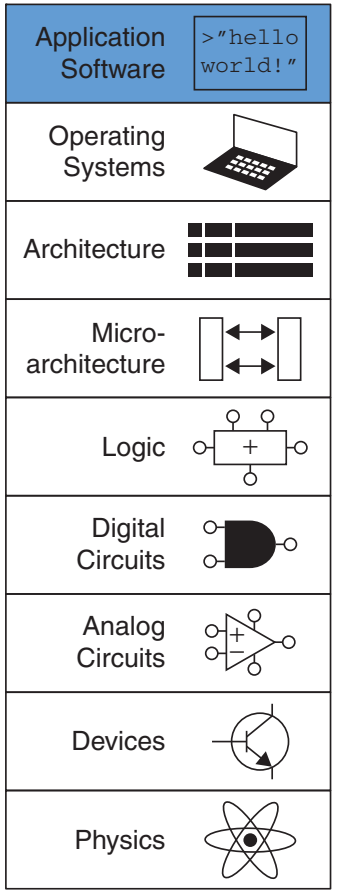
E85 Digital Design & Computer Engineering

Lecture 13:
More C Programming

HARVEY MUDD COLLEGE

Lecture 13

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



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)
  - float 4 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;
```

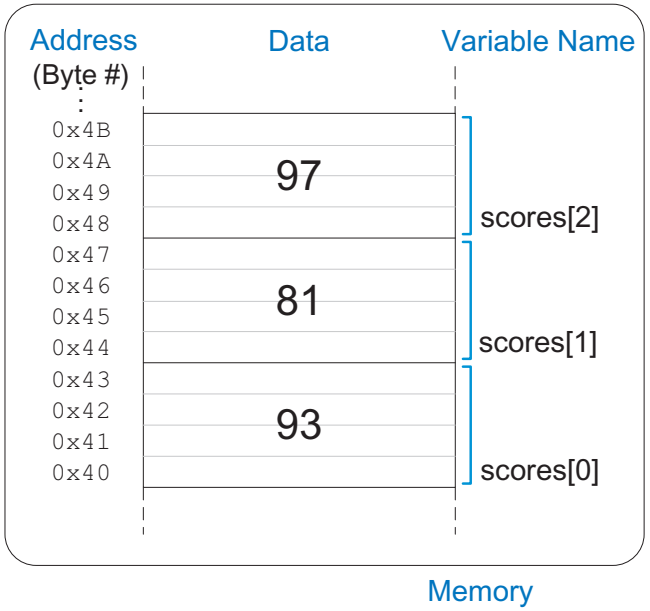
```
typedef struct rect {
  point ll;
  point ur;
  int color;
} rect;

rect r1;
r1.color = 1;
r1.ll = p1;
r1.ur.x = r1.ll.x + width;
r1.ur.y = r1.ll.y + height;
```

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;
```



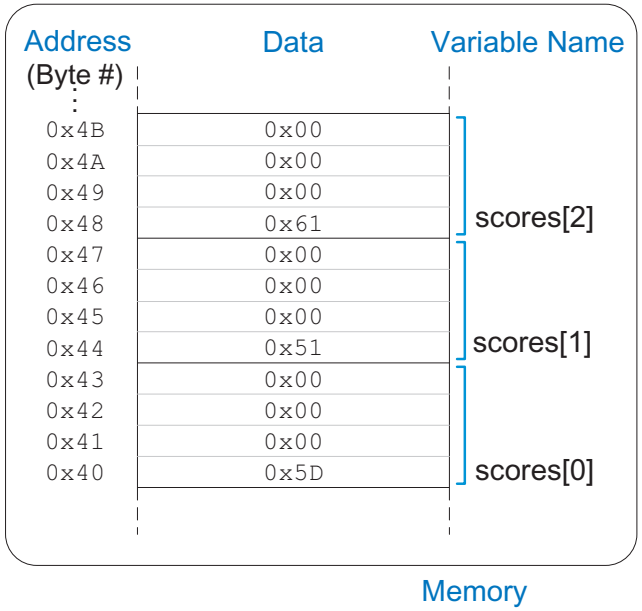
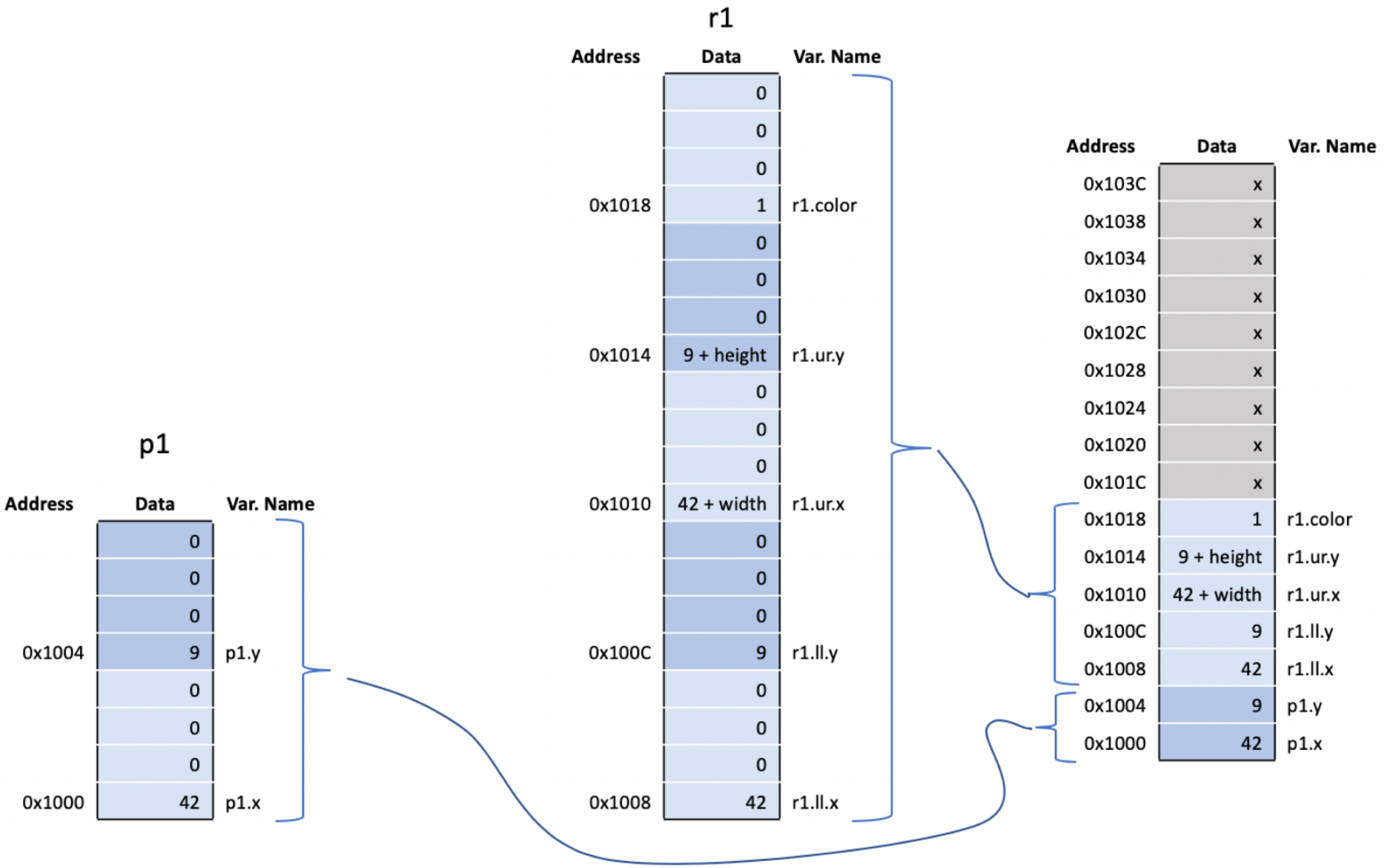


Figure eC.4 `scores` **array stored in memory**

Memory Example: Structure



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
```

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
```

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

Pointer Example

Now add

```
int arv[4];
int a = 37, b;
int *ptr;
int i;
```

Address	Data	Var. Name
0x103C	x	
0x1038	x	
0x1034	x	
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

Pointer Example

Now add

```
int ary[4];     // suppose at addresses 0x101C, 0x1020, 0x1024, 0x1028
int a = 37, b;
int *ptr;
int i;
```

Address	Data	Var. Name
0x103C	x	
0x1038	x	
0x1034	x	
0x1030	x	
0x102C	x	
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

Pointer Example

Now add

```
int ary[4];     // suppose at addresses 0x101C, 0x1020, 0x1024, 0x1028
int a = 37, b; // suppose program places at addresses 0x102C, 0x1030
int *ptr;
int i;
```

Address	Data	Var. Name
0x103C	x	
0x1038	x	
0x1034	x	
0x1030	x	b
0x102C	37	a
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

Pointer Example

Now add

```
int ary[4];     // suppose at addresses 0x101C, 0x1020, 0x1024, 0x1028
int a = 37, b; // suppose program places at addresses 0x102C, 0x1030
int *ptr;       // suppose ptr is at address 0x1034, initially undefined
int i;
```

Address	Data	Var. Name
0x103C	x	
0x1038	x	
0x1034	x	ptr
0x1030	x	b
0x102C	37	a
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

Pointer Example

Now add

```
int ary[4];     // suppose at addresses 0x101C, 0x1020, 0x1024, 0x1028
int a = 37, b; // suppose program places at addresses 0x102C, 0x1030
int *ptr;       // suppose ptr is at address 0x1034, initially undefined
int i;          // suppose at address 0x1038, initially undefined
```

Address	Data	Var. Name
0x103C	x	
0x1038	x	i
0x1034	x	ptr
0x1030	x	b
0x102C	37	a
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

Pointer Example

Now add

```
int ary[4];     // suppose at addresses 0x101C, 0x1020, 0x1024, 0x1028
int a = 37, b; // suppose program places at addresses 0x102C, 0x1030
int *ptr;       // suppose ptr is at address 0x1034, initially undefined
int i;          // suppose at address 0x1038, initially undefined

for (i=0; i<3; i++) ary[i] = i*i;
ptr = &a;
b = *ptr;
*ptr = 3;
ptr = ary;
ptr[1] = b;
*(ptr+2) = 7
ary[4] = 1;
*(ptr+5) = 2
```

Address	Data	Var. Name
0x103C	x	
0x1038	x	i
0x1034	x	ptr
0x1030	x	b
0x102C	37	a
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

Pointer Example

Now add

```
int ary[4];     // suppose at addresses 0x101C, 0x1020, 0x1024, 0x1028
int a = 37, b; // suppose program places at addresses 0x102C, 0x1030
int *ptr;       // suppose ptr is at address 0x1034, initially undefined
int i;          // suppose at address 0x1038, initially undefined

for (i=0; i<3; i++) ary[i] = i*i;
ptr = &a;
b = *ptr;
*ptr = 3;
ptr = ary;
ptr[1] = b;
*(ptr+2) = 7
ary[4] = 1;
*(ptr+5) = 2
```

Address	Data	Var. Name
0x103C	x	
0x1038	3	i
0x1034	x	ptr
0x1030	x	b
0x102C	37	a
0x1028	x	ary[3]
0x1024	4	ary[2]
0x1020	1	ary[1]
0x101C	0	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

Pointer Example

Now add

```
int ary[4];     // suppose at addresses 0x101C, 0x1020, 0x1024, 0x1028
int a = 37, b; // suppose program places at addresses 0x102C, 0x1030
int *ptr;       // suppose ptr is at address 0x1034, initially undefined
int i;          // suppose at address 0x1038, initially undefined

for (i=0; i<3; i++) ary[i] = i*i;
ptr = &a;       // ptr = 0x102C
b = *ptr;
*ptr = 3;
ptr = ary;
ptr[1] = b;
*(ptr+2) = 7
ary[4] = 1;
*(ptr+5) = 2
```

Address	Data	Var. Name
0x103C	x	
0x1038	3	i
0x1034	0x102C	ptr
0x1030	x	b
0x102C	37	a
0x1028	x	ary[3]
0x1024	4	ary[2]
0x1020	1	ary[1]
0x101C	0	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

Pointer Example

Now add

```
int ary[4];     // suppose at addresses 0x101C, 0x1020, 0x1024, 0x1028
int a = 37, b; // suppose program places at addresses 0x102C, 0x1030
int *ptr;       // suppose ptr is at address 0x1034, initially undefined
int i;          // suppose at address 0x1038, initially undefined

for (i=0; i<3; i++) ary[i] = i*i;
ptr = &a;       // ptr = 0x102C
b = *ptr;       // dereference pointer, b = 37
*ptr = 3;
ptr = ary;
ptr[1] = b;
*(ptr+2) = 7
ary[4] = 1;
*(ptr+5) = 2
```

Address	Data	Var. Name
0x103C	x	
0x1038	3	i
0x1034	0x102C	ptr
0x1030	37	b
0x102C	37	a
0x1028	x	ary[3]
0x1024	4	ary[2]
0x1020	1	ary[1]
0x101C	0	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

Pointer Example

Now add

```
int ary[4];     // suppose at addresses 0x101C, 0x1020, 0x1024, 0x1028
int a = 37, b; // suppose program places at addresses 0x102C, 0x1030
int *ptr;       // suppose ptr is at address 0x1034, initially undefined
int i;          // suppose at address 0x1038, initially undefined

for (i=0; i<3; i++) ary[i] = i*i;
ptr = &a;       // ptr = 0x102C
b = *ptr;       // dereference pointer, b = 37
*ptr = 3;       // a = 3
ptr = ary;
ptr[1] = b;
*(ptr+2) = 7
ary[4] = 1;
*(ptr+5) = 2
```

Address	Data	Var. Name
0x103C	x	
0x1038	3	i
0x1034	0x102C	ptr
0x1030	37	b
0x102C	3	a
0x1028	x	ary[3]
0x1024	4	ary[2]
0x1020	1	ary[1]
0x101C	0	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

Pointer Example

Now add

```
int ary[4];     // suppose at addresses 0x101C, 0x1020, 0x1024, 0x1028
int a = 37, b; // suppose program places at addresses 0x102C, 0x1030
int *ptr;       // suppose ptr is at address 0x1034, initially undefined
int i;          // suppose at address 0x1038, initially undefined

for (i=0; i<3; i++) ary[i] = i*i;
ptr = &a;       // ptr = 0x102C
b = *ptr;       // dereference pointer, b = 37
*ptr = 3;       // a = 3
ptr = ary;      // ptr = 0x101C
ptr[1] = b;
*(ptr+2) = 7
ary[4] = 1;
*(ptr+5) = 2
```

Address	Data	Var. Name
0x103C	x	
0x1038	3	i
0x1034	0x101C	ptr
0x1030	37	b
0x102C	3	a
0x1028	x	ary[3]
0x1024	4	ary[2]
0x1020	1	ary[1]
0x101C	0	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

Pointer Example

Now add

```
int ary[4];     // suppose at addresses 0x101C, 0x1020, 0x1024, 0x1028
int a = 37, b; // suppose program places at addresses 0x102C, 0x1030
int *ptr;       // suppose ptr is at address 0x1034, initially undefined
int i;          // suppose at address 0x1038, initially undefined

for (i=0; i<3; i++) ary[i] = i*i;
ptr = &a;       // ptr = 0x102C
b = *ptr;       // dereference pointer, b = 37
*ptr = 3;       // a = 3
ptr = ary;      // ptr = 0x101C
ptr[1] = b;     // ary[1] = 37
*(ptr+2) = 7
ary[4] = 1;
*(ptr+5) = 2
```

Address	Data	Var. Name
0x103C	x	
0x1038	3	i
0x1034	0x101C	ptr
0x1030	37	b
0x102C	3	a
0x1028	x	ary[3]
0x1024	4	ary[2]
0x1020	37	ary[1]
0x101C	0	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

Pointer Example

Now add

```
int ary[4];     // suppose at addresses 0x101C, 0x1020, 0x1024, 0x1028
int a = 37, b; // suppose program places at addresses 0x102C, 0x1030
int *ptr;       // suppose ptr is at address 0x1034, initially undefined
int i;          // suppose at address 0x1038, initially undefined

for (i=0; i<3; i++) ary[i] = i*i;
ptr = &a;       // ptr = 0x102C
b = *ptr;       // dereference pointer, b = 37
*ptr = 3;       // a = 3
ptr = ary;      // ptr = 0x101C
ptr[1] = b;     // ary[1] = 37
*(ptr+2) = 7    // ary[2] = 7, note offset is in integer sizes, not bytes
ary[4] = 1;
*(ptr+5) = 2
```

Address	Data	Var. Name
0x103C	x	
0x1038	3	i
0x1034	0x101C	ptr
0x1030	37	b
0x102C	3	a
0x1028	x	ary[3]
0x1024	7	ary[2]
0x1020	37	ary[1]
0x101C	0	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

Pointer Example

Now add

```
int ary[4];     // suppose at addresses 0x101C, 0x1020, 0x1024, 0x1028
int a = 37, b; // suppose program places at addresses 0x102C, 0x1030
int *ptr;       // suppose ptr is at address 0x1034, initially undefined
int i;          // suppose at address 0x1038, initially undefined

for (i=0; i<3; i++) ary[i] = i*i;
ptr = &a;       // ptr = 0x102C
b = *ptr;       // dereference pointer, b = 37
*ptr = 3;       // a = 3
ptr = ary;      // ptr = 0x101C
ptr[1] = b;     // ary[1] = 37
*(ptr+2) = 7    // ary[2] = 7, note offset is in integer sizes, not bytes
ary[4] = 1;     // a = 1, BAD: trash variable past end of array
*(ptr+5) = 2
```

Address	Data	Var. Name
0x103C	x	
0x1038	3	i
0x1034	0x101C	ptr
0x1030	37	b
0x102C	1	a
0x1028	x	ary[3]
0x1024	7	ary[2]
0x1020	37	ary[1]
0x101C	0	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

Pointer Example

Now add

```
int ary[4];     // suppose at addresses 0x101C, 0x1020, 0x1024, 0x1028
int a = 37, b; // suppose program places at addresses 0x102C, 0x1030
int *ptr;       // suppose ptr is at address 0x1034, initially undefined
int i;          // suppose at address 0x1038, initially undefined

for (i=0; i<3; i++) ary[i] = i*i;
ptr = &a;       // ptr = 0x102C
b = *ptr;       // dereference pointer, b = 37
*ptr = 3;       // a = 3
ptr = ary;      // ptr = 0x101C
ptr[1] = b;     // ary[1] = 37
*(ptr+2) = 7    // ary[2] = 7, note offset is in integer sizes, not bytes
ary[4] = 1;     // a = 1, BAD: trash variable past end of array
*(ptr+5) = 2    // b = 2, BAD: trash variable past end of array
```

Address	Data	Var. Name
0x103C	x	
0x1038	3	i
0x1034	0x101C	ptr
0x1030	2	b
0x102C	1	a
0x1028	x	ary[3]
0x1024	7	ary[2]
0x1020	37	ary[1]
0x101C	0	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

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.
```

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 `r2` 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;
}
```

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]

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 1-dimensional array of M x N entries

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);
}
```