ALX LESSON

0x10 C - Variadic functions

C - Programming

TABLE OF CONTENTS

01

OVERVIEW topics

Topics

C Programming

Topics

What are variadic functions

How to use va_start, va_arg and va_end macros

Why and how to use the const type qualifier

Slides On Telegram

https://t.me/alx_2023

C Programming

Topics

02

Learning Objectives

What are variadic functions

Variadic functions are functions that can take a variable number of arguments. In C programming, a variadic function adds flexibility to the program. It takes one fixed argument and then any number of arguments can be passed. The variadic function consists of at least one fixed variable and then an ellipsis(...) as the last parameter.

Syntax:

```
int function_name(data_type variable_name, ...);
```

Values of the passed arguments can be accessed through the header file named as:

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

How to use va_start, va_arg and va_end macros

<stdarg.h> includes the following methods:

va_start(va_list ap, argN)
va_arg(va_list ap, type)
va_copy(va_list dest, va_list src)
va_end(va_list ap)

Facility	Use
va_start	start access to variadic arguments
va_arg	access the next variadic argument
va_copy	makes a copy of the variadic arguments
va_end	end traversal of variadic arguments
va_list	holds information needed by other facilities

How to use va_start, va_arg and va_end macros

va_start(va_list ap, argN)

This enables access to variadic function arguments.

where *va_list* will be the pointer to the last fixed argument in the variadic function

argN is the last fixed argument in the variadic function.

From the above variadic function (function_name (data_type variable_name, ...);), variable_name is the last fixed argument making it the argN. Whereas *va_list ap* will be a pointer to argN (variable_name)

How to use va_start, va_arg and va_end macros

va_arg(va_list ap, type)

This one accesses the next variadic function argument.

va_list ap is the same as above i.e a pointer to argN

type indicates the data type the *va_list ap* should expect (double, float, int etc.)

How to use va_start, va_arg and va_end macros

va_copy(va_list dest, va_list src)
This makes a copy of the variadic function arguments.

va_end(va_list ap)
This ends the traversal of the variadic function arguments.

Note:
va_list holds the information needed by va_start, va_arg, va_end, and va_copy.

e1

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

int variadic_addition (int count,...)
{
  va_list args;
  int i, sum;

  va_start (args, count);         /* Save arguments in list. */

  sum = 0;
  for (i = 0; i < count; i++)
    sum += va_arg (args, int);    /* Get the next argument value. */

  va_end (args);                  /* Stop traversal. */
  return sum;
}


int main(){

  // call 1: 4 arguments
  printf("Sum: %d\n", variadic_addition(3, 10, 20, 30));

  //call 2: 6 arguments
  printf("Sum: %d\n", variadic_addition(5, 10, 20, 30, 40, 50));

  return 0;

}
```

e2

```c
// C program for the above approach

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

// Variadic function to add numbers
int AddNumbers(int n, ...)
{
                int Sum = 0;

                // Declaring pointer to the
                // argument list
                va_list ptr;

                // Initializing argument to the
                // list pointer
                va_start(ptr, n);

                for (int i = 0; i < n; i++)

                                // Accessing current variable
                                // and pointing to next one
                                Sum += va_arg(ptr, int);

                // Ending argument list traversal
                va_end(ptr);

                return Sum;
}

// Driver Code
int main()
{
                printf("\n\n Variadic functions: \n");

                // Variable number of arguments
                printf("\n 1 + 2 = %d ",
                                AddNumbers(2, 1, 2));

                printf("\n 3 + 4 + 5 = %d ",
                                AddNumbers(3, 3, 4, 5));

                printf("\n 6 + 7 + 8 + 9 = %d ",
                                AddNumbers(4, 6, 7, 8, 9));

                printf("\n");

                return 0;
}
```

e3

```c
// C program for the above approach
#include <stdarg.h>
#include <stdio.h>

// Variadic function to find the largest number
int LargestNumber(int n, ...)
{
                        // Declaring pointer to the
                        // argument list
                        va_list ptr;

                        // Initializing argument to the
                        // list pointer
                        va_start(ptr, n);

                        int max = va_arg(ptr, int);

                        for (int i = 0; i < n-1; i++) {

                                                // Accessing current variable
                                                // and pointing to next
                                                int temp = va_arg(ptr, int);
                                                max = temp > max ? temp : max;
                        }

                        // End of argument list traversal
                        va_end(ptr);

                        return max;
}

// Driver Code
int main()
{
                        printf("\n\n Variadic functions: \n");

                        // Variable number of arguments
                        printf("\n %d ",
                                                LargestNumber(2, 1, 2));

                        printf("\n %d ",
                                                LargestNumber(3, 3, 4, 5));

                        printf("\n %d ",
                                                LargestNumber(4, 6, 7, 8, 9));

                        printf("\n");

                        return 0;
}
```

Why and how to use the const type qualifier

The const qualifier is useful in several situations:

To enforce read-only access: If you have a variable or pointer that should not be modified during the program's execution, declaring it as const can help ensure that it is not accidentally modified by the program.

To improve code quality and maintainability: Declaring a variable or pointer as const can make it easier for others to understand your code because they know that the value of the variable or pointer will not be changed. This can also help prevent bugs from being introduced into the codebase.

To allow the compiler to optimize code: If the compiler knows that a variable or pointer is const, it can make certain optimizations that would not be possible otherwise. For example, the compiler can perform constant folding, where it replaces expressions that involve constants with their computed values.

Why and how to use the const type qualifier

```
const int *p = &x;
```

```
int* const p = &x;
```

What is the difference!

Why and how to use the const type qualifier

```
const int *p = &x;
```

This declares a pointer p to a constant integer value. This means that the integer value pointed to by p cannot be modified through p. However, p itself can be modified to point to a different integer value. For example, the following code would be valid:

```
const int x = 10;
const int y = 20;
const int *p = &x;
p = &y; // OK
*P = 55; //Not Valid
```

Why and how to use the const type qualifier

int* const p = &x;

This declares a constant pointer p to an integer value. This means that p itself cannot be modified to point to a different integer value, but the integer value it points to can be modified. For example, the following code would be valid:

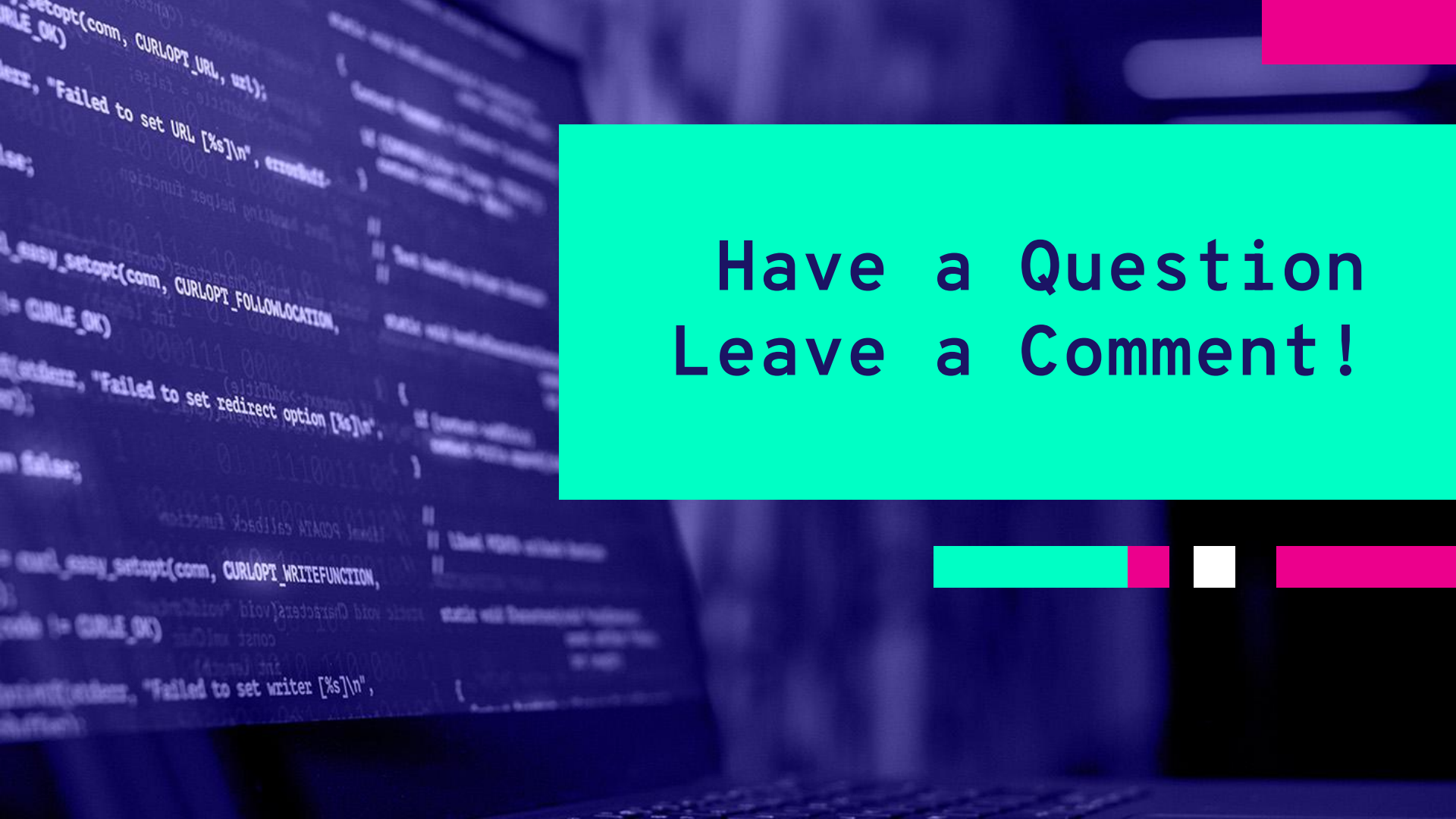
```
int x = 10;
int y = 20;
int* const p = &x;
*p = 30; // OK
p = &y; // NOT VALID
```

e4

```c
#include <stdio.h>

int main() {
    const int x = 10;
                const int y = 30;
    const int *p = &x;
    int* const q = &x;

    printf("x = %d\n", x);
    printf("*p = %d\n", *p);
    printf("*q = %d\n", *q);

    // This would be valid:
    p = &y;
    printf("*p = %d\n", *p);

    // This would not be valid:
    // *p = 20; // ERROR: Cannot modify a const value through a const pointer

    // This would also not be valid:
    // q = &y; // ERROR: Cannot modify a const pointer

    return 0;
}
```

04

Hands on lab Practice


Have a Question
Leave a Comment!

Share

To let the others know more

Subscribe

To stay updated with latest videos

Thanks