POINTER RECAP

CSSE 120—Rose Hulman Institute of Technology

Recap: Declarations Reserve Space

- Variable declarations reserve space in memory:
  - **int x;** /* *reserves enough space for an int, names it* **x** */
  - **double d;** /* *reserves enough space for a double, names it* **d** */
- Formal parameter declarations do the same:
  - **void average(double sum, int count) {…}**
  - /* *reserves enough space for a double (named* **sum**) *and an int (named* **count**)*/

Recap: Variables with "Pointer Types" Store Addresses

- Besides holding "things" like ints and doubles, variables in C can also hold memory addresses
- Samples:
  - **int *xPtr;** /* *reserves enough space for an address, names it* **xPtr**, *says that xPtr can store the address of another variable that holds an int* */
  - **double *dPtr;** /* *reserves enough space for an address, names it* **dPtr**, *says that dPtr can store the address of another variable that holds a double* */

Recap: Pointer Operators, &

- The *address* operator, **&**:
  - **&var** gives the address where **var**'s value is stored
  - Examples:
    - **xPtr = &x;** /* *Read "***xPtr** *gets the address of* **x***"* */*
    - **dPtr = &d;** /* *Read "***dPtr** *gets the address of* **d***"* */*

Recap: Pointer Operators, *

- Use * two ways:
  - In type declarations, * says that the name refers to address of something: **int *xPtr; double *dPtr;**
  - In expressions, ***var** gives the "thing" pointed to by **var**
  - Examples:
    - **printf("%d", *xPtr);**

The format string, **"%d"**, says that we want to print an int. ***xPtr** is the thing pointed to by **xPtr**. That is, ***xPtr** is the value of **x**.

    - ***dPtr = 3.14159;**

This says that the thing pointed to by **dPtr** should get the value **3.14159**. So the result is the same as **d = 3.14159**.

Pointer Assignments

```
int x=3, y = 5;
int *px = &x;
int *py = &y;
printf("%d %d\n", x, y);
*px = 10;
printf("%d %d\n", x, y);  /* x is changed  */
px = py;
printf("%d %d\n", x, y);  /* x not changed */
*px = 12;
printf("%d %d\n", x, y);  /* y is changed  */
```

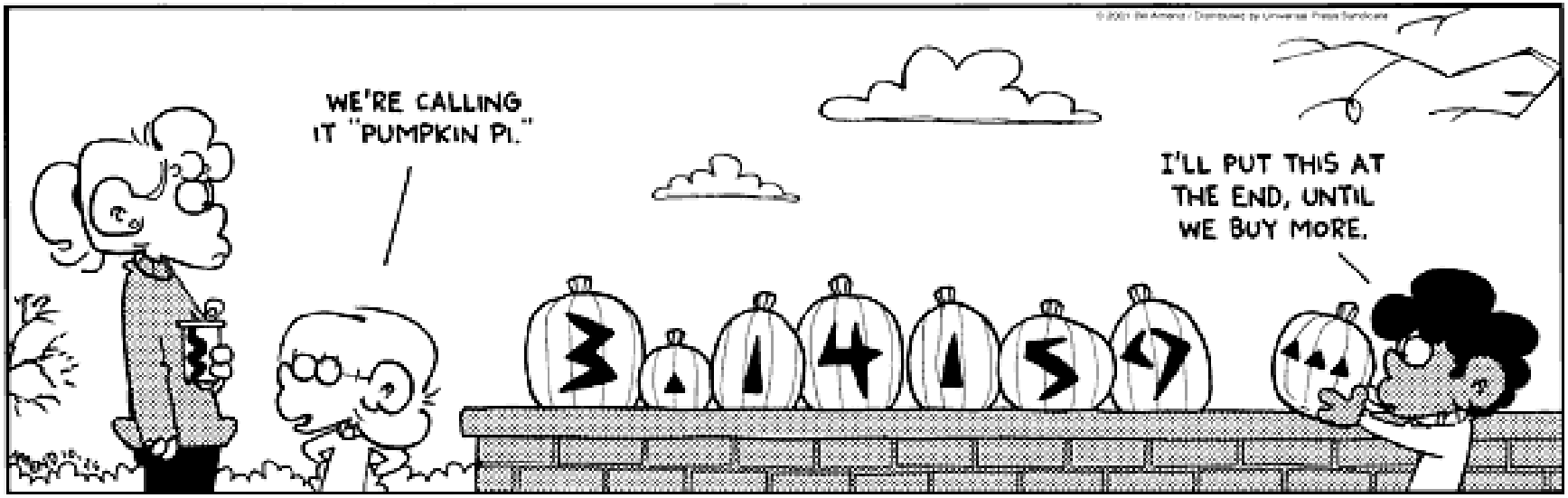
Pointer Pitfalls

- Don't try to dereference an unassigned pointer:
  - ```
    int *p;
    *p = 5;      /* oops! Program probably dies! */
    ```
- Pointer variables must be assigned *address* values.
  - ```
    int x = 3;
    int *p;
    p = x /* oops,  RHS should be &x */
    ```
- Be careful how you increment
  - ```
    *p +=1;      /* is not the same as … */
    ```
  - ```
    *p++;
    ```

Recap: Another look at the use of **&** in **scanf**

- int x, y;
- scanf("%d %d", &x, &y);
- What would happen if we used **y** instead of **&y**?

We're not Punkin' you !



Recap: Using Pointers to "Return" Multiple Results

- C only allows us to **return** one value from a function
- Can use pointers to return multiples
- Suppose we want a function that takes an array and returns the mean, min, and max values:

```
void calcStats(double values [ ], int count,
               double *mean, double *min, double *max) {
  /* … some logic omitted …*/
  *mean = meanValue;
  *min = minValue;
  *max = maxValue;
}
```

This says that the thing pointed to by **mean** should get the value stored in **meanValue**.

Arrays as function parameters

- int [ ] and int * are equivalent, when used as formal parameters in a function definition.
  - **void f (int a[], int count) { …**
  - **void f (int *a, int count) { …**
- Note that in neither case can we know the size of the array, unless it is passed in as a separate parameter.
- In either case, element 5 of **a** can be equivalently referred to as
  - a[5]
  - *(a+5)

Using a pointer to step through an array

```
int arraySum(int *a, int count) {
        int *final = a + count;
        int *current;
        int sum = 0;
        for (current = a; current < final; current++)
                sum += *current;
                return sum;
}
```

Calling the arraySum function:

```
int numArray[] = {3, 4, 5, 6, 7, 8};
printf("Array sum is %d\n", arraySum(numArray, 6));
```

A function to exchange the values of two variables

- Call it swap