CS 33

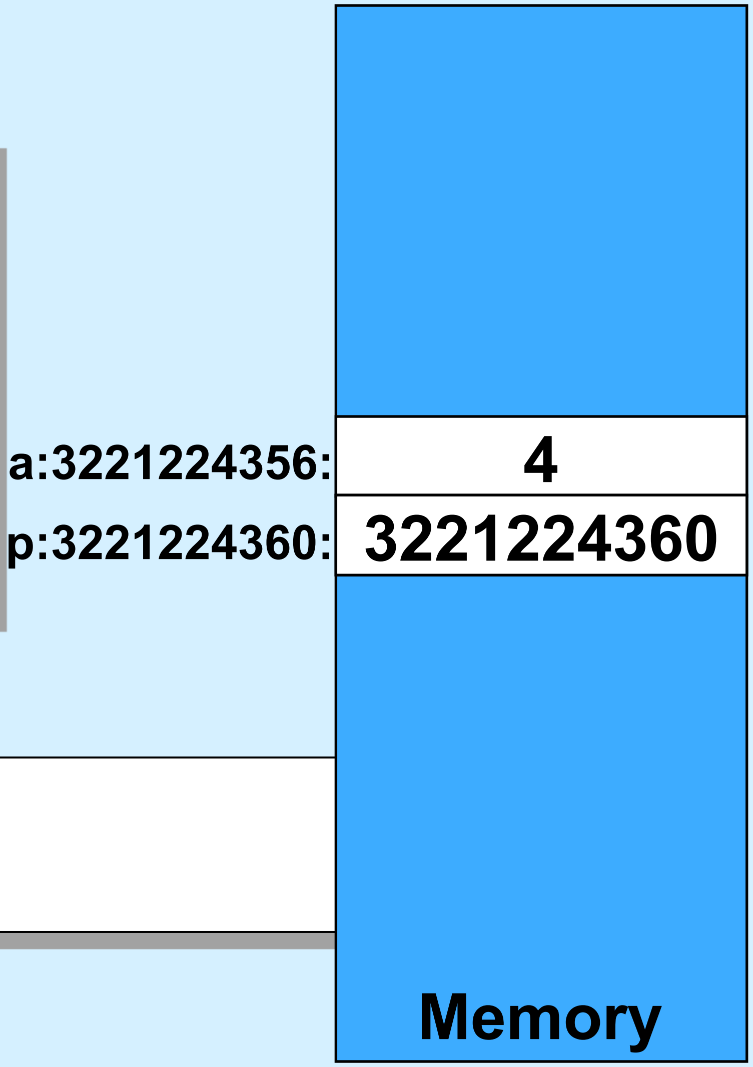
Introduction to C
Part 3

Arrays and Parameters

```
void func(int arg[]) {
   /* arg points to the caller’s array */
   int local[7];      /* seven ints */
   arg++;             /* legal */
   arg = local;       /* legal */
   local++;           /* illegal */
   local = arg;       /* illegal */
}
```

Dereferencing C Pointers

```
int main() {
   int *p; int a = 4;
   p = &a;
   *p++;
   printf("%d %u\n", *p, p);
}
```



```
% ./a.out
3221224360 3221224360
```

Dereferencing C Pointers

```
int main() {
   int *p; int a = 4;
   p = &a;
   (*p)++;
   printf("%d %u\n", *p, p);
}
```

```
% ./a.out
5 3221224356
```

Dereferencing C Pointers

```
int main() {
   int *p; int a = 4;
   p = &a;
   ++*p;
   printf("%d %u\n", *p, p);
}
```

```
% ./a.out
5 3221224356
```

Quiz 1

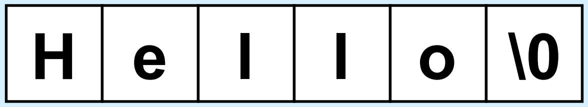
```
int proc(int arg[]) {
   arg++;
   return arg[1];
}

int main() {
   int A[3]={0, 1, 2};
   printf("%d\n",
     proc(A));
}
```

What's printed?

a) 0
b) 1
c) 2
d) indeterminate

Strings

- Strings are arrays of characters terminated by '\0' ("null")
  - the '\0' is included at the end of string constants
    - » `"Hello"`



Strings

```
int main() {
   printf("%s\n","Hello");
   return 0;
}
```

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

Strings

```
void printString(char s[]) {
   int i;
   for(i=0; s[i]!='\0'; i++)
      printf("%c", s[i]);
}
int main() {
   printString("Hello");
   printf("\n");
   return 0;
}
```

Tells C that this function does not return a value

2-D Arrays

- Suppose `T` is a datatype (such as `int`)
- `T n[6]`
  - declares `n` to be an array of (six) `T`
  - the type of `n` is `T[6]`
- Thus `T[6]` is effectively a datatype
- Thus we can have an array of `T[6]`
- `T m[7][6]`
  - `m` is an array of (seven) `T[6]`
  - `m[i]` is of type `T[6]`
  - `m[i][j]` is of type `T`

3-D Arrays

- How do we declare an array of eight `T[7][6]`?

  `T p[8][7][6]`

  - `p` is an array of (eight) `T[7][6]`
  - `p[i]` is of type `T[7][6]`
  - `p[i][j]` is of type `T[6]`
  - `p[i][j][k]` is of type `T`

2-D Arrays

```
% ./a.out
        0        1        2        3
        4        5        6        7
        8        9       10       11
```

```
#define NUM_ROWS 3
#define NUM_COLS 4
…
int main() {
   int row, col;
   int m[NUM_ROWS][NUM_COLS];
   for(row=0; row<NUM_ROWS; row++)
     for(col=0; col<NUM_COLS; col++)
        m[row][col] = row*NUM_COLS+col;
   printMatrix(NUM_ROWS, NUM_COLS, m);
   return 0;
}
```

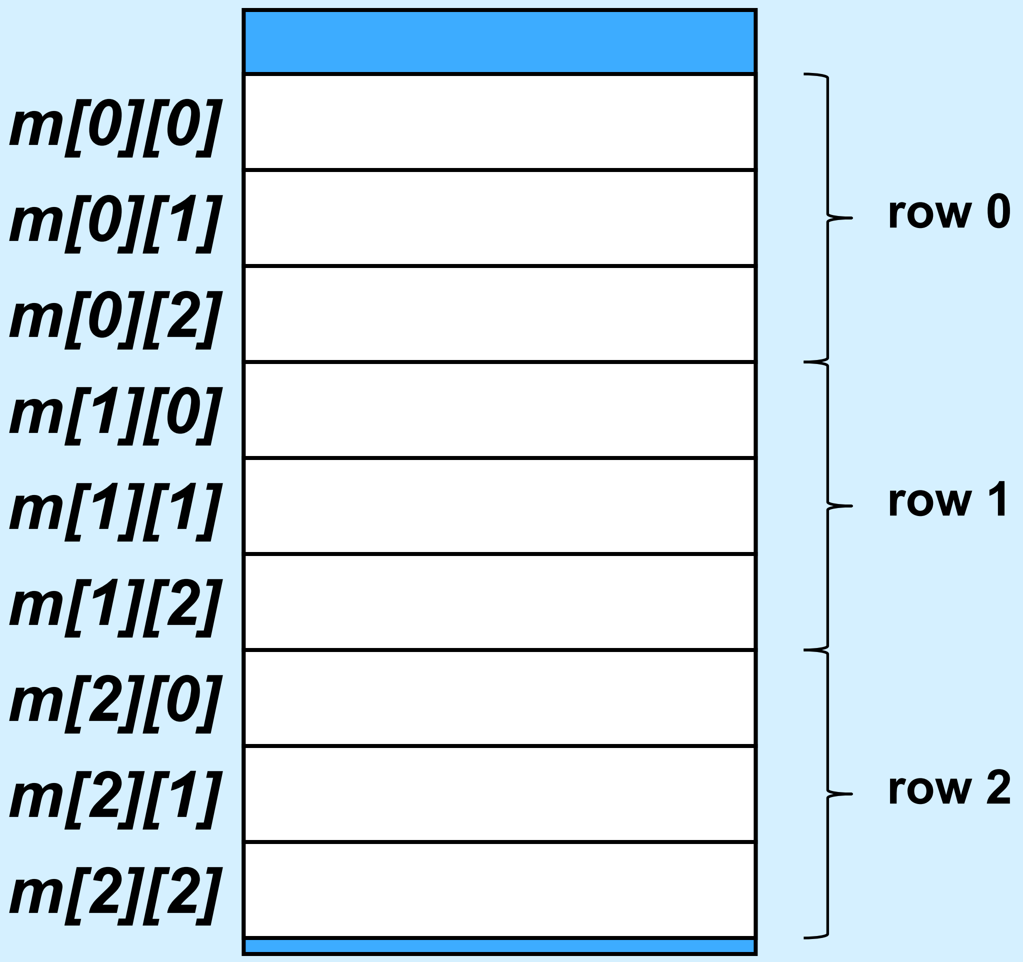
2-D Arrays

It must be told the dimensions

```
void printMatrix(int nr, int nc,
      int m[nr][nc]) {
   int row, col;
   for(row=0; row<nr; row++) {
     for(col=0; col<nc; col++)
        printf("%6d", m[row][col]);
     printf("\n");
   }
}
```

Memory Layout

```
#define NUM_ROWS 3
#define NUM_COLS 3
```



2-D Arrays

Alternatively …

```
void printMatrix(int nr, int nc,
      int m[][nc]) {
   int row, col;
   for(row=0; row<nr; row++) {
     for(col=0; col<nc; col++)
        printf("%6d", m[row][col]);
     printf("\n");
}
```

2-D Arrays

Or …

```
void printMatrix(int nr, int nc,
        int m[][nc]) {
    int i;
    for(i=0; i<nr; i++)
        printArray(nc, m[i]);
}
```

```
void printArray(int nc, int a[nc]) {
    int i;
    for(i=0; i<nc; i++)
        printf("%6d", a[i]);
    printf("\n");
}
```

Parameters

```
void func1(int A[], int size);
void func2(int *A, int size);
  /* both work fine */

void func3(int A[][], int r, int c);
void func4(int **A, int r, int c);
  /* no good: compiler doesn’t know
     the size of A’s rows, among
     other problems */
void func5(int A[][3], int r);
void func6(int r, int c, int A[][c]);
  /* both good: row sizes are known */
```

Quiz 2

1) Consider the array
   `int A[6][6];`
   - which element is adjacent to `A[0][0]` in memory?
     a) `A[0][1]`
     b) `A[1][0]`
     c) none of the above

2)
```
int *B = &A[0][0];
B[9] = 9;
```
   - which element of A was modified?
     a) `A[0][3]`
     b) `A[2][2]`
     c) `A[3][0]`
     d) none of the above

Global Variables

The scope is global;
m can be used
by all functions

```
#define NUM_ROWS 3
#define NUM_COLS 4
int m[NUM_ROWS][NUM_COLS];

int main() {
   int row, col;
   for(row=0; row<NUM_ROWS; row++)
     for(col=0; col<NUM_COLS; col++)
        m[row][col] = row*NUM_COLS+col;
   return 0;
}
```

Global Variables

```
#define NUM_ROWS 3
#define NUM_COLS 4
int m[NUM_ROWS][NUM_COLS];

int main() {
   int row, col;
   printf("%u\n", m);
   printf("%u\n", &row);
   return 0;
}
```

```
% ./a.out
8384
3221224352
```

Global Variables are Initialized!

```c
#define NUM_ROWS 3
#define NUM_COLS 4
int m[NUM_ROWS][NUM_COLS];

int main() {
   printf("%d\n", m[0][0]);
   return 0;
}
```

```
% ./a.out
0
```

Scope

```
int a;   // global variable

int main() {
   int a;   // local variable
   a = 0;
   proc();
   printf("a = %d\n", a); // what's printed?
   return 0;
}

int proc() {
   a = 1;
   return a;
}
```

```
$ ./a.out
0
```

Scope (continued)

```
int a;   // global variable

int main() {
   a = 0;
   proc(1);
   return 0;
}

int proc(int a) {
   printf("a = %d\n", a); // what's printed?
   return a;
}
```

```
$ ./a.out
1
```

Scope (still continued)

```
int a;   // global variable

int main() {
   a = 0;
   proc(1);
   return 0;
}

int proc(int a) {
   int a;
   printf("a = %d\n", a); // what's printed?
   return a;
}
```

```
$ gcc prog.c
prog.c:12:8: error: redefinition of 'a'
   int a;
       ^
```

Scope (more ...)

```
int a;   // global variable

int main() {
   {
      // the brackets define a new scope
      int a;
      a = 6;
   }
   printf("a = %d\n", a); // what's printed?
   return 0;
}
```

```
$ ./a.out
0
```

Quiz 3

```
int a;

int proc(int b) {
   {int b=4;}
   a = b;
   return a+2;
}

int main() {
   {int a = proc(6);}
   printf("a = %d\n", a);
   return 0;
}
```

- What's printed?
  a) 0
  b) 4
  c) 6
  d) 8
  e) nothing; there's a syntax error