Lesson 27

Objectives: -

- Variable Length Argument Lists
- Prime Numbers
- Fibonacci Numbers

Variable Length Argument Lists

Function

This program demonstrates how to declare and use functions in which the number and type of arguments passed to it are not known to the function itself.

Programming Issues

Functions which can take a variable number of arguments must be declared:

```
(type) func( (type) arg, ... );
```

At least one of the arguments must be known (i.e. `int func(...)` is illegal), and must all be placed before the ellipsis (the "...").

A variable of type `va_list` must be declared.

Accessing the arguments involves three main steps:

1. Calling `va_start(va_list argptr, last_arg)` where `argptr` is the variable declared earlier, and `last_arg` is the name of the last known argument before the ellipsis.
2. Looping through the number of arguments, retrieving each one in turn with `va_arg(va_list argptr, type)` where `type` is the type of the argument, such as `int` or `char *`. There are a number of ways to determine the number and type of arguments. In the function `sum()`, each argument is always of type `int`, and the argument `nArgs` supplies the number of arguments to the function. In the function `PrintArgs()`, the length of the format string passed to it gives the number of arguments, and the type of each argument is determined by examining the characters in the string. This is similar to how `printf()` works.
3. Calling `va_end(va_list argptr)` to clean up nicely. This step must be carried out.

Usage

Execute the program as normal.

4.vararg.c

```
/*

  VARARG.C
  ========
  (c) Paul Griffiths 1999
  Email: mail@paulgriffiths.net

  Example of using variable length argument lists

*/


#include <stdlib.h>
#include <stdio.h>
```

```
#include <stdarg.h>


/*  Forward function declarations  */

int  sum      (int nArgs, ...);
void PrintArgs(char *format, ...);


/*  main() function  */

int main() {
    int a = 5, b = 7, c = 11, d = 13, e = 18,
result;

    double db     = 3.45;
    char   ch     = 'x';
    char  *st1    = "Random string";
    char  *st2    = "Another random string";


    /*  Demonstrate usage of sum() function  */

    result = sum(3, a, b, c);
    printf("%d + %d + %d = %d\n", a, b, c,
result);

    result = sum(4, a, b, d, e);
    printf("%d + %d + %d + %d = %d\n", a, b, d,
e, result);

    result = sum(5, a, b, c, d, e);
    printf("%d + %d + %d + %d + %d = %d\n\n", a,
b, c, d, e, result);


    /*  Demonstrate usage of PrintArgs() function
*/

    PrintArgs("csds", ch, st1, db, st2);

    return EXIT_SUCCESS;
}


/*  Function returns the sum of a variable number
of arguments
    of type int. 'nArgs' should contain the
number of arguments  */

int sum(int nArgs, ...) {
    int     result = 0;
    va_list argptr;


    /*  Initialize argument pointer  */
```

```
    va_start(argptr, nArgs);


    /*  Loop through arguments and add to result
*/

    while ( nArgs-- )
      result += va_arg(argptr, int);


    /*  Clean up nicely and return result  */

    va_end(argptr);
    return result;
}


/*  Outputs the type and contents of a variable
number of arguments
    with different types. 'format' should be a
string containing one
    character for each argument. For example, if
'format' points to
    the string "csds", it signifies that there
are four arguments, and
    that:
        'c' : the first argument is a character
      's' : the second argument is a string
      'd' : the third argument is a double
      's' : the fourth argument is a string
    No other argument types other than the three
above are supported
    by this function.
*/

void PrintArgs(char *format, ...) {
    int     n = 0;
    va_list argptr;


    /*  Initialize argument pointer  */

    va_start(argptr, format);


    /*  Determine and output arguments  */

    while ( format[n] ) {
      switch ( format[n] ) {
      case 'c':
          printf("Argument %d is a char, and is
set to: %c\n",
                 n + 1, va_arg(argptr, char));
          break;

      case 's':
```

```
          printf("Argument %d is a string, and is
set to: %s\n",
                 n + 1, va_arg(argptr, char*));
          break;

      case 'd':
          printf("Argument %d is a double, and is
set to: %f\n",
                 n + 1, va_arg(argptr, double));
          break;

      default:
          printf("Unsupported argument.\n");
          exit(EXIT_FAILURE);
      }
      ++n;
    }


    /*  Clean up nicely and exit  */

    va_end(argptr);
}
```

Prime Numbers

Function

This program calculates and displays the first 1,000 prime numbers (including 1, 2 and 3).

Programming Issues

To test if a number *n* is prime, we could loop through 2 to *n - 1* and test whether each number divides exactly into n (`(n % test) == 0`). If any of them do, the number is not prime.

However, since by definition any number which is not prime can be divided by at least one other prime number, a more efficient way to do it is to test only the prime numbers less than *n*. Therefore, our program maintains a list of prime numbers already found, and uses that to test *n*.

Usage

Just type `./prime` to run the program

5.prime.c

```
/*
  PRIME.C
  =======
  (c) Copyright Paul Griffiths 2000
  Email: mail@paulgriffiths.net

  A program to find the first 1,000 prime numbers
*/


#include <stdio.h>
#include <stdlib.h>


int prime(int number, int * primes);

int main(void) {
    int primes[998] = { 3, 0 };
    int n = 5, i;
    int count = 0, found;

    printf("%8d%8d%8d", 1, 2, 3);
/*  Print first 3 primes     */


    /*  Find the next 997  */

    while ( count < 997 ) {
        i = 0;
        found = 1;


        /*  Test if number divides by any of the
primes already found  */

        while ( primes[i] ) {
            if ( (n % primes[i++]) == 0 ) {   /*
If it does...              */
                found = 0;                    /*
...then it isn't prime...  */
                break;                        /*
...so stop looking         */
            }
        }

        if ( found ) {
            printf("%8d", n);                /*
If it's prime, print it...  */
            primes[i] =  n;                  /*
...and add it to the list   */
```

```
            ++count;


            /*  Start a new line every 8 primes
found  */

            if ( ((count + 3) % 8) == 0 )
                putchar('\n');
        }

        n += 2;     /*  There's no point testing
even numbers, so skip them  */
    }

    putchar('\n');

    return EXIT_SUCCESS;
}
```

Fibonacci Numbers

Function

This program prints out the first *n* numbers in the Fibonacci sequence. In case you are unfamiliar with the Fibonacci sequence, the first two numbers are both 1, and from then on each following number is equal to the sum of the previous two.

To demonstrate:

Computation	Fibonacci number
1	1
1	1
1 + 1	2
1 + 2	3
2 + 3	5
3 + 5	8
5 + 8	13
8 + 13	21

13 + 21 34

Programming Issues

The computations are done using a simple `for` loop, using two variables (`f1` and `f2`) to hold each pair, and a temporary variable used when updating them. The Fibonacci sequence can be computed recursively, but this is a much less efficient way in this case.

The number of numbers to calculate is passed via the command line, and extracted by `ParseCmdLine()`. The number must be greater than 2 (because the first two numbers are always output) and less than 48 (because `unsigned long` is not guaranteed to hold numbers larger than 4,294,967,295).

Usage

Pass the desired number as the sole command line argument. A sample session is shown below:

```
[paul@hermes fibo]$ ./fibo 47
First 47 numbers in the Fibonacci sequence:

          1          1          2          3
5
          8         13         21         34
55
         89        144        233        377
610
        987       1597       2584       4181
6765
      10946      17711      28657      46368
75025
     121393     196418     317811     514229
832040
    1346269    2178309    3524578    5702887
9227465
   14930352   24157817   39088169   63245986
102334155
  165580141  267914296  433494437  701408733
1134903170
 1836311903 2971215073
```

6.fibo.c

```
/*
  FIBO.C
  ======
  (c) Copyright Paul Griffiths 2000
  Email: mail@paulgriffiths.net

  Finds the first 'n' numbers in the Fibonacci
sequence.
  'n' must be supplied as the sole command line
argument,
  and must be between 3 and 47.
*/


#include <stdio.h>
#include <stdlib.h>


int ParseCmdLine(int argc, char *argv[]);

int main(int argc, char * argv[]) {
    unsigned long f1 = 1, f2 = 1;
    int temp, i = 3;
    int n = ParseCmdLine(argc, argv);

    printf("First %d numbers in the Fibonacci
sequence:\n\n", n);

    printf("%11lu%11lu", f1, f2);

    for ( i = 3; i <= n; ++i ) {
        temp = f2;
        f2 += f1;
        f1 = temp;

        printf("%11lu", f2);

        if ( i % 5 == 0 )
            putchar('\n');
    }
    putchar('\n');

    return EXIT_SUCCESS;
}


/*  Returns the integer specified on the command
line  */

int ParseCmdLine(int argc, char *argv[]) {
    int n;
    char * endptr;

    if ( argc < 2 ) {
        fprintf(stderr, "You must supply an
argument\n");
        exit(EXIT_FAILURE);
```

```
    }
    else if ( argc > 2 ) {
        fprintf(stderr, "You must only supply one
argument\n");
        exit(EXIT_FAILURE);
    }

    n = strtol(argv[1], &endptr, 0);
    if ( *endptr ) {
        fprintf(stderr, "You must supply a whole
number as an argument\n");
        exit(EXIT_FAILURE);
    }

    if ( n < 3 ) {
        fprintf(stderr, "You must supply a number
greater than 2\n");
        exit(EXIT_FAILURE);
    }
    else if ( n > 47 ) {
        fprintf(stderr, "You must supply a number
less than 48\n");
        exit(EXIT_FAILURE);
    }

    return n;
}
```

Questions: -

1. Try above program and check o/p