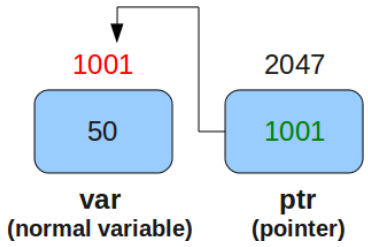
Pointers (part 1)

EECS 2031

What are pointers?

We have seen “pointers” before.

```
scanf( “%f”, &inches );
```



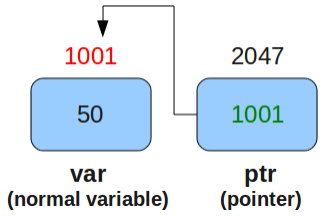
Example

```
char c;

c = getchar();

printf("%c", c);
```

```
char c;
char *p;
c = getchar();
p = &c;
printf("%c", *p);
```



3

Pointers and Addresses (5.1)

- Use the **"address of"** operator (&)
- General form:

pointer_variable = &ordinary_variable

Name of the pointer

Name of ordinary variable

4

Pointers Variables

- Pointer = memory address of a variable
- Declared with data type, * and identifier
  ```
  type *pointer_var1, *pointer_var2, …;
  ```
- Example.
  ```
  double *p;
  int *p1, *p2;
  ```
- There has to be a * before **each** of the pointer variables

5

Using a Pointer Variable

- Can be used to access a value
- Use unary operator *
  *pointer_variable
  - In executable statement, indicates value
- Example

```
int *p1, v1;
v1 = 0;
p1 = &v1;
*p1 = 42;
printf("%d\n",v1);
printf("%d\n,*p1);
```

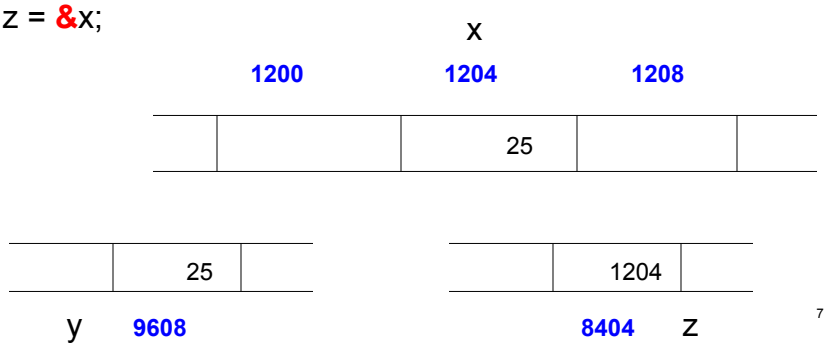
Output:
42
42

6

Pointer Example 1

```
int x, y;
int *z;
```

x = 25;
y = x;
z = &x;



7

Pointer Variables

~~z = 1024~~

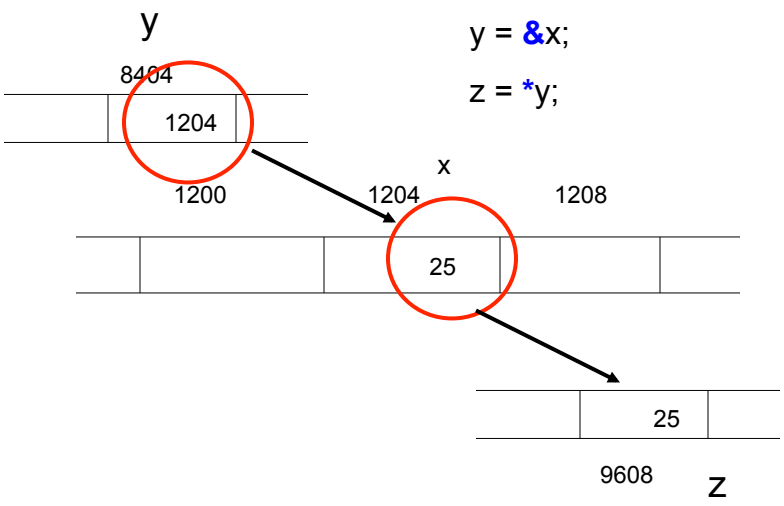
BAD idea

Instead, use z = &x

8

Pointer Example 2

```
int x = 25, *y, z;
y = &x;
z = *y;
```

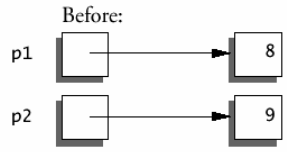


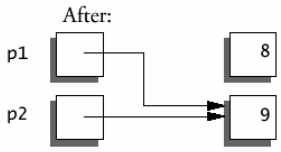
9

Pointer Example 3

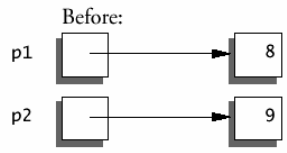
```
int *p1, *p2, x = 8, y = 9;  p1 = &x;  p2 = &y;
```

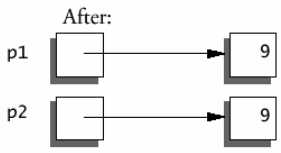
```
p1 = p2;
```






```
*p1 = *p2;
```





10

More Examples

```
int x = 1, y = 2, z[10], k;
int *ip;
ip = &x;    /* ip points to x*/
y = *ip;    /* y is now 1 */
*ip = 0;    /* x is now 0 */
z[0] = 0;
ip = &z[0]; /* ip points to z[0] */
for (k = 0; k < 10; k++)
  z[k] = *ip + k;
*ip = *ip + 100;
++*ip;
(*ip)++;    /* How about *ip++ ??? */
```

11

Precedence and Associativity

Operators	Associativity
`() [] -> .`	left to right
`! ~ ++ -- + - * (type) sizeof`	right to left
`* / %`	left to right
`+ -`	left to right
`<< >>`	left to right
`< <= > >=`	left to right
`== !=`	left to right
`&`	left to right
`^`	left to right
`|`	left to right
`&&`	left to right
`||`	left to right
`?:`	right to left
`= += -= *= /= %= &= ^= |= <<= >>=`	right to left
`,`	left to right

12

Pointers and Function Arguments (5.2)

Write a function that swaps the contents of two integers a and b.

```
void main( ) {
    int a, b;
    /* Input a and b */
    swap(a, b);
    printf("%d %d", a, b);
{
```

C passes arguments to functions by values (as Java does)

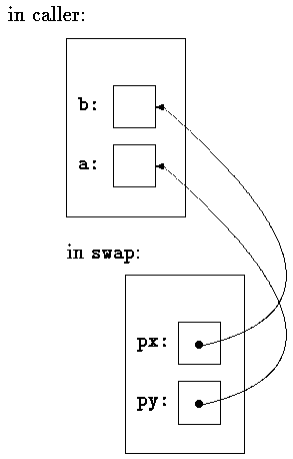
```
void swap(int x, int y)
{
   int temp;
   temp = x;
   x = y;
   y = temp;
}
```

13

The Correct Version

```
void swap(int *px, int *py)
{
   int temp;
   temp = *px;
   *px = *py;
   *py = temp;
}

void main( ) {
    int a, b;
    /* Input a and b */
    swap(&a, &b);
    printf("%d %d", a, b);
{
```



14

Pointers and Arrays

Pointers and Arrays (5.3)

- Identifier of an array is equivalent to the address of its first element.

```
int numbers[20];
int *p;

p = numbers      // valid
numbers = p      // invalid
```

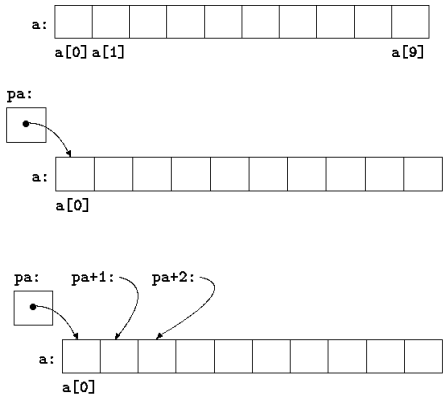
- **p** and **numbers** are equivalent and they have the same properties.
- Only difference is that we could assign another value to the pointer **p** whereas **numbers** will always point to the first of the 20 integer numbers of type int.

Pointers and Arrays: Example

```
int a[10];
/* Init a[i] = i */

int *pa;
pa = &a[0]
/*same as pa = a */
x = *pa;
/*same as x = a[0]*/

int y, z;
y = *(pa + 1);
z = *(pa + 2);
```



Pointers and Arrays: More Examples

```
int a[10], *pa;
pa = a;
/* same as pa = &a[0]*/
pa++;
/*same as pa = &a[1]*/

a[i]  ⇔    *(a+i)
&a[i] ⇔    a+i
pa[i] ⇔    *(pa+i)
```

Notes

`a = pa; a++;` are **illegal**. Think of `a` as a constant, not a modifiable variable.

`p[-1], p[-2],` etc. are syntactically legal.

Accessing Arrays Using Pointers

```c
#include <stdio.h>

int main()
{
    int data[5], i;
    printf("Enter elements: ");

    for(i = 0; i < 5; ++i)
       scanf("%d", data + i);

    printf("You entered: \n");
    for(i = 0; i < 5; ++i)
        printf("%d\n", *(data + i));

    return 0;
}
```

Note the use of (**`data + i`**)

Note the use of `*(data + i)`

19

Homework

- Lab 2, problem B: Write a C program to input a set of integers, store them in an array, find the maximum and minimum values of the set, and display those two values.
- Lab 3, problem A: Write a C program to input a line of characters and store the input characters in an array. Reverse the order of the input characters and display the reversed string on the standard output using *printf*.
- This time access elements of the arrays using pointers, not array indexing.

20

Arrays Passed to a Function

- Arrays passed to a function are passed by reference.
- The name of the array is a pointer to its first element.

21

Computing String Lengths

```
/* strlen: return length of string s */
int strlen( char *s )   /* or ( char s[] ) */
{
   int n;
   for ( n = 0; *s != '\0', s++ )
       n++;
   return n;
}

Callers:
strlen( "hello, world" ); /* string constant */
char array[100];    /* then input a string into array */
strlen( array );
char *ptr = array;
strlen( ptr );
```

22

Passing Sub-arrays to Functions

- It is possible to pass part of an array to a function, by passing a pointer to the beginning of the sub-array.

- Function

```
my_func( int arr[ ] )  {...}
or
my_func( int *arr )  {...}
```

Caller:

```
int a[100];
my_func(&a[5]);
or
my_func(a + 5);
```

23

Address Arithmetic (5.4)

Given pointers p and q of the same type and integer n, the following pointer operations are legal:

- p + n, p – n
  - n is scaled according to the size of the objects p points to. If p points to an integer of 4 bytes, p + n advances by 4*n bytes.
- q – p, q – p + 10, q – p + n (assuming q > p)
  - But p + q is illegal!
- q = p; p = q + 100;
  - If p and q point to different types, must cast first. Otherwise, the assignment is illegal!
- if (p == q), if (p != q + n)
- p = NULL;
- if (p == NULL), same as if (!p)

24

Address Arithmetic: Example

```
/* strlen: return length of string s */
int strlen(char *s)
{
  char *p = s;
  while (*p != '\0')
     p++;
  return p - s;
}
```

25

Address Arithmetic: Summary

- Legal:
  - assignment of pointers of the same type
  - adding or subtracting a pointer and an integer
  - subtracting or comparing two pointers to members of the same array
  - assigning or comparing to zero (NULL)
- Illegal:
  - add two pointers
  - multiply or divide or shift or mask pointer variables
  - add float or double to pointers
  - assign a pointer of one type to a pointer of another type (except for void *) without a cast

26

Character Pointers and Functions (5.5)

- A *string constant* (“hello world”) is an array of characters.
- The array is terminated with the null character '\0' so that programs can find the end.

```
char *pmessage;
pmessage = "now is the time";
```

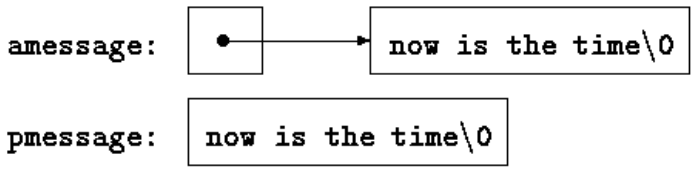
  - assigns to **pmessage** a pointer to the character array. This is *not a string copy; only pointers are involved.*
  - C does not provide any operators for processing an entire string of characters as a unit.

27

Important Difference between ...

```
char amessage[] = "now is the time"; /* an array */
char *pmessage = "now is the time"; /* a pointer */
```

- **amessage** will always refer to the same storage.
- **pmessage** may later be modified to point elsewhere.



28

Example: String Copy Function

```
/* strcpy: copy t to s; array
   subscript version */
void strcpy(char *s, char *t)
{
  int i;
  i = 0;
  while ((s[i] = t[i]) != '\0')
    i++;
}
```

```
/* strcpy: copy t to s; pointer
   version */
void strcpy(char *s, char *t)
{
  int i;
  i = 0;
  while ((*s = *t) != '\0') {
    s++; t++;
  }
}

/* strcpy: copy t to s; pointer
   version 2 */
void strcpy(char *s, char *t)
{
while ((*s++ = *t++) != '\0') ;
}
```


Dynamic Memory Allocation

EECS 2031

Dynamic Memory Allocation (7.8.5)

- How to allocate memory during run time?
- Example: input an integer n. Allocate an array of size n.

```
int n;
scanf( "%d", &n );
int my_array[ n ];   /* not allowed in C */
```

31

malloc()

- In stdlib.h

```
void *malloc( int n );
```

- Allocates memory at run time.
- Returns a pointer (to a void) to at least *n* bytes available.
- Returns null if the memory was not allocated.
- The allocated memory is not initialized.

32

Example

```
#include<stdio.h>
#include<stdlib.h>
main() {
    int *a, i, n, sum=0;
    printf( "Input an aray size  " );
    scanf( "%d", &n );
    a = malloc ( n * sizeof(int) );
    for( i=0; i<n; i++ ) scanf( "%d", &a[i] );
    for( i=0; i<n; i++ ) sum += a[i];
    free( a );
    printf("Number of elements = %d and the sum is %d\n", n, sum);
}
```


calloc()

```
void *calloc( int n, int s );
```

- Allocates an array of *n* elements where each element has size *s*;
- **`calloc`**( ) initializes the allocated memory all to 0.

realloc()

- What if we want our array to grow (or shrink)?

```
void *realloc( void *ptr, int n );
```

- Resizes a previously allocated block of memory.
- **`ptr`** must have been returned from a previous **`calloc`**, **`malloc`**, or **`realloc`**.
- The new array may be moved if it cannot be extended in its current location.

35

free()

```
void  free( void *ptr )
```

- Releases the memory we previously allocated.
- **`ptr`** must have been returned from a previous **`calloc`**, **`malloc`**, or **`realloc`**.
- C does not do automatic “garbage collection”.

36

Example

```
#include<stdio.h>
#include<stdlib.h>
main() {
    int *a, i, n, sum=0;
    printf( "Input an aray size  " );
    scanf( "%d", &n );
    a = calloc( n, sizeof(int) );
    /* a = malloc ( n * sizeof(int) ) */
    for( i=0; i<n; i++ ) scanf( "%d", &a[i] );
    for( i=0; i<n; i++ ) sum += a[i];
    free( a );
    printf("Number of elements = %d and the sum is %d\n", n, sum);
}
```

37

Next time ...

- Structures (Chapter 6)

38