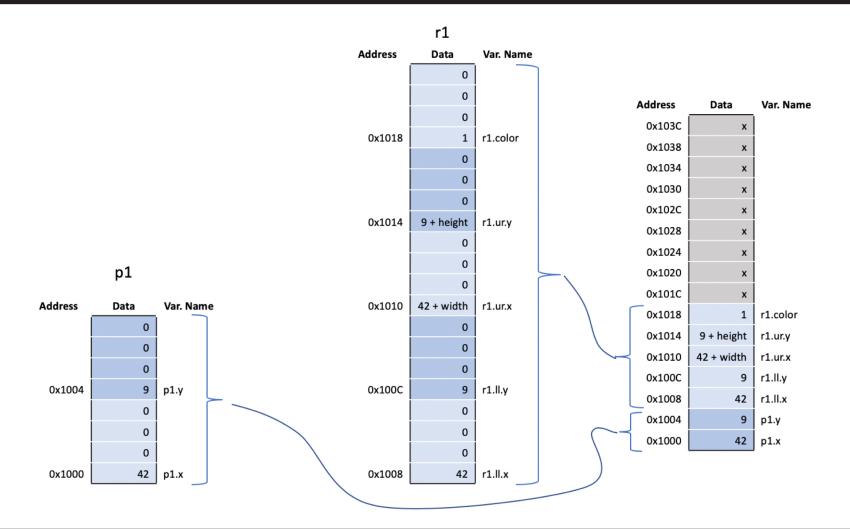


Digital Design and Computer Architecture: ARM[®] Edition © 2019

Lecture 13 <8>



Memory Example: Structure





Digital Design and Computer Architecture: ARM[®] Edition © 2019

Lecture 13 <9>



Sizeof

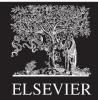
• Sizeof operator returns size of a datatype

```
char c;
double d;
point p;
rect r;
int s1 = sizeof c; // s1 = 1
int s2 = sizeof(d); // s2 = 8
int s3 = sizeof(p); // s3 = 4 + 4 = 8
int s4 = sizeof(r); // s4 = 8 + 8 + 4 = 20
```



Digital Design and Computer Architecture: ARM[®] Edition © 2019

Lecture 13 <10>



Pointers

- A pointer is an address in memory
- Pointer variables are declared with * and a data type to which the pointer points int salary1, salary2;

int *ptr; // a pointer to an integer

- & returns address of a variable
 salary1 = 98500; // suppose this is at address 100 in memory
 ptr = &salary1; // ptr contains 100 (the address of salary1)
- * dereferences a pointer (finds value it points to)
 salary2 = *ptr + 1000; // salary2 gets 99500



Digital Design and Computer Architecture: ARM[®] Edition © 2019



Arrays and Pointers

- An array in C is viewed as the address of the zeroth element
- Equivalent to a pointer to the beginning of the array



Digital Design and Computer Architecture: ARM[®] Edition © 2019



Now odd	Address	Data	Var. Name
Now add int ary[4];	0x103C	x	
int a = 37, b;	0x1038	x	
int *ptr;	0x1034	x	
int i;	0x1030	x	
	0x102C	x	
	0x1028	x	
	0x1024	x	
	0x1020	x	
	0x101C	x	
	0x1018	1	r1.color
	0x1014	9 + height	r1.ur.y
	0x1010	42 + width	r1.ur.x
	0x100C	9	r1.ll.y
	0x1008	42	r1.ll.x
	0x1004	9	p1.y
	0x1000	42	p1.x

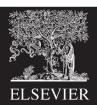


Lecture 13 <13>



Now add 0x103c x int ary[4]; // suppose at addresses 0x101C, 0x1020, 0x1024, 0x1028 0x103c x int a = 37, b; int *ptr; int i; 0x103c x 0x103c x 0x103c 0x103c x 0x102c x 0x102c 0x102 x ary[2] 0x102c x ary[1] 0x101c x ary[0] 0x101c x ary[0] 0x101d 9 + height r1.ury 0x101c 42 + width r1.urx 0x100c 9 r1.lly 0x100d 42 + r1.llx 0x100d 9 p1.y 0x100d 42 p1.x	Nowodd						Address	Data	Var. Name
int a = 37, b; 0x1038 x int *ptr; 0x1030 x int i; 0x1020 x 0x1022 x ax/3] 0x1024 x ax/2] 0x1020 x ax/2] 0x1021 1.1.color 1.1.color 0x1021 42 + width r1.ur.x 0x1020 9 r1.ll.x 0x1020 42 r1.ll.x 0x1024 9 p1.y	Now add	se at addresses	0v101C	0 √1070	0v107/	0v1078	0x103C	x	
int *ptr; 0x1034 x 0x1030 x 0x1031 x 0x1022 x 0x1024 x 0x1024 x 0x1024 x 0x1020 x 0x1021 x 0x1022 x 0x1024 x 0x1025 x 0x1026 x 0x1027 x 0x1028 x 0x1020 x 0x1020 x 0x1020 x 0x1020 x 0x1020 x 0x1010 1 1.color 1 0x1011 1 1.color 1 0x1010 1 1.color 1 0x1010 1 1.color 1 0x1000 1 1.lx 1 0x1000 1 1.lx 1 0x1000 1		se at audiesses	5 0X101C,	0,1020,	0X1024,	0X1020	0x1038	x	
int i; 0x1030 x 0x102C x 0x102C x 0x102B x ary[3] 0x102C x ary[2] 0x102C x ary[1] 0x102C x ary[1] 0x101C x r1.urx 0x100C 9 r1.llx 0x100C 9 p1y	int *ptr:						0x1034	x	
N102N102N102N102Iav[3]N102Iav[2]N102Iav[0]N101Iav[0]N101Ir1.colorN101Ir1.urxN101Ir1.urxN101Ir1.urxN102Ir1.urxN103Ir1.urxN104Ir1.urxN105IIN106IIN107IIN108IIN109II<							0x1030	x	
0x1024 (x) 0x1020 (x) 0x1020 (x) 0x1010 (x) 0x1010 (x) 0x1011 (x) 0x1012 (x) 0x1013 (x) 0x1014 (x) 0x1014 (x) 0x1010 (x) 0x1011 (x) 0x1012 (x) 0x1013 (x) 0x1014 (x) 0x1015 (x) 0x1016 (x) 0x1017 (x) 0x1018 (x) 0x1014 (x) 0x1015 (x) 0x1016 (x) 0x1017 (x) 0x1018 (x) 0x1019 (x) 0x1010 (x) 0x1011 (x) 0x1011							0x102C	x	
0x1020 (1) 0x1010 (1) 0x1011 (1) 0x1012 (1) 0x1014 (1) 0x1010 (1)							0x1028	x	ary[3]
0x101C Image: Comparison of the symbol 0x101C Image: Comparison of the symbol 0x101C Image: Symbol 0x101C <							0x1024	x	ary[2]
0x1018 1 r1.color 0x1014 9+height r1.ur.y 0x1010 42+width r1.ur.x 0x1000 09 r1.ll.y 0x1008 442 r1.ll.x 0x1004 9+height r1.ll.x 0x1005 9 p1.y							0x1020	x	ary[1]
0x1014 9 + height r1.ur.y 0x1010 42 + width r1.ur.y 0x1000 0 19 r1.ll.y 0x1004 104 r1.ll.x 0x1004 104 p1.y							0x101C	x	ary[0]
0x1010 42 + width r1.ur.x 0x1000 09 r1.ll.y 0x1008 424 r1.ll.x 0x1004 9 p1.y							0x1018	1	r1.color
0x100C 9 r1.ll.y 0x1008 42 r1.ll.x 0x1004 9 p1.y							0x1014	9 + height	r1.ur.y
0x1008 42 r1.ll.x 0x1004 9 p1.y							0x1010	42 + width	r1.ur.x
0x1004 9 p1.y							0x100C	9	r1.ll.y
							0x1008	42	r1.ll.x
0x1000 42 p1.x							0x1004	9	p1.y
							0x1000	42	p1.x





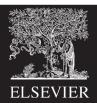
Newadd	Address	Data	Var. Name
Now add int ary[4]; // suppose at addresses 0x101C, 0x1020, 0x1024, 0x1028	0x103C	x	
\Rightarrow int a = 37, b; // suppose program places at addresses 0x1020, 0x1024, 0x1020	0x1038	x	
int *ptr;	0x1034	x	
int i;	0x1030	x	b
	0x102C	37	а
	0x1028	x	ary[3]
	0x1024	x	ary[2]
	0x1020	x	ary[1]
	0x101C	x	ary[0]
	0x1018	1	r1.color
	0x1014	9 + height	r1.ur.y
	0x1010	42 + width	r1.ur.x
	0x100C	9	r1.ll.y
	0x1008	42	r1.ll.x
	0x1004	9	p1.y
	0x1000	42	p1.x





Newadd	Address	Data	Var. Name
Now add int arv[4]; // suppose at addresses 0x101C, 0x1020, 0x1024, 0x1028	0x103C	x	
int a = 37, b; // suppose program places at addresses 0x1020, 0x1024, 0x1020	0x1038	x	
→ int *ptr; // suppose ptr is at address 0x1034, initially undefined	0x1034	x	ptr
int i;	0x1030	x	b
	0x102C	37	а
	0x1028	x	ary[3]
	0x1024	x	ary[2]
	0x1020	x	ary[1]
	0x101C	x	ary[0]
	0x1018	1	r1.color
	0x1014	9 + height	r1.ur.y
	0x1010	42 + width	r1.ur.x
	0x100C	9	r1.ll.y
	0x1008	42	r1.ll.x
	0x1004	9	p1.y
	0x1000	42	p1.x





Newedd	Address	Data	Var. Name
Now add int arv[4]; // suppose at addresses 0x101C, 0x1020, 0x1024, 0x1028	0x103C	x	
int a = 37, b; // suppose program places at addresses 0x1020, 0x1024, 0x1020 int a = 37, b; // suppose program places at addresses 0x102C, 0x1030	0x1038	x	Ĺ
int *ptr; // suppose ptr is at address 0x1034, initially undefined	0x1034	x	ptr
→ int i; // suppose at address 0x1038, initially undefined	0x1030	x	b
	0x102C	37	а
	0x1028	x	ary[3]
	0x1024	x	ary[2]
	0x1020	x	ary[1]
	0x101C	x	ary[0]
	0x1018	1	r1.color
	0x1014	9 + height	r1.ur.y
	0x1010	42 + width	r1.ur.x
	0x100C	9	r1.ll.y
	0x1008	42	r1.ll.x
	0x1004	9	p1.y



Lecture 13 <17>

0x1000



Navy add	Address	Data	Var. Name
Now add int arv[4]; // suppose at addresses 0x101C, 0x1020, 0x1024, 0x1028	0x103C	x	
int a = 37, b; // suppose program places at addresses 0x1020, 0x1020, 0x1024, 0x1020	0x1038	x	i
int *ptr; // suppose ptr is at address 0x1034, initially undefined	0x1034	x	ptr
<pre>int i; // suppose at address 0x1038, initially undefined</pre>	0x1030	x	b
	0x102C	37	а
for $(j=0; j<3; j++) ary[j] = j*j;$	0x1028	x	ary[3]
ptr = &a	0x1024	x	ary[2]
b = *ptr;	0x1020	x	ary[1]
*ptr = 3;	0x101C	x	ary[0]
ptr = arv; ptr[1] = b;	0x1018	1	r1.color
(ptr+2) = 7	0x1014	9 + height	r1.ur.y
arv[4] = 1;	0x1010	42 + width	r1.ur.x
*(ptr+5) = 2	0x100C	9	r1.ll.y
	0x1008	42	r1.ll.x
	0x1004	9	p1.y



0x1000

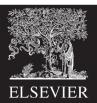


42

Newedal	Address	Data	Var. Name
Now add int ary[4]; // suppose at addresses 0x101C, 0x1020, 0x1024, 0x1028	0x103C	x	
<pre>int ary[4]; // suppose at addresses 0x101C, 0x1020, 0x1024, 0x1028 int a = 37, b; // suppose program places at addresses 0x102C, 0x1030</pre>	0x1038	3	i
int *ptr; // suppose ptr is at address 0x1034, initially undefined	0x1034	x	ptr
int j; // suppose at address 0x1038, initially undefined	0x1030	x	b
	0x102C	37	а
➡ for (j=0; j<3; j++) ary[j] = j*j;	0x1028	x	ary[3]
ptr = &a	0x1024	4	ary[2]
b = *ptr;	0x1020	1	ary[1]
<pre>*ptr = 3; ptr = arv;</pre>	0x101C	0	ary[0]
ptr = arv; ptr[1] = b;	0x1018	1	r1.color
(ptr+2) = 7	0x1014	9 + height	r1.ur.y
arv[4] = 1;	0x1010	42 + width	r1.ur.x
*(ptr+5) = 2	0x100C	9	r1.ll.y
	0x1008	42	r1.ll.x
	0x1004	9	p1.y



0x1000



42

New edd	Address	Data	Var. Name
Now add int arv[4]; // suppose at addresses 0x101C, 0x1020, 0x1024, 0x1028	0x103C	x	
int a = 37, b; // suppose program places at addresses 0x1020, 0x1024, 0x1020 int a = 37, b; // suppose program places at addresses 0x102C, 0x1030	0x1038	3	i
int *ptr; // suppose ptr is at address 0x1034, initially undefined	0x1034	0x102C	ptr
<pre>int i; // suppose at address 0x1038, initially undefined</pre>	0x1030	x	b
	0x102C	37	а
for $(i=0; i<3; i++) ary[i] = i*i;$	0x1028	x	ary[3]
⇒ ptr = &a // ptr = 0x102C	0x1024	4	ary[2]
b = *ptr;	0x1020	1	ary[1]
*ptr = 3; ptr = arv;	0x101C	0	ary[0]
ptr[1] = b;	0x1018	1	r1.color
*(ptr+2) = 7	0x1014	9 + height	r1.ur.y
arv[4] = 1;	0x1010	42 + width	r1.ur.x
*(ptr+5) = 2	0x100C	9	r1.ll.y
	0x1008	42	r1.ll.x
	0x1004	9	p1.y

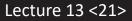


0x1000



New edd	Address	Data	Var. Name
Now add int ary[4]; // suppose at addresses 0x101C, 0x1020, 0x1024, 0x1028	0x103C	x	
int a = 37, b; // suppose program places at addresses 0x1020, 0x1020, 0x1024, 0x1020	0x1038	3	i
int *ptr; // suppose ptr is at address 0x1034, initially undefined	0x1034	0x102C	ptr
int i; // suppose at address 0x1038, initially undefined	0x1030	37	b
	0x102C	37	а
for (j=0; j<3; j++) ary[j] = j*j;	0x1028	x	ary[3]
ptr = &a // ptr = 0x102C	0x1024	4	ary[2]
<pre>b = *ptr; // dereference pointer, b = 37 *ptr = 2;</pre>	0x1020	1	ary[1]
*ptr = 3; ptr = ary;	0x101C	0	ary[0]
ptr[1] = b;	0x1018	1	r1.color
*(ptr+2) = 7	0x1014	9 + height	r1.ur.y
arv[4] = 1;	0x1010	42 + width	r1.ur.x
*(ptr+5) = 2	0x100C	9	r1.ll.y
	0x1008	42	r1.ll.x
	0x1004	9	p1.y



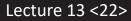


0x1000



N		Address	Data	Var. Name
Now add	// suppose at addresses 0x101C, 0x1020, 0x1024, 0x1028	0x103C	x	
	<pre>// suppose at addresses 0x1010, 0x1024, 0x1020 // suppose program places at addresses 0x102C, 0x1030</pre>	0x1038	3	i
<pre>int *ptr;</pre>	// suppose ptr is at address 0x1034, initially undefined	0x1034	0x102C	ptr
int i;	<pre>// suppose at address 0x1038, initially undefined</pre>	0x1030	37	b
		0x102C	3	а
	\underline{j} ++) $\underline{ary}[\underline{j}] = \underline{j}*\underline{j};$	0x1028	x	ary[3]
	$// ptr = 0 \times 102C$	0x1024	4	ary[2]
·····	// dereference pointer, $b = 37$	0x1020	1	ary[1]
*ptr = 3; ptr = ary;	// a = 3	0x101C	0	ary[0]
ptr[1] = b;		0x1018	1	r1.color
*(ptr+2) = 7		0x1014	9 + height	r1.ur.y
arv[4] = 1;		0x1010	42 + width	r1.ur.x
*(ptr+5) = 2		0x100C	9	r1.ll.y
		0x1008	42	r1.ll.x
		0x1004	9	p1.y





0x1000

• - I - I - - - - •



42 p1.x

.. ..

	Navy a dal		Address	Data	Var. Name
	Now add	// suppose at addresses 0x101C, 0x1020, 0x1024, 0x1028	0x103C	x	
		<pre>// suppose program places at addresses 0x1020, 0x1024, 0x1020 // suppose program places at addresses 0x102C, 0x1030</pre>	0x1038	3	i
	<pre>int *ptr;</pre>	// suppose ptr is at address 0x1034, initially undefined	0x1034	0x101C	ptr
	int i;	<pre>// suppose at address 0x1038, initially undefined</pre>	0x1030	37	b
			0x102C	3	а
		$i + j = i \cdot i;$	0x1028	x	ary[3]
		// ptr = 0x102C	0x1024	4	ary[2]
	****	// dereference pointer, $b = 37$	0x1020	1	ary[1]
_	~~~~~	// a = 3 // ptr = 0x101C	0x101C	0	ary[0]
7	ptr = arv; ptr[1] = b;		0x1018	1	r1.color
	*(ptr+2) = 7		0x1014	9 + height	r1.ur.y
	arv[4] = 1;		0x1010	42 + width	r1.ur.x
	*(ptr+5) = 2		0x100C	9	r1.ll.y
			0x1008	42	r1.ll.x
			0x1004	9	p1.y
			,	,	



0x1000

. . .



	News add		Address	Data	Var. Name
	Now add	// suppose at addresses 0x101C, 0x1020, 0x1024, 0x1028	0x103C	x	
		// suppose program places at addresses 0x1020, 0x1024, 0x1028	0x1038	3	i
	<pre>int *ptr;</pre>	// suppose ptr is at address 0x1034, initially undefined	0x1034	0x101C	ptr
	int i;	<pre>// suppose at address 0x1038, initially undefined</pre>	0x1030	37	b
			0x102C	3	а
		\underline{i} ++) $\underline{arv}[\underline{i}] = \underline{i}*\underline{i};$	0x1028	x	ary[3]
	*****	// ptr = 0x102C	0x1024	4	ary[2]
	~~~~~	<pre>// dereference pointer, b = 37</pre>	0x1020	37	ary[1]
	~~~~~	// a = 3 // ptr = 0x101C	0x101C	0	ary[0]
_		// ary[1] = 37	0x1018	1	r1.color
	*(ptr+2) = 7		0x1014	9 + height	r1.ur.y
	arv[4] = 1;		0x1010	42 + width	r1.ur.x
	*(ptr+5) = 2		0x100C	9	r1.ll.y
			0x1008	42	r1.ll.x
			0x1004	9	p1.y



0x1000

. . .



	Address	Data	Var. Name
Now add int arv[4]; // suppose at addresses 0x101C, 0x1020, 0x1024, 0x1028	0x103C	x	
int a = 37, b; // suppose program places at addresses 0x1020, 0x1020, 0x1020	0x1038	3	i
int *ptr; // suppose ptr is at address 0x1034, initially undefined	0x1034	0x101C	ptr
int i; // suppose at address 0x1038, initially undefined	0x1030	37	b
	0x102C	3	а
for (j=0; j<3; j++) ary[j] = j*j;	0x1028	x	ary[3]
ptr = &a // ptr = 0x102C	0x1024	7	ary[2]
<pre>b = *ptr; // dereference pointer, b = 37</pre>	0x1020	37	ary[1]
<pre>*ptr = 3; // a = 3 ptr = ary; // ptr = 0x101C</pre>	0x101C	0	ary[0]
ptr = arv; // ptr = 0x101C ptr[1] = b; // arv[1] = 37	0x1018	1	r1.color
<pre>*(ptr+2) = 7 // ary[2] = 7, note offset is in integer sizes, not bytes</pre>	0x1014	9 + height	r1.ur.y
arv[4] = 1;	0x1010	42 + width	r1.ur.x
*(ptr+5) = 2	0x100C	9	r1.ll.y
	0x1008	42	r1.ll.x
	0x1004	9	p1.y



Lecture 13 <25>

0x1000

. . .



Neweede	Address	Data	Var. Name
Now add int ary[4]; // suppose at addresses 0x101C, 0x1020, 0x1024, 0x1028	0x103C	x	
int a = 37, b; // suppose program places at addresses 0x1020, 0x1020, 0x1020	0x1038	3	i
int *ptr; // suppose ptr is at address 0x1034, initially undefined	0x1034	0x101C	ptr
int i; // suppose at address 0x1038, initially undefined	0x1030	37	Ь
	9x102C	1	а
for (i=0; i<3; i++) arv[i] = i*i;	0x1028	x	ary[3]
ptr = &a // ptr = 0x102C	0x1024	7	ary[2]
<pre>b = *ptr; // dereference pointer, b = 37</pre>	0x1020	37	ary[1]
*ptr = 3; // a = 3	0x101C	0	ary[0]
ptr = arv; // $ptr = 0x101Cptr[1] = b;$ // $arv[1] = 37$	0x1018	1	
<pre>ptr[1] = b; // ary[1] = 37 *(ptr+2) = 7 // ary[2] = 7, note offset is in integer sizes, not bytes</pre>	0x1014	9 + height	r1.ur.y
\Rightarrow arv[4] = 1; // a = 1, BAD: trash variable past end of array	0x1010	42 + width	r1.ur.x
*(ptr+5) = 2	0x100C	9	r1.II.y
	0x1008	42	r1.ll.x
			1



0x1004

0x1000



9 p1.y

42

p1.x

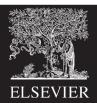
.. ..

	Navyada		Address	Data	Var. Name
	Now add int arv[4];	<pre>// suppose at addresses 0x101C, 0x1020, 0x1024, 0x1028</pre>	0x103C	x	
		<pre>// suppose ar addresses 0x101c, 0x1020, 0x1024, 0x1020 // suppose program places at addresses 0x102C, 0x1030</pre>	0x1038	3	į
	int *ptr;	// suppose ptr is at address 0x1034, initially undefined	0x1034	0x101C	ptr
	int i;	<pre>// suppose at address 0x1038, initially undefined</pre>	0x1030	2	b
			0x102C	1	а
		i + j = i * i;	0x1028	x	ary[3]
		$// ptr = 0 \times 102C$	0x1024	7	ary[2]
		<pre>// dereference pointer, b = 37</pre>	0x1020	37	ary[1]
	*****	//a = 3	0x101C	0	ary[0]
		$// ptr = 0 \times 101C$	0x1018	1	r1.color
	······	<pre>// ary[1] = 37 // ary[2] = 7, note offset is in integer sizes, not bytes</pre>	0x1014	9 + height	r1.ur.y
	·1 /	//a = 1, BAD: trash variable past end of array	0x1010	42 + width	r1.ur.x
⇒		// b = 2, BAD: trash variable past end of array	0x100C	9	r1.ll.y
ĺ	·• ·		0x1008	42	r1.ll.x
			0x1004	9	p1.y



Digital Design and Computer Architecture: ARM[®] Edition © 2019

0x1000



9 p1.y 42 p1.x

.. ..

Pointers and Structures

```
rect *rptr; // Let rptr know it's pointing to a rect
rptr = &r1; // Have rptr point at r1
```

```
(*rptr).color = 3; // Change r1.color to 3
rptr->color = 4; // Change r1.color to 4
```

// Use dot "." when you are using the structure name.
// Arrow "->" is preferred when you are using the pointer.



Digital Design and Computer Architecture: ARM[®] Edition © 2019



Passing Structures to Functions

Complex data structures and arrays are normally passed to C programs by address rather than copied; it's more efficient.

```
void createRect(int xl, int yl, int width, int height, int color, rect *r) {
   r->ll.x = x1; r->ll.y = yl;
   r->ur.x = xl + width; r->ur.y = yl + height;
   r->color = color;
}
int main(void) {
   rect r1;
   createRect(3, 5, 10, 20, 1, &r1);
}
```





Local Variable Hazard

```
void doubleWidthRect(rect *r1, rect *r2) {
  rect s;
  s.ll.x = r1.ll.x;
  s.ll.y = r1.ll.y;
  s.ur.x = (r1.ur.x - r1.ll.x) * 2 + r1.ll.x;
  s.ur.y = r1.ll.y;
  r2 = &s; // bad; s is a local variable and is lost
}
```





Solution

Be sure to declare rectangle r 2 in calling function. Then:

```
void doubleWidthRect(rect *r1, rect *r2) {
   r2.ll.x = r1.ll.x;
   r2.ll.y = r1.ll.y;
   r2.ur.x = (r1.ur.x - r1.ll.x) * 2 + r1.ll.x;
   r2.ur.y = r1.ll.y;
}
```



Digital Design and Computer Architecture: ARM[®] Edition © 2019



Multidimensional Arrays

Stored in consecutive addresses
 – last dimension first

double field[2][3][3];

Address0	Entry
0x1068	field[1][2][2]
0x1060	field[1][2][1]
0x1068	field[1][2][0]
0x1060	field[1][1][2]
0x1068	field[1][1][1]
0x1060	field[1][1][0]
0x1068	field[1][0][2]
0x1060	field[1][0][1]
0x1068	field[1][0][0]
0x1060	field[0][2][2]
0x1068	field[0][2][1]
0x1060	field[0][2][0]
0x1068	field[0][1][2]
0x1060	field[0][1][1]
0x1058	field[0][1][0]
0x1050	field[0][0][2]
0x1048	field[0][0][1]
0x1040	field[0][0][0]



Digital Design and Computer Architecture: ARM[®] Edition © 2019

Lecture 13 <32>



Complex Structures in Memory

typedef struct foo {
 double d[4][5];
 unsigned short s[16];
} foo;

foo z[10]; int s5 = sizeof(z[0]); // 8*4*5 + 2*12 = 192 = 0xC0 int s5 = sizeof(z); // 10*192 = 1920 = 0x780

Address	Entry
0x277E	z[9].s[15]
0x217E	z[1][s[15]
0x20C0	z[1].d[0][0]
0x20BE	z[0].s[15]
0x20A2	z[0].s[1]
0x20A0	z[0].s[0]
0x2098	z[0].d[3][4]
0x2008	z[0].d[0][1]
0x2000	z[0].d[0][0]





Memory Allocation

- malloc returns a pointer to allocated memory of a certain number of bytes.
- free frees this memory.
- These functions are declared in stdlib





Variable Sized Arrays

- In standard C, multidimensional array sizes must be declared at compile time.
- Treat variable-sized M row x N column array as 1dimensional array of M x N entries



Digital Design and Computer Architecture: ARM[®] Edition © 2019



Variable Dimension Matrix Example

#include <stdlib.h> // for malloc

```
double* newMatrix(int m, int n) {
  double *mat:
  mat = (double*)malloc(m*n*sizeof(double));
  return mat:
}
double* newIdentityMatrix(int n) {
  double *mat = newMatrix(n, n);
  int i, j;
  for (i=0; i<n; i++)
    for (j=0; j<n; j++)
      mat[j+i*n] = (i==j);
  return mat;
```



}



Variable Dimension Matrix Example

void scaleMatrix(double *mat, double *scaled, int m, int n, double c) {
 int i, j;

```
for (i=0; i<m; i++)
    for (j=0; j<n; j++)
        scaled[j+i*n] = mat[j+i*n]*c;
}
int main(void) {
    double *m1, *m2;
    m1 = newIdentityMatrix(3);
    m2 = newMatrix(3, 3);
    scaleMatrix(m1, m2, 3, 3, 10);
    free(m1);
}</pre>
```



