Programming and Algorithms

COMP1038.PGA

Session 21: Variable length argument

Dr. Pushpendu Kar

Introduction

- Almost all functions we have seen so far in C take a fixed number of arguments
- But *printf* can take a variable number of arguments.
- How is it implemented and how useful is that technique?

Syntax

- A function may be declared to take a variable number of arguments, instead of a fixed number of arguments.
- Eg, you can give *printf* as many values as you want to print.
- The function must have at least 1 formal argument.
- The var-args must be after all formal arguments.
- Variable argument function is declared with ellipsis:

```
Return_type name_of_the_function(int argument_count, ...)
{
// function definition
}
```

ellipsis

Process to use Var-arg

- You can access the variable arguments by using macros in the *stdarg.h* header.
- Declare a variable of type *va_list* to store the current state of the var-args.
- Initialize this with *va_start*.
- Call *va_arg* as many times as you need to get each *var-arg* value, specifying what type you want to treat the value as.
- Clean up after the *var-args* by calling *va_end*.

Initializing Var-args

- Define a function with its last parameter as ellipses and the one just before the ellipses is always an **int** which will represent the number of arguments.
- Declare a variable of type va_list.
- Use **int** parameter and **va_start** macro to initialize the **va_list** variable to an argument list. The macro va_start is defined in stdarg.h header file.

```
int average(int count, ...)
{
// ...
va_list ap;
va_start(ap, count);
// ...
}
```

The University of Nottingham
UNITED KINGDOM · CHINA · MALAYSIA

Getting Var-args values

- Use **va_arg** macro and **va_list** variable to access each item in argument list.
- Wrong type may return you nonsense values or cause a segmentation fault.
- Trying to access more variables than were passed can cause a segmentation fault.

```
int average(int count, ...)
{
// ...
int x = va_arg(ap, int);
// ...
}
```

Clean up Var-args

- Call va_end with the va_list variable.
- Must do this before you exit the function.
- Cannot use va_arg after calling va_end.

```
int average(int count, ...)
{
// ...
va_end(ap);
// ...
}
```

Use of Var-args

- Useful when programmer knows at compile time how many variables and which variables to pass to a function, and this changes each time they use the function.
- If you have a variable amount of things, most of the time you only know what and how much at run time.
- var-args need you to write each variable name in the source code function call.
- If you don't know this at compile time, you can't use var-args.

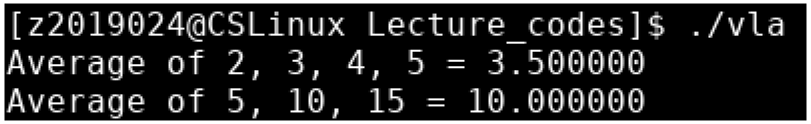
Example program

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

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

  /* initialize valist for num number of
arguments */
  va_start(valist, num);

  /* access all the arguments assigned to
valist */
  for (i = 0; i < num; i++) {
    sum += va_arg(valist, int);
  }
/* clean memory reserved for valist */
  va_end(valist);
  return sum/num;
}
int main() {
  printf("Average of 2, 3, 4, 5 = %f\n",
average(4, 2,3,4,5));
  printf("Average of 5, 10, 15 = %f\n",
average(3, 5,10,15));
}
```



```
[z2019024@CSLinux Lecture_codes]$ ./vla
Average of 2, 3, 4, 5 = 3.500000
Average of 5, 10, 15 = 10.000000
```

The University of Nottingham
UNITED KINGDOM · CHINA · MALAYSIA

The End