MODERN C++ DESIGN PATTERNS

C/C++ Functions

- So far, functions declared in our C/C++ programs have specified parameter lists
 - Fixed number of parameters
 - Every parameter has pre-specified type
- □ This is good
 - Compiler is able to use function declarations to detect deviations in numbers and type of arguments
- What about printf and scanf?

C-Style Variadic Functions

- printf and scanf are known as variadic functions
- How to declare such variable functions?
- How can such functions access extra arguments passed when they don't know how many arguments there are or what types these arguments have?

```
printf("Hello world\n");
printf("Hello %s\n", fnam);
printf("Your name is %s %s\n", fnam, lnam);
printf("%d + %d = %d\n", n1, n2, n1+n2);

scanf("%s", fnam);
scanf("%s%s", fnam, lnam);
scanf("%s%s %d/%d/%d", fnam, lnam, &m, &d, &y);
```

Declaring C-Style Variadic Functions

Provide one or more defined parameter
 declarations in parameter list followed by . . .

```
defined parameter declaration required!!!

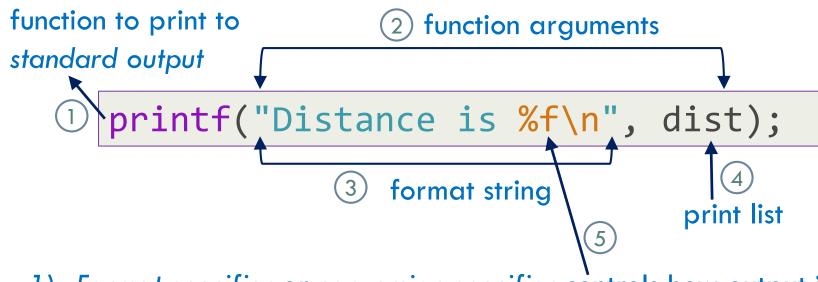
That is parameter with name and type associated with it
```

int printf(char const *format_string, ...);

Ellipsis means "and maybe some more parameters"

Declaration of printf says it has a defined parameter of type char const* followed by variable number of additional parameters

How Do printf and scanf Know?



- 1) Format specifier or conversion specifier controls how output is printed to standard output
- 2) Literal characters are printed as is
- 3) Character following % is abbreviation for type of data it represents and **must match** with corresponding argument
- 4) %f means print floating-point value using fixed-point notation

What Can Go Wrong?

- Variadic functions are flexible
- □ However, compiler is unable to check:
- Type safety of arguments being passed to function

```
printf("%s %s %d\n", age, lastname, firstname);
```

If number of arguments being passed matches semantics of function definition

```
printf("%s %s %d\n", firstname, lastname);
```

In both cases, we've <u>undefined behavior</u>

How Do printf and scanf Work?

- How does printf get access to unnamed, variable count of additional parameters following defined parameter?
- <cstdarg> provides a type and set of macro definitions that define how to step through variable parameter list

Type va_list

 Type va_list is name of opaque structure that will maintain information about entire variable parameter list

```
#include <cstdarg>

void foo(int defined_param, ...) {
   // argp will point to each unnamed
   // parameter in turn ...
   va_list argp;
   // other code follows
}
```

Macro va_start

 Makes variable argp of type va_list point to first unnamed parameter

```
void foo(int first_param, double second_param, ...) {
  va_list argp;
  // argp is pointing to first unnamed parameter
  va_start(argp, second_param);

  // other code follows
}
```

Macro va_arg

Each call of va_arg returns one parameter
 and steps argp to next unnamed parameter

```
void foo(int first_param, double second_param, ...) {
  va list argp;
 // argp is pointing to first unnamed parameter
  va_start(argp, second_param);
 // assuming first two unnamed parameters have
 // int type followed by double type
  int ival = va_arg(argp, int);
  double dval = va_arg(argp, double);
 // other code follows
```

Macro va_arg: Default Promotions

- Compiler performs default promotions on all parameters that match ellipsis
 - char and short arguments promoted to int
 - float values promoted to double
 - Therefore, doesn't make sense to pass types such as char, short, and float

Macro va_end

 Must be called to reset global variables and perform cleanup

```
void foo(int first param, double second param, ...) {
 va list argp;
 // argp is pointing to first unnamed parameter
 va_start(argp, second_param);
 // assuming first two unnamed parameters have
 // int type followed by double type
  int ival = va_arg(argp, int);
  double dval = va arg(argp, double);
 // extract other unnamed parameters
 va_end(argp);
```