

# CS61B Lecture #28

**Administrative:** Yes, there are labs (project checkpoints).

**Today:**

- Pointers in C
- Arrays
- Structures

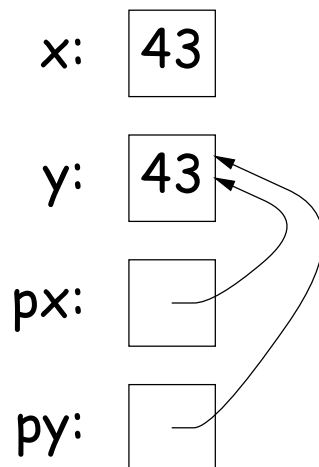
# Pointers

- One major difference from Java: can have pointers to any variable.
- Notation for types:

```
int x,y;      /* Like Java */
int* px, *py; /* pointers to integer variables */
```

- Notation for referencing and dereferencing:

```
px = &x;      /* px now contains pointer to x */
py = &y;      /* ... and py a pointer to y */
x = 42;
y = *px;      /* *px: contents of variable
               pointed to by px */
               /* y now contains 42. */
*py += 1;     /* y now contains 43. */
*px = *py;    /* x now contains 43. */
px = py;      /* px now points to y. */
```



# Heap Allocation

- No **new** operator.
- Use standard library method instead, with this idiom:

```
T* xp;  
...  
xp = (T*) malloc (sizeof(T)); /* T is some type */  
if (xp == NULL) /* Check for out of memory. */  
    take error action
```

- Objects are not destroyed automatically as in Java; programmer must clean up explicitly, or storage remains around forever:

```
free (xp);
```

# Arrays

- Arrays are very primitive. Don't keep track of their length.
- Can have names (don't have to be pointed to as in Java).
- Syntax:

```
int A[10];      /* A is an array of 10 ints. */
int B[N];      /* ILLEGAL in Standard C,
                if N is a variable */
```

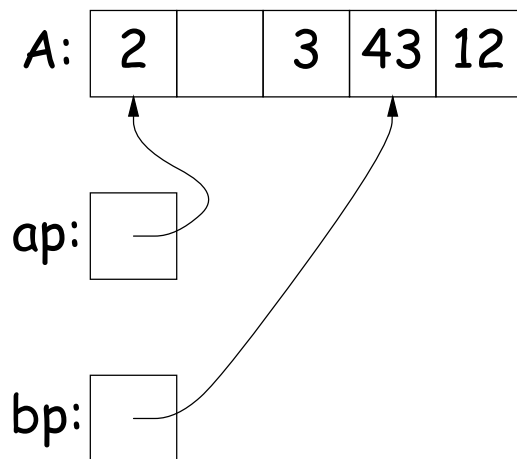
- Whole arrays are never assigned to, nor are they passed as parameters.
- Instead, any appearance of array *A* "decays" into a *pointer* to the first element in *A*:

```
int* ap = A;   /* Legal: means ap contains pointer to
                first element of array A (element #0). */
```

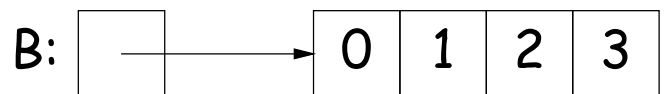
# Arrays, cont.

- Now  $ap + i$ , where  $i$  is an integer means  
"assuming that  $ap$  contains a pointer to element  $\#k$  of some array, the pointer to element  $(k + i)$ ."
- And  $ap[i]$  is short for  $*(ap+i)$ .
- Putting it together,  $ap[i]$  and  $A[i]$  refer to "element  $\#i$  of  $A$ " as you'd expect.

```
int A[5];
int N = 4, i;
int* ap = A,
      *bp = A+3,
      *B;
*ap = 1;
A[0] += 1;
*bp = 42;
A[3] += 1;
bp[-1] = 3;
*(ap+4) = 12;
```



```
B = (int*) malloc (N*sizeof(int));
for (i = 0; i < N; i += 1)
    B[i] = i;
```



# Structs

- Instead of Java classes, have *structs*, which have only instance variables—no class (static) variables, methods, or constructors.
- Syntax (I prefer alternative on right):

<code>struct Person {</code>		<code>typedef struct Person Person;</code>
<code>  char* name;</code>		<code>struct Person {</code>
<code>  int age;</code>		<code>  char* name;</code>
<code>  struct Person* next;</code>		<code>  int age;</code>
<code>};</code>		<code>  Person* next;</code>
		<code>};</code>
<code>struct Person john;</code>		<code>Person john;</code>
<code>struct Person* mary;</code>		<code>Person* mary;</code>

- The typedef defines the type name `Person` to be a synonym for `struct Person`.
- This type gives you a linked list.
- Access:

```
john.age = 42;  
mary->age = 29; /* Same as */ (*mary).age = 29;
```

## Example: List facility header file

```
/* File intlist.h */
/* Lists of ints */

#ifndef LIST_H    /* Standard idiom so that #including */
#define LIST_H    /* twice only declares once. */

typedef struct IntList IntList;
struct IntList {
    int head;
    IntList* prev;
    IntList* next;
};

/** A new singleton IntList containing X. */
extern IntList* consIntList (int x);
/** Insert CELL immediately after AFTER. */
extern void intListLink (IntList* cell, IntList* after);
/** Print L on standard output */
extern void intListPrint (IntList* L);
#endif
```

# Implementation of IntList

```
/* File list.c */
#include <stdlib.h>
#include "intlist.h"

IntList* consIntList (int x) {
    IntList* r = (IntList*) malloc (sizeof(IntList));
    r->head = x; r->next = r->prev = NULL;
    return r;
}

void intListLink (IntList* cell, IntList* after) {
    cell->next = after->next; cell->prev = after;
    if (cell->next != NULL)
        cell->next->prev = cell;
    after->next = cell;
}

void intListPrint (IntList* L) {
    IntList* p;
    for (p = L; p != NULL; p = p->next)
        printf ("%s%d", p == L ? "" : ", ", p->head);
}
```