Notes 16.0: Variadic functions

COMP9021 Principles of Programming

School of Computer Science and Engineering
The University of New South Wales

2013 session 1

The stdarg.h header file

Functions like `printf()` and `scanf()` take a fixed argument, of type `char *`, and a variable number of arguments. Their prototypes are

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

and

```
int scanf(const char *, ...);
```

They are examples of variadic functions, that take at least one fixed argument, and a variable number of arguments indicated in the function prototype and definition by an ellipsis.

The `stdarg.h` header file provides the `va_list` type as well as the `va_start()`, `va_arg()`, `va_end()` and `va_copy()` macros to define such functions.

Defining variadic functions (1)

Consider a variadic function `f()` with arguments declared as

```
type_1 fixed_arg_1, ..., type_n fixed_arg_n, ...
```

with n at least equal to 1 (the first ellipsis is a meta-notation, whereas the second ellipsis is literal!).

The function definition must first define a variable, say `ap`, of type `va_list`:

```
va_list ap;
```

Then the function `va_start()` must be called with `ap` as first argument and the name of `f()`'s last fixed argument as second argument:

```
va_start(ap, fixed_arg_n);
```

Defining variadic functions (2)

Every variable argument will be accessed in turn with a call to the function `va_arg()`, of the form `va_arg(ap, arg_type)` where *`arg_type`* is the type of the argument, that must therefore been "known" in one way or another. For instance, if the next argument to retrieve is known to be of type `double` then a typical statement would be

```
double x = va_arg(ap, double);
```

Finally, when all (needed) variable arguments will have been retrieved and processed, which has to be "known" in one way or another, the function `va_end()` must be called with `ap` as unique argument:

```
va_end(ap);
```

The number of types of variable arguments

Note that when `printf()` or `scanf()` is called, the actual number of arguments passed to the function, as well as their type, can be determined by analysing the fixed argument (the format string).

More generally, when a variadic function is called, it has to be possible to find out the number of (needed) variable arguments and their types. For instance, the number of variable arguments can

- be given as the value of a nonvariable argument, or
- be computed from some nonvariable arguments, or
- be indicated by the last variable argument taking a special value, that plays the role of a flag.

Copying variable arguments

At any point, the remaining sequence of variable arguments (that have not yet been retrieved by a call to `va_arg()`) can be duplicated by a call to the function `va_copy()` of the form `va_copy(what_remain_of_ap, ap)` where `what_remain_of_ap` is a declared variable of type `va_list`.

Eventually, the function `va_end()` must be called with `what_remain_of_ap` provided as argument.

The type, functions and constructions previously described are illustrated in stdarg.c

A note on the second argument of va_arg()

As values of type `signed char` or `short` are automatically promoted to `int`, values of type `unsigned char` or `unsigned short` are automatically promoted to `unsigned`, values of type `char` are automatically promoted to `int` or `unsigned`, and values of type `float` sometimes automatically promoted to `double`, `va_arg()` might not accept `signed char`, `short`, `unsigned char`, `unsigned short`, `char` or `float` as second argument.