

MODERN C++ DESIGN PATTERNS

C-Style Variadic Functions

by Prasanna Ghali

C/C++ Functions

2

- 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

3

- `printf` and `scanf` are known as *variadic* functions
- How to declare such variadic 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

4

- 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?

5

function to print to
standard output

② function arguments

① `printf("Distance is %f\n", dist);`

③ format string

④ print list

⑤

- 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?

6

- ❑ 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 undefined behavior

How Do `printf` and `scanf` Work?

7

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

Type `va_list`

8

- 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`

9

- 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`

10

- 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

11

- 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`

12

- ❑ 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);  
}
```