C Basics 14: Variadic Functions

1. Overview of C and C vs. Java
2. C Program Elements
3. The Preprocessor
4. Identifiers and Data Types
5. Pointers and Arrays
6. Strings
7. Library I/O
8. Errors and Error Handling
9. Functions and Parameter Passing
10. Multidimensional Arrays

C Basics 14: Variadic Functions

11. Dynamic and Static Memory
12. Structs (Structures)
13. Higher-Order Functions
14. **Variadic Functions**
    - **variadic functions**
    - **stdarg.h and its macros**
    - **examples**
    - **variadic preprocessor macros**
15. Miscellaneous

Variadic Functions

Variadic functions are functions that can accept a *variable number of arguments*.

They are also simply called functions with *variable argument lists*.

Another way of saying this is that they are functions of *indefinite arity*.

A key example in C is `printf()`, which takes a single format string, but then an *arbitrary* number of value arguments:

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

Lisp has a wide range of variadic functions, such as the arithmetic operators: `(+ x y z w)`.

C allows users to define variadic functions and macros.

stdarg.h

To implement variadic functions in C, the header file `stdarg.h` must be included.

stdarg.h declares the type `va_list` and four macros:

- `void va_start(va_list ap, last)`
- `type va_arg(va_list ap, type)`
- `void va_end(va_list ap)`
- `void va_copy(va_list dest, va_list src)` (C99 addition)

See "`man stdarg`" for further info.

A key limitation is that C variadic functions with *no fixed arguments* are *not possible*.

stdarg.h (contd.)

`void va_start(va_list ap, last)`:

- initializes `ap` for subsequent use by `va_arg()` and `va_end()`
- must be *called first*
- `last` is the name of the last argument before the variable argument list
- last argument the calling function knows the type of

`type va_arg(va_list ap, type)`:

- expands to an expression that has the type and value of the next argument in the call
- when first called after `va_start()`, it returns the argument immediately after `last`
- each subsequent call returns the next argument

stdarg.h (contd.)

`void va_end(va_list ap)`:

- each `va_start()` invocation must be matched by a corresponding `va_end()` invocation in the same function
- multiple argument list traversals are allowed, each must be bracketed by `va_start()` and `va_end()`

`void va_copy(va_list dest, va_list src)`:

- copies the (previously initialized) variable argument list `src` to `dest`
- each `va_copy()` invocation must be matched by a corresponding `va_end()` invocation in the same function

Example Variadic Function

Example C variadic function from Wikipedia:

```
double average(int count, ...)
{
  va_list ap;
  int j;
  double sum = 0;

  va_start(ap, count);
  for (j = 0; j < count; j++) {
      sum += va_arg(ap, int);
  }
  va_end(ap);

  return sum / count;
}

int main(int argc, char const *argv[])
{
  printf("%f\n", average(3, 1, 2, 3) );
  return EXIT_SUCCESS;
}
```

Example Variadic Function

Example C variadic function from `stdarg` man page:

```
void simple_printf(char *fmt, ...)
{
  va_list ap;
  int d;
  char c, *s;

  va_start(ap, fmt);
  while (*fmt)
    switch (*fmt++) {
      case 's':  /* string */
        s = va_arg(ap, char *);
        printf("string %s\n", s);
        break;
      case 'd':  /* int */
        d = va_arg(ap, int);
        printf("int %d\n", d);
        break;
      case 'c':  /* char */
        /* cast needed since va_arg takes only fully promoted types */
        c = (char) va_arg(ap, int);
        printf("char %c\n", c);
        break;
      }
  va_end(ap);
}
```

Variadic Preprocessor Macros

C99 added the ability to define **variadic preprocessor macrors**.

Example (from GCC manual):

```
#define debug(format, ...) fprintf(stderr, format, __VA_ARGS__)
```

GCC allows a name to be given to the variable arguments:

```
#define debug(format, args...) fprintf(stderr, format, args)
```

The `debug` macro could be called like:

```
debug("The value of x is: %d\n",x);
```

Variadic Preprocessor Macros (contd.)

The C standard does not allow the variable argument to be left out entirely (you are allowed to pass an empty argument).

E.g., the following invocation is invalid in ISO C, because there is no comma after the string:

```
debug("A message")
```

It will lead GCC will complain since the expansion of the macro will contain an extra comma after the format string:

```
fprintf(stderr, "A message", )
```

GCC allows the `##` operator to be used to remove the comma before it if the variable arguments are omitted or empty:

```
#define debug(format, ...) fprintf(stderr, format, ## __VA_ARGS__)
```