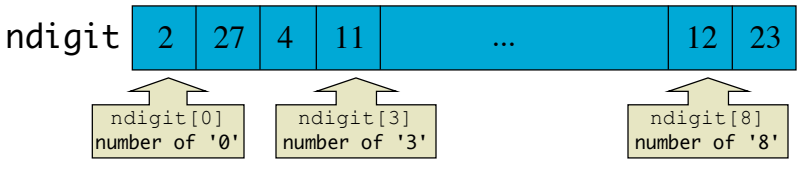
IS12 - Introduction to Programming
Lecture 18: Functions

Peter Brusilovsky
http://www2.sis.pitt.edu/~peterb/0012-051/

Using Arrays for Counting

- Idea: Use an array as a set of counting variables
- Each element will count separate events



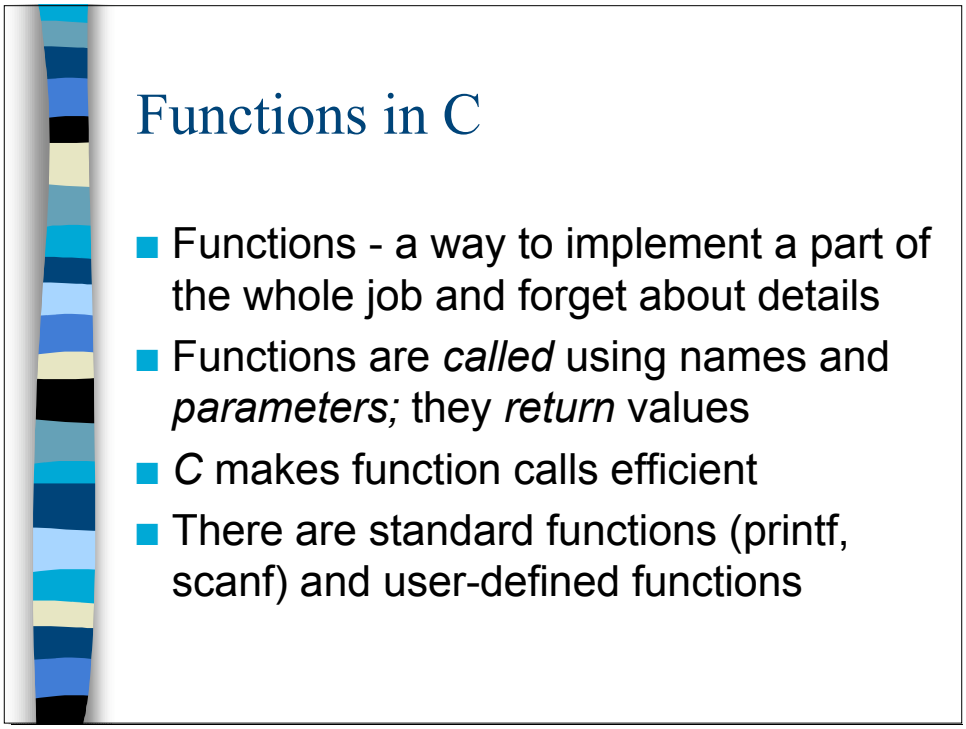
Array ndigit is used for counting digits (numeric characters)

Example: Digit Counting (1)

```
/*  Example 8.4 Digit Counting
    Source: K&R2, p.22
     count digits, white space, others */

#include <stdio.h>

void main () {
   int c, i, nwhite, nother;
   int ndigit[10];

   /* initialize */
   nwhite = nother = 0;
   for(i = 0; i < 10; ++i) ndigit[i] = 0;
```

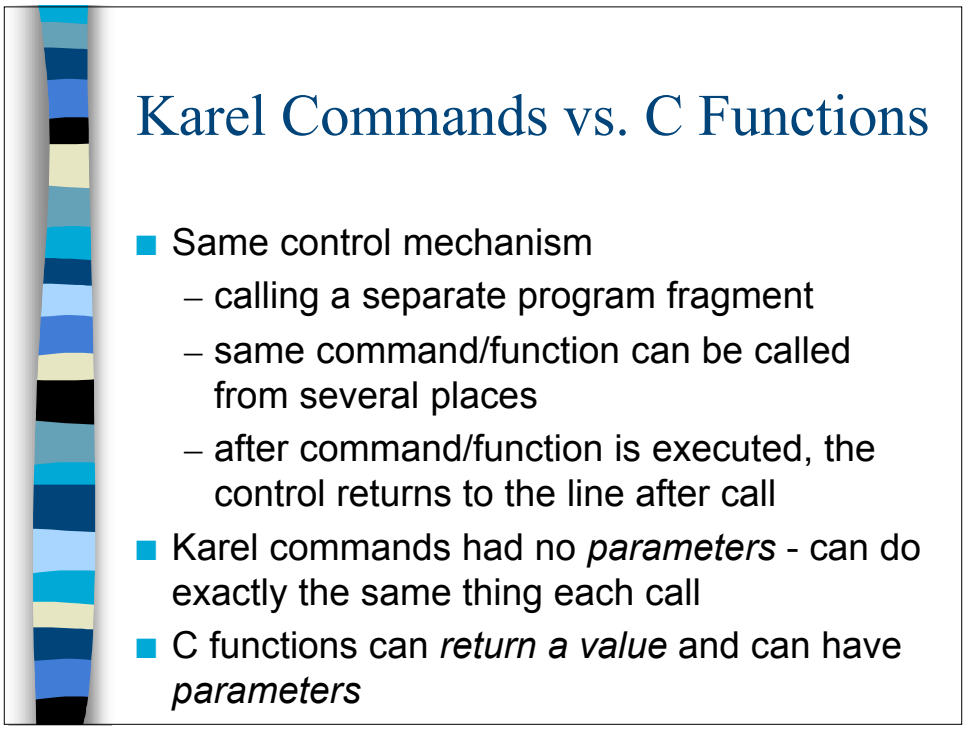
Example: Digit Counting (2)

```
   /* count */
   while ((c = getchar()) != EOF)
       if (c >= '0' && c <= '9')
              ++ndigit[c - '0'];
       else if (c == ' ' || c == '\n' || c == '\t')
              ++nwhite;
       else
              ++nother;
   /* report */
   printf("digits =");
   for(i = 0; i < 10; ++i)
       printf(" %d", ndigit[i]);
   printf(", white space = %d, other = %d\n", nwhite,
   nother);
}
```



Functions in C

- Functions - a way to implement a part of the whole job and forget about details
- Functions are *called* using names and *parameters;* they *return* values
- *C* makes function calls efficient
- There are standard functions (printf, scanf) and user-defined functions



Karel Commands vs. C Functions

- Same control mechanism
  - calling a separate program fragment
  - same command/function can be called from several places
  - after command/function is executed, the control returns to the line after call
- Karel commands had no *parameters* - can do exactly the same thing each call
- C functions can *return a value* and can have *parameters*

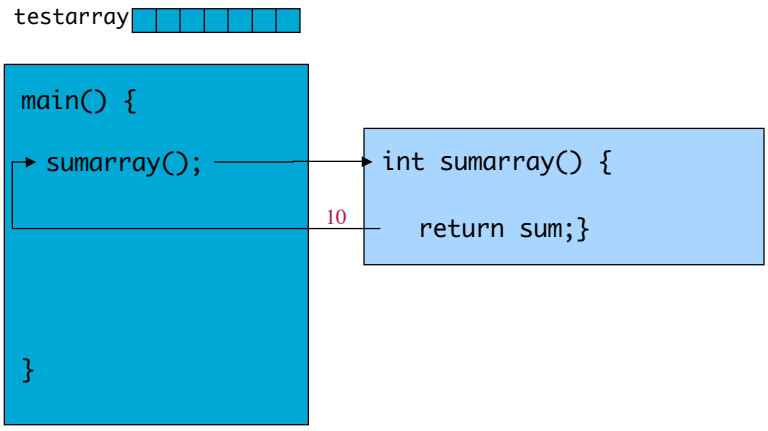
Example: summing an array

```
#include <stdio.h>
#define N 7 /* dimension of the array */
int testarray[N]; /* global array */
int sumarray(); /* function prototype */

void main() {
   int i;
   /* input */
   for (i = 0; i < N; ++i) {
        printf("%d> ", i);
        scanf("%d", &testarray[i]);
   }
```

Example: summing an array

```
   /* printing the sum of elements */
   printf("Sum of array elements = %d\n", sumarray());

}

/* function for summing array elements */
int sumarray() {
   int i, sum = 0;
   for (i = 0; i < N; ++i)
        sum += testarray[i];
   return sum;
}
```

Summing Up: Control flow



Power Example: Main Part

```
#include <stdio.h>

int power(int, int); /* function prototype */

/* test power function */
void main() {
   int i, x, y;

   for(i = 0; i < 8; ++i) {
        x = power(2, i);
        y = power (-3, i);
        printf("%2d %5d %7d\n", i, x, y);
   }
}
```

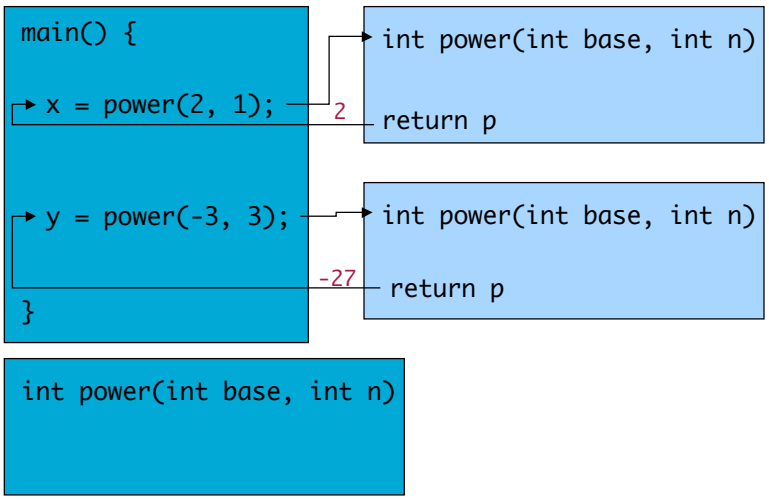
Actual Parameters

Power Example: Function

```
/* power: raise base to n-th power n >= 0
  */
int power(int base, int n) {
  int i, p;

  p = 1;
  for(i = 0; i < n; ++i)
     p = p * base;
  return p;
}
```

Formal parameters

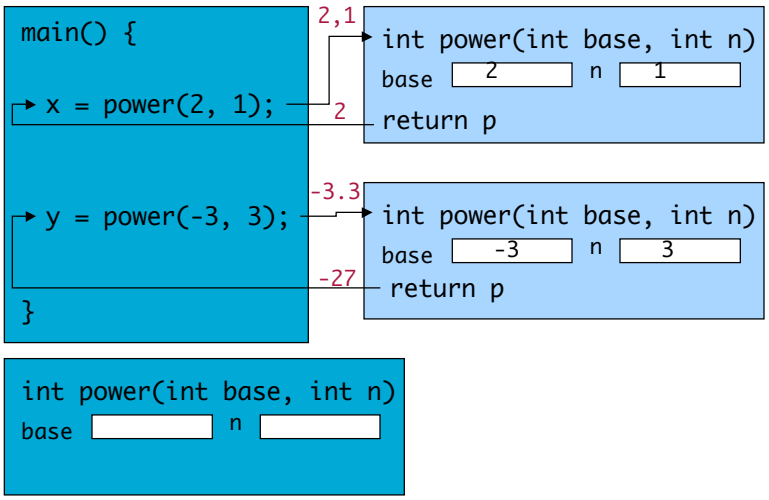
Function Call: Flow of Control



Parameter Passing by Value

- In C scalar parameters are passed to a function *by value*
- Formal parameters inside a function could be considered as local variables
- These variables are initialized with the values of actual parameters at the time of the function call
- Values of actual parameters unchanged

Function call: parameter passing



Function Prototypes

- A prototype of the function enables the compiler to check the use return value and the matching between formal and actual parameters

```
int power(int, int);
```

- The above prototype tells that power takes two arguments of type int an returns int
- In ANSI C there is no need to add names for the formal parameters as it was in old C:

```
int power(int x, int y);
```

Before next lecture:

- Do reading assignment
  - Perry: Chapter 30; First reading of Chapters 31 and 32
- Run Classroom Examples
- Use KnowledgeTree
- Exercise: Celsius to Fahrenheit table with a function: organize calculation as a function (use power as a sample)
- Exercise: Array max with a function