Variables, Pointers, and Arrays

Prof. David August

COS 217

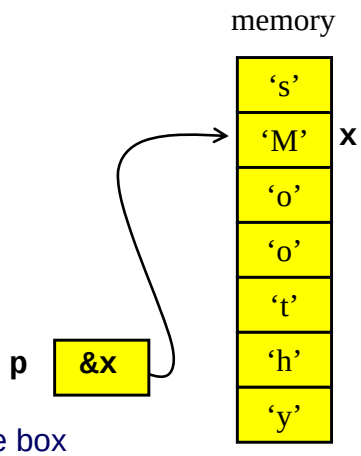
http://www.cs.princeton.edu/courses/archive/fall07/cos217/

Overview of Today's Lecture

- Pointers
  - Differences between value, variable, and pointer
  - Using pointers to do call-by-reference in C
- Struct
  - Multiple values grouped together
  - Dereferencing to access individual elements
- Arrays
  - List of elements of the same type
  - Relationship between arrays and pointers
  - Example program to reverse an array
- Strings
  - Array of characters ending in '\0'

Values, Variables, and Pointers

- Value
  - E.g., `'M'`
- Variable
  - A named box that holds a value
  - E.g., `char x = 'M';`
- Pointer value
  - Address of the box
  - E.g., `&x`
- Pointer variable
  - A box holding the address of the box
  - E.g., char* `p = &x;`



Example Program

```
#include <stdio.h>
int main(void) {
   char x = 'M';
   char* p = &x;
   printf("Value of x is %c\n", x);
   printf("Address of x is %u\n", p);
   printf("Address of p is %u\n," &p);
   return 0;
}
```

- Output
  - Value of x is M
  - Address of x is 4290770463
  - Address of p is 4290770456

Values vs. Variables

```
int n;
```
n ?

```
n = 217;
```
n 217

```
n = n + 9;
```
n 226

```
3 = n;
```
??

```
&n
```
a pointer value

```
&3
```
??

What is this?

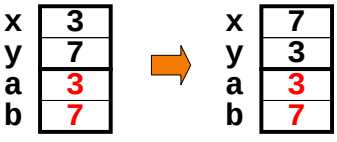
```
*(&n)
```


Call by Value is Not Enough

- Function parameters are transmitted by value
  - Values copied into "local variables"

```
void swap(int x, int  y)
{
   int t;

   t = x;
   x = y;
   y = t;
}

int main(void) {
  ...
  swap(a,b);
  ...
}
```

No!

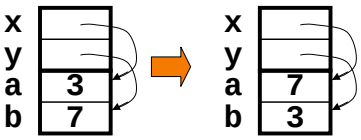


Call by Reference Using Pointers

- Use pointers to pass variables “by reference”

```
void swap(int *x, int  *y)
{
   int t;

   t = *x;
   *x = *y;
   *y = t;
}

int main(void) {
   ...
   swap(&a,&b);
   ...
}
```

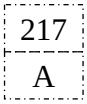
Yes



Structures

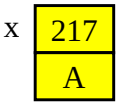
A struct value is a bunch of values glued together

```
struct pair {
  int number;
  char grade;
};
```



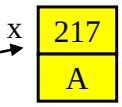
A struct variable is a box holding a struct value

```
struct pair x;
x.number = 217;
x.grade = 'A';
```



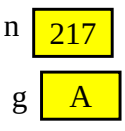
Pointers to structs

```
struct pair {int number; char grade;};
struct pair x;   x.number=217;  x.grade='A';
```



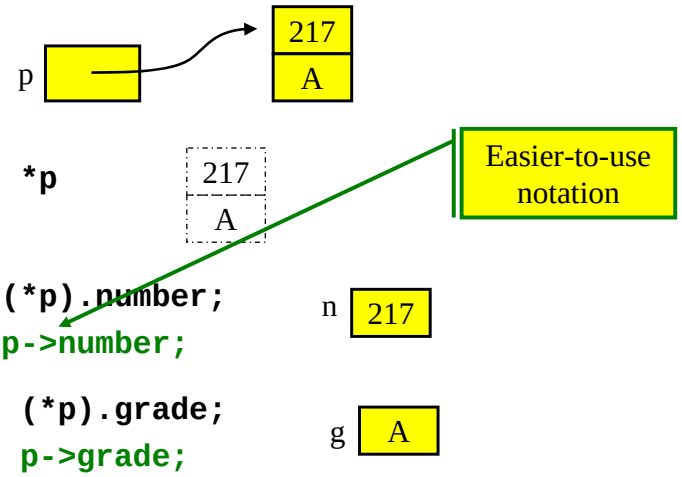

```
struct pair *p;
  p = &x;
```


```
int n = (*p).number;
char g = (*p).grade;
```



Dereferencing Fields

```
struct pair {int number; char grade;}  *p;
```




```
int n = (*p).number;
int n = p->number;
```

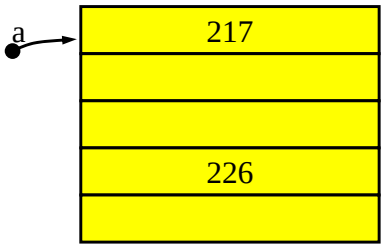
n 217

```
char g = (*p).grade;
char g = p->grade;
```

g A

Arrays in C

```
int a[5];
```



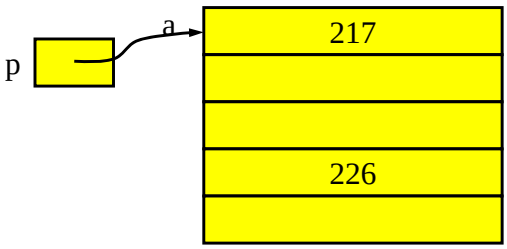
a is a *value* of type "pointer to int"

What is "**a**" in the picture above?

a is the pointer ***constant***, not the five consecutive memory locations!

Arrays and Pointers

```
int a[5];
int *p;

p = a;
```



a is a *value* of type "pointer to int" (int *)

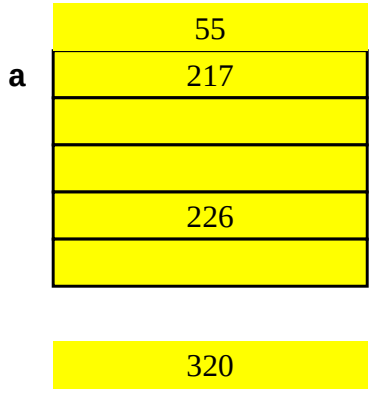
p is a *variable* of type "pointer to int" (int *)

OK: `p = a; if (a == p)...; a[i] = p[j];`

Wrong: `a = p;` `3 = i;`

C Does Not Do Bounds Checking!

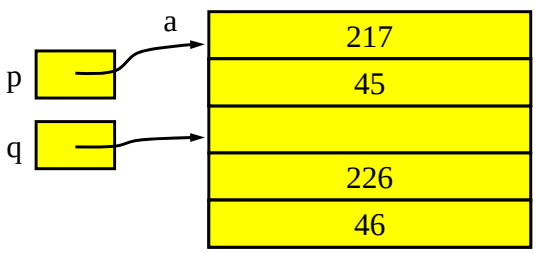
```
int a[5];

a[0] = 217;
a[3] = 226;

a[-1] = 55;
a[7] = 320;
```



Unpleasant if you happened to have another variable before the array variable **a**, or after it!

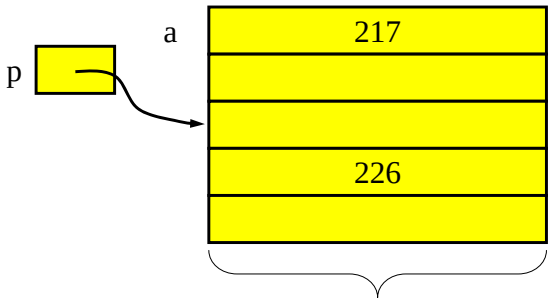
Arrays and Pointers

```
int a[5];
int *p, *q;

p = a;

p[1]= 44;

q = p + 2;

q[-1] = 45; q[2] = 46;
```



Pointer Arithmetic

```
int a[5];
```



Subscript: **a[i]** "means" ***(a+i)**

```
int *p;
p = a + 2;
```

Note: arithmetic scales by data size (e.g., int of 4 bytes)

Quaint usage of pointer arithmetic

Add up the elements of an array:

```
int a[100];
int sum, *p;
...
for (p=a; p<a+100; p++)
    sum += *p;
```

More straightforwardly:

```
int a[100];
int sum, i;
...
for (i=0; i<100; i++)
    sum += a[i];
```


Array Parameters to Functions

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

int fib[5] = {1, 1, 2, 3, 5};

int main(void)  {
  printArray(fib, 5);
}
```


Array Params ≡ Pointer Params

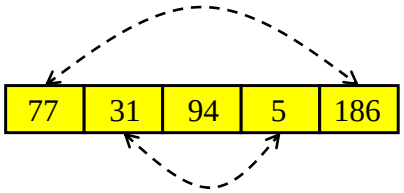
```
void printArray(int *p,    int n) { ... }
void printArray(int p[5], int n) { ... }
void printArray(int p[ ], int n) { ... }
void printArray(int p[1000], int n) { ... }
```

All these declarations are equivalent!

```
int main(void) {
  printArray(fib, 5);
}
```

Example Program: Reverse Array

- Reverse the values in an array
  - Inputs: integer array **a**, and number of elements **n**
  - Output: values of **a** stored in reverse order
- Algorithm
  - Swap the first and last elements in the array
  - Swap the second and second-to-last elements
  - …



Example of Array by Reference

```
void reverse (int a[], int n) {
  int l, r, temp;
  for (l=0, r=n-1; l<r; l++, r--) {
    temp = a[l];
    a[l] = a[r];
    a[r] = temp;
  }
}

int main(void) {
  reverse(fib, 5);
}
```


Strings

A string is just an array of characters (pointer to character), terminated by a '\0' char (a null, ASCII code 0).

char mystring[6] = {'H','e','l','l','o','\0'};
char mystring[6] = "Hello";
char mystring[] = "Hello";

Equivalent

mystring | H | e | l | l | o | \0 |

char *yourstring = "Hello";

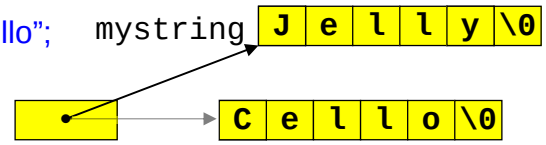
Different

yourstring → | H | e | l | l | o | \0 |

Char Array and Pointer Manipulation

char mystring[] = “Hello”;

char *yourstring = “Hello”;



mystring[0] = ‘J’;

yourstring[0] = ‘C’;

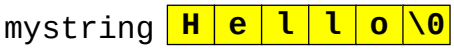
yourstring = mystring;

yourstring[4] = ‘y’;

~~mystring = yourstring;~~

Printing a String

printf(“%s”,mystring);



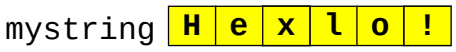

```
int i;
for (i=0; mystring[i]; i++)
   putchar(mystring[i]);
```

or,

```
char *p;
for (p=mystring; *p; p++)
   putchar(*p);
```


String termination

char mystring[] = “Hello”;



mystring[2] = 0; *equivalently,* mystring[2]=’\0’;

printf(“%s\n”,mystring);

He

mystring[2] = ‘x’; mystring[5] = ‘!’;

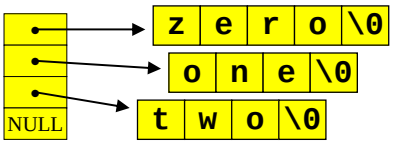
printf(“%s\n”,mystring);

What will happen?

Boxes and Arrows

In designing and analyzing your data structures, draw pictures!

Example: you want an array of strings




```
char *query[4] =
     {"zero","one","two",NULL};
```

how to parse it: `*(query[4])`

postfix operators bind tighter than prefix; whenever you're not sure, just put the parentheses in

Summary of Today's Class

- C variables
  - Pointer
  - Struct
  - Array
  - String

- Readings
  - See Course Schedule on Web page!