When static is just not good enough…

More Advanced C

Variable Argument Lists

Variable Argument Lists

- Variable argument lists are used to create functions that allow a variable number of parameters.
- `printf()` is the classic example:

```
printf ( “%d %s %d “, a, b, c );
```

Format statement

```
printf ( “%d %d “, a, c );
```

Format statement

`printf()`'s Format Statement

- The format statement indicates how many and of what type of the parameters that follow.
- *E.g.* "`%d %f`" is an integer followed by a float

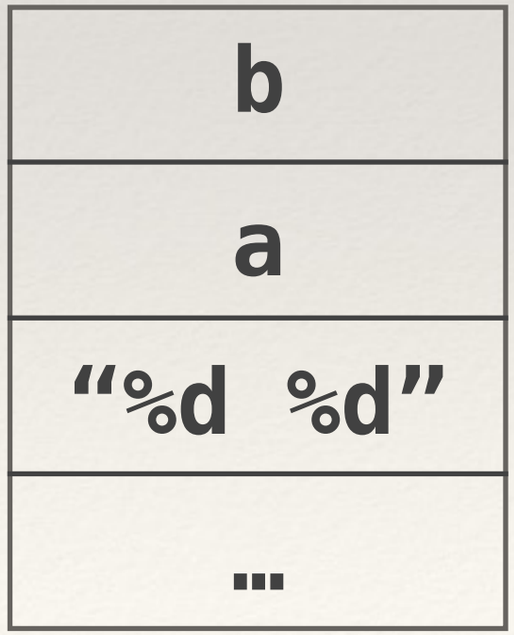
  "`%d %d %d`" are three integers

The Stack

- Parameters to a function are stored on the stack when the function is called.
- The function call,

```
printf ( “%d %d”, a, b );
```

- will create the stack



The Stack

- Inside the function call we can take the format string off the stack (`“%d %d”`).
- The format string parameter tells us that:
  - there are two more parameters on the stack
  - those parameters are both integers.

```
#include <stdarg.h>

void va_start ( va_list ap, last );

type va_arg ( va_list ap, type );

void va_end ( va_list ap );
```

A function may be called with a varying number of arguments of varying types.

The include file `<stdarg.h>` declares a type (`va_list`) and defines three macros for stepping through a list of arguments whose number and types are not known to the called function.

The called function must declare an object of type `va_list` which is used by the macros `va_start(), va_arg(), and va_end().`

```
void va_start ( va_list ap, last );
```

The `va_start()` macro must be called first, and it initializes `ap`, which can be passed to `va_arg()` for each argument to be processed.

```
void va_end ( va_list ap );
```

Calling `va_end()` signals that there are no further arguments, and causes `ap` to be invalidated. Note that each call to `va_start()` must be matched by a call to `va_end()`, from within the same function.

The parameter `last` is the name of the last parameter before the variable argument list, *i.e.*, the last parameter of which the calling function knows the type.

```
type va_arg ( va_list ap, type );
```

The `va_arg()` macro expands to an expression that has the type and value of the next argument in the call. The parameter `ap` is the `va_list ap` initialized by `va_start()`. Each call to `va_arg()` modifies `ap` so that the next call returns the next argument.

The parameter `type` is a type name specified so that the type of a pointer to an object that has the specified type can be obtained simply by adding a * to type.

If there is no next argument, or if type is not compatible with the type of the actual next argument, random errors will occur.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>
double add ( char *format, ... )
{
   va_list args;
   int ia;
   double fa;
   double total = 0.0;
   char parm[10];

   va_start ( args, format );
   while ( sscanf ( format, "%s", parm ) == 1 ) {
      parm[strlen(parm)] = '\0';
      if ( strncmp ( parm, "integer", 7 ) == 0 ) {
         ia = va_arg ( args, int );
         total = total + (double)ia;
      } else if ( strncmp ( format, "float", 5 ) == 0 ) {
         fa = va_arg ( args, double );
         total = total + fa;
      }
      format = format + strlen(parm) + 1 ;
   }
   va_end ( args );
   return ( total );
}
```

Initialize the va_list variable args

Take one parameter off the list and convert to the appropriate type

Finished using the argument list

```c
#include <stdio.h>

double add ( char *fmt, ... );

int main ( int argc, char *argv[] )
{
   int ia, ib, isum;
   float fa, fb, fc;
   double dsum;

   ia = 3;
   ib = 5;
   printf ( "%d + %d = ", ia, ib );
   isum = (int) add ( "integer integer", ia, ib );
   printf ( "%d\n", isum );

   printf ( "%d + %d + %d + %d = %d\n", ia, ib, ia, ib,
      (int)add("integer integer integer integer”,
         ia,ib,ia,ib) );
```

```
    fa = 1.2;
    fb = 3.456;
    fc = 4.544;
    printf ( "%8.3f + %8.3f = ", fa, fb );
    dsum = add ( "float float", fa, fb );
    printf ( "%8.3f\n", dsum );

    printf ( "%8.3f + %8.3f + %8d = %8.3f\n", fa, fb, ia,
       add ("float float integer",fa,fb,ia) );

    printf ( "%8.3f + %8.3f + %8.3f = %8.3f\n", fa, fb, fc,
       add ("float float float",fa,fb,fc) );

    return (0);

}
```

A (int) add ("integer integer", ia, ib);

B (int)add("integer integer integer integer", ia,ib,ia,ib)

C add ("float float", fa, fb);

D add ("float float integer",fa,fb,ia));

E add ("float float float",fa,fb,fc));

```
$ ./vaLists
A   3 + 5 = 8
B   3 + 5 + 3 + 5 = 16
C     1.200 +   3.456 =   4.656
D     1.200 +   3.456 +       3 =   7.656
E     1.200 +   3.456 +   4.544 =   9.200
```

https://www.tutorialspoint.com/cprogramming/c_variable_arguments.htm

```c
#include <stdio.h>
#include <stdarg.h>

double average(int num,...) {
   va_list numList;
   double sum = 0.0;
   int i;

   va_start(numList, num);

   for (i = 0; i < num; i++) {
      sum += va_arg(numList, int);
   }

   va_end(numList);

   return ( sum / (double)num );
}
```

In this example, the first argument is an integer, not a string. It is very simple to read all of the arguments since we just have to make repeated calls to the macro `va_arg`.

```
double average ( int num, ... );
…
printf("Average of 25, 3, 54, 105 = %8.3f\n",
average(4, 25, 3, 54, 105));
```

```
Average of 25, 3, 54, 105 =   46.750
```

Command Line Arguments

- Command line arguments allow you to write code that does not need to hardcode values - you can change or supply values to variables via the command line.
- `main()` has two function arguments
  - `argc` - number of arguments passed
  - `argv[]` - pointer array that references each of the command line arguments

```
#include <stdio.h>

int main ( int argc, char *argv[] )
{
   int i;

   printf ( "argc = %d\n",argc );

   printf ( "The program name is %s\n", argv[0] );

   printf ( "This program has %d arguments\n”,argc-1);

   printf ( "The arguments are: \n");
   for ( i=1; i<argc; i++ ) {
      printf ( "%s\n",argv[i] );
   }
}
```

```
$ ./command 1  5.25  this  is  a  test  "this is a test"
The program name is ./command
argc = 8
This program has 7 arguments
The arguments are:
1
5.25
this
is
a
test
this is a test
```