EECS 22: Advanced C Programming

Lecture 24

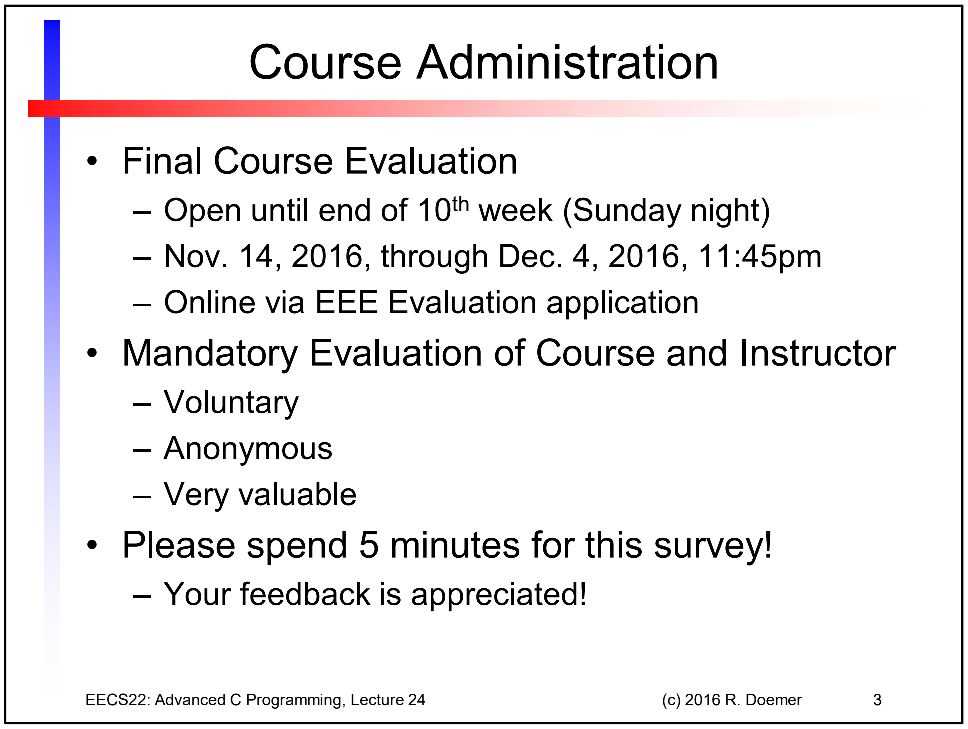
Rainer Dömer

doemer@uci.edu

The Henry Samueli School of Engineering
Electrical Engineering and Computer Science
University of California, Irvine

Lecture 24: Overview

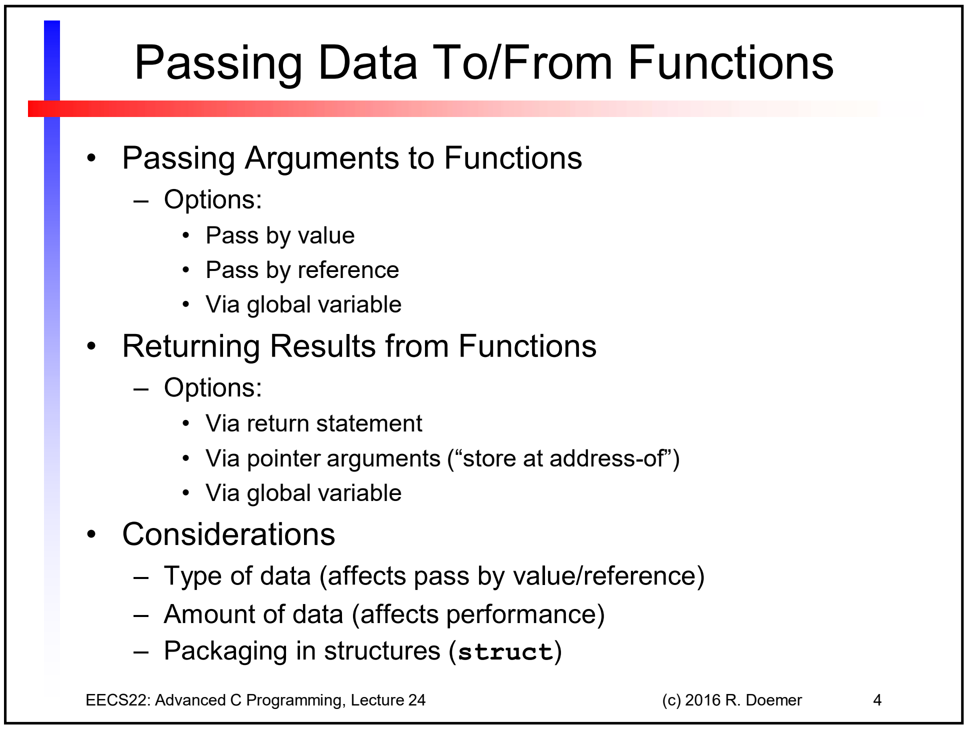
- Course Administration
  - Reminder: Final course evaluation
- Functions
  - Passing Data To/From Functions
  - Variable Argument Lists



Course Administration

- Final Course Evaluation
  - Open until end of 10th week (Sunday night)
  - Nov. 14, 2016, through Dec. 4, 2016, 11:45pm
  - Online via EEE Evaluation application
- Mandatory Evaluation of Course and Instructor
  - Voluntary
  - Anonymous
  - Very valuable
- Please spend 5 minutes for this survey!
  - Your feedback is appreciated!

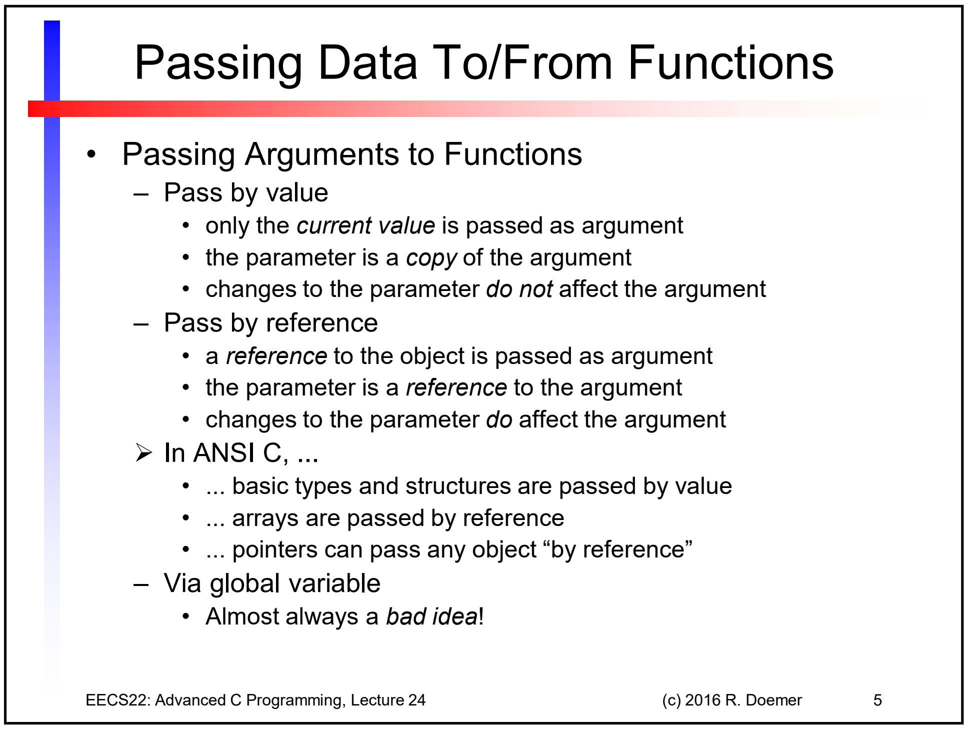
EECS22: Advanced C Programming, Lecture 24 (c) 2016 R. Doemer 3



Passing Data To/From Functions

- Passing Arguments to Functions
  - Options:
    - Pass by value
    - Pass by reference
    - Via global variable
- Returning Results from Functions
  - Options:
    - Via return statement
    - Via pointer arguments (“store at address-of”)
    - Via global variable
- Considerations
  - Type of data (affects pass by value/reference)
  - Amount of data (affects performance)
  - Packaging in structures (**`struct`**)

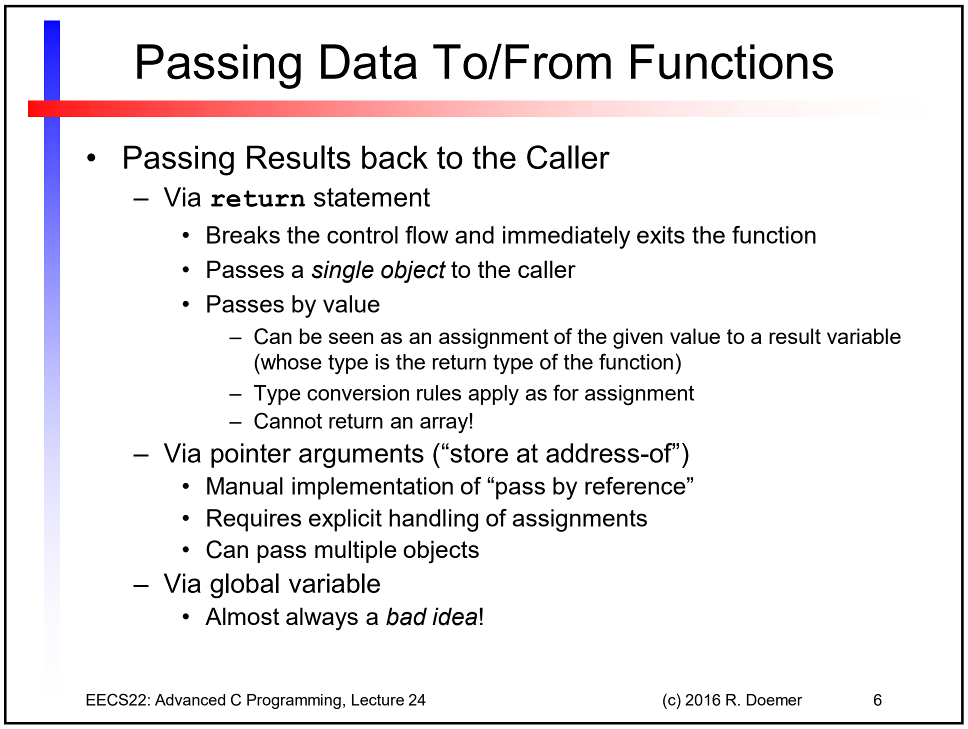
EECS22: Advanced C Programming, Lecture 24 (c) 2016 R. Doemer 4



Passing Data To/From Functions

- Passing Arguments to Functions
  - Pass by value
    - only the *current value* is passed as argument
    - the parameter is a *copy* of the argument
    - changes to the parameter *do not* affect the argument
  - Pass by reference
    - a *reference* to the object is passed as argument
    - the parameter is a *reference* to the argument
    - changes to the parameter *do* affect the argument
  - ➢ In ANSI C, ...
    - ... basic types and structures are passed by value
    - ... arrays are passed by reference
    - ... pointers can pass any object “by reference”
  - Via global variable
    - Almost always a *bad idea*!

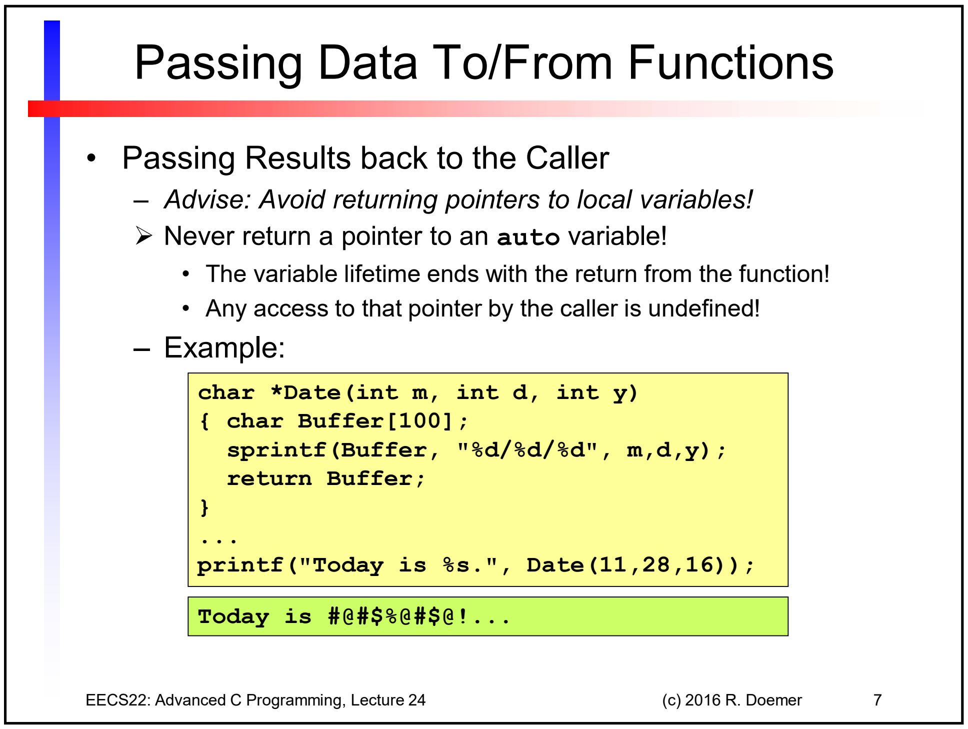
EECS22: Advanced C Programming, Lecture 24 (c) 2016 R. Doemer 5



Passing Data To/From Functions

- Passing Results back to the Caller
  - Via **`return`** statement
    - Breaks the control flow and immediately exits the function
    - Passes a *single object* to the caller
    - Passes by value
      - Can be seen as an assignment of the given value to a result variable (whose type is the return type of the function)
      - Type conversion rules apply as for assignment
      - Cannot return an array!
  - Via pointer arguments (“store at address-of”)
    - Manual implementation of “pass by reference”
    - Requires explicit handling of assignments
    - Can pass multiple objects
  - Via global variable
    - Almost always a *bad idea*!

EECS22: Advanced C Programming, Lecture 24 (c) 2016 R. Doemer 6

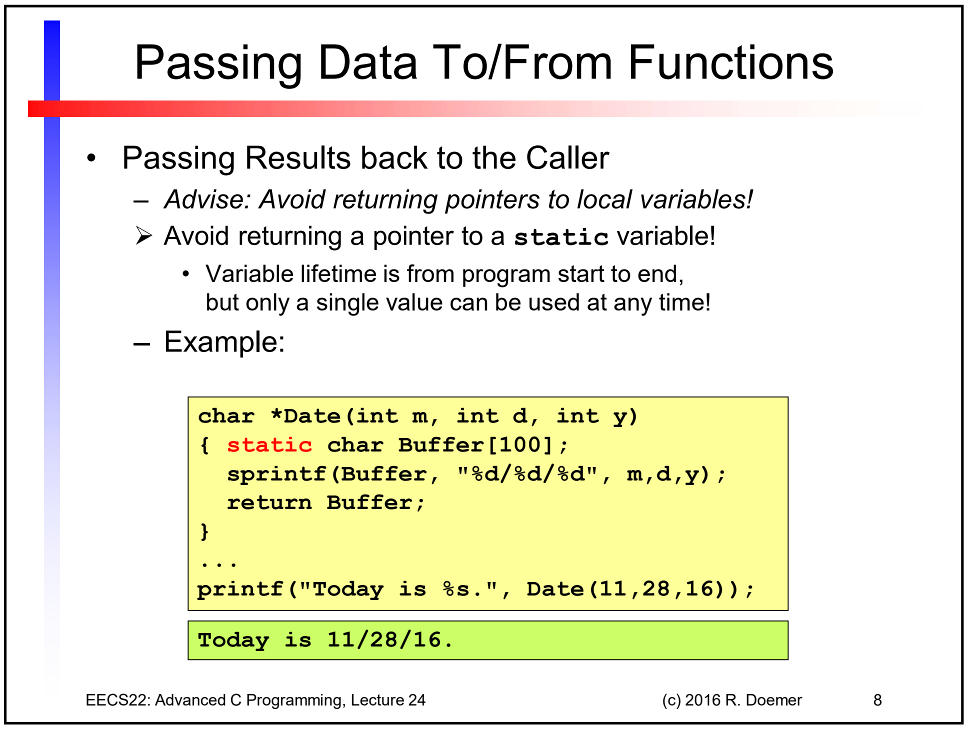


Passing Data To/From Functions

- Passing Results back to the Caller
  - *Advise: Avoid returning pointers to local variables!*
  - ➢ Never return a pointer to an **`auto`** variable!
    - The variable lifetime ends with the return from the function!
    - Any access to that pointer by the caller is undefined!
  - Example:

```
char *Date(int m, int d, int y)
{ char Buffer[100];
  sprintf(Buffer, "%d/%d/%d", m,d,y);
  return Buffer;
}
...
printf("Today is %s.", Date(11,28,16));
```

```
Today is #@#$%@#$@!...
```

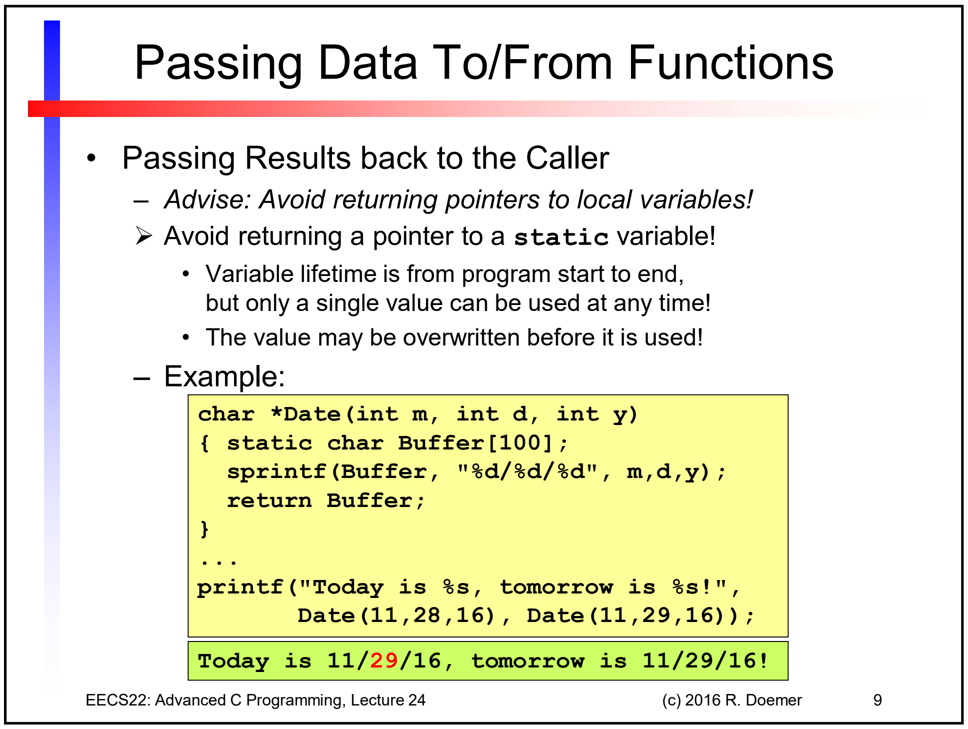


Passing Data To/From Functions

- Passing Results back to the Caller
  - *Advise: Avoid returning pointers to local variables!*
  - ➢ Avoid returning a pointer to a **`static`** variable!
    - Variable lifetime is from program start to end,
      but only a single value can be used at any time!
  - Example:

```
char *Date(int m, int d, int y)
{ static char Buffer[100];
  sprintf(Buffer, "%d/%d/%d", m,d,y);
  return Buffer;
}
...
printf("Today is %s.", Date(11,28,16));
```

```
Today is 11/28/16.
```

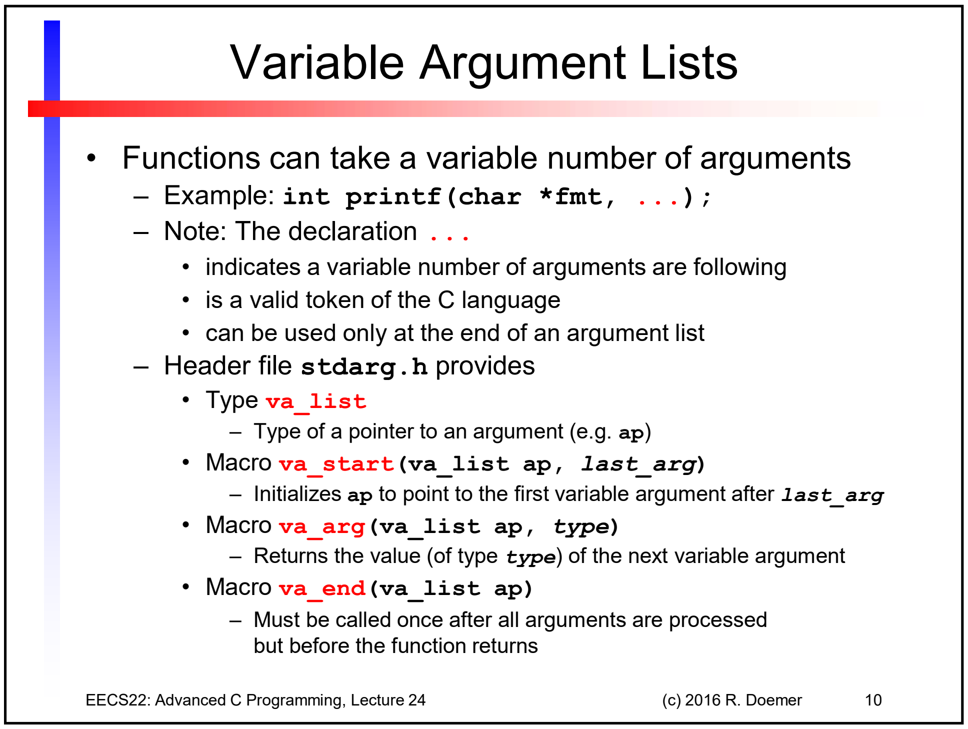


Passing Data To/From Functions

- Passing Results back to the Caller
  - *Advise: Avoid returning pointers to local variables!*
  - ➢ Avoid returning a pointer to a **`static`** variable!
    - Variable lifetime is from program start to end, but only a single value can be used at any time!
    - The value may be overwritten before it is used!
  - Example:

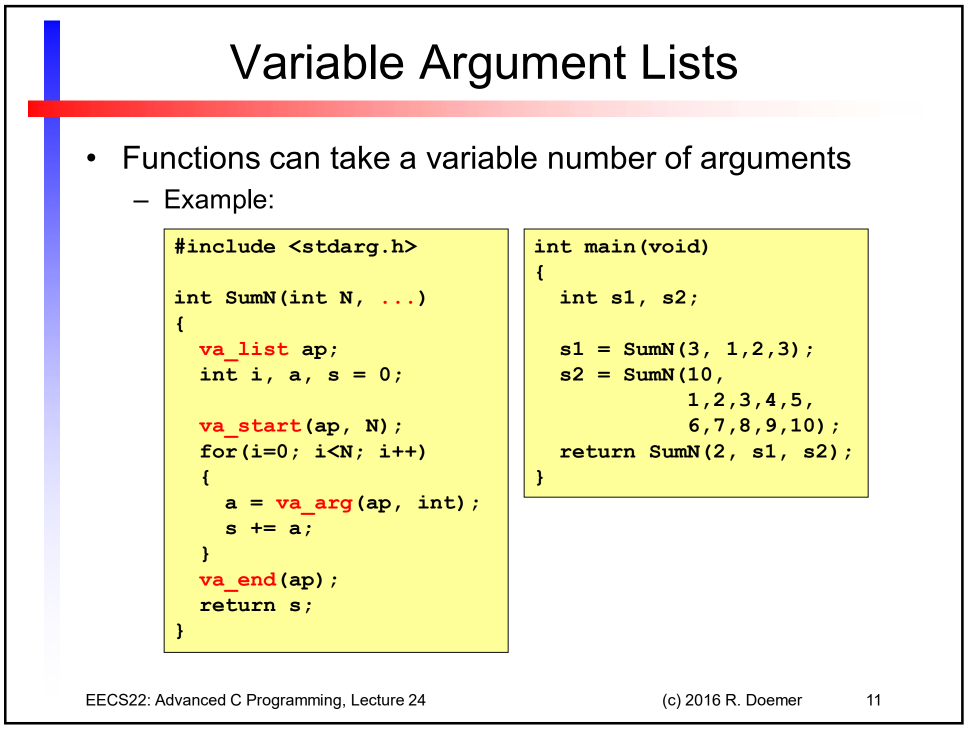
```
char *Date(int m, int d, int y)
{ static char Buffer[100];
  sprintf(Buffer, "%d/%d/%d", m,d,y);
  return Buffer;
}
...
printf("Today is %s, tomorrow is %s!",
       Date(11,28,16), Date(11,29,16));
```

```
Today is 11/29/16, tomorrow is 11/29/16!
```



Variable Argument Lists

- Functions can take a variable number of arguments
  - Example: **`int printf(char *fmt, ...);`**
  - Note: The declaration **`...`**
    - indicates a variable number of arguments are following
    - is a valid token of the C language
    - can be used only at the end of an argument list
  - Header file **`stdarg.h`** provides
    - Type **`va_list`**
      - Type of a pointer to an argument (e.g. **`ap`**)
    - Macro **`va_start(va_list ap, `*`last_arg`*`)`**
      - Initializes **`ap`** to point to the first variable argument after ***`last_arg`***
    - Macro **`va_arg(va_list ap, `*`type`*`)`**
      - Returns the value (of type ***`type`***) of the next variable argument
    - Macro **`va_end(va_list ap)`**
      - Must be called once after all arguments are processed but before the function returns



Variable Argument Lists

- Functions can take a variable number of arguments
  - Example:

```
#include <stdarg.h>

int SumN(int N, ...)
{
  va_list ap;
  int i, a, s = 0;

  va_start(ap, N);
  for(i=0; i<N; i++)
  {
    a = va_arg(ap, int);
    s += a;
  }
  va_end(ap);
  return s;
}
```

```
int main(void)
{
  int s1, s2;

  s1 = SumN(3, 1,2,3);
  s2 = SumN(10,
            1,2,3,4,5,
            6,7,8,9,10);
  return SumN(2, s1, s2);
}
```

EECS22: Advanced C Programming, Lecture 24 (c) 2016 R. Doemer 11