**Variable length arguments  
 Writer:** Sayak Haldar

**To handle variable argument list <stdarg.h> is to be used.**

**Name:**

stdarg.h - handle variable argument list

**Synopsis:**

void va\_start(va\_list ap, argN);

void va\_copy(va\_list dest, va\_list src);

type va\_arg(va\_list ap, type);

void va\_end(va\_list ap);

**Description:**

The <stdarg.h> header shall contain a set of macros which allows portable functions that accept variable argument lists to be written. Functions that have variable argument lists (such as printf()) but do not use these macros are inherently non-portable, as different systems use different argument-passing conventions.

The type va\_list shall be defined for variables used to traverse the list.

The va\_start() macro is invoked to initialize ap to the beginning of the list before any calls to va\_arg().

The va\_copy() macro initializes dest as a copy of src, as if the va\_start() macro had been applied to dest followed by the same sequence of uses of the va\_arg() macro as had previously been used to reach the present state of src. Neither the va\_copy() nor va\_start() macro shall be invoked to reinitialize dest without an intervening

invocation of the va\_end() macro for the same dest.

The object ap may be passed as an argument to another function; if that function invokes the va\_arg() macro with parameter ap, the value of ap in the calling function is unspecified and shall be passed to the va\_end() macro prior to any further reference to ap. The parameter argN is the identifier of the rightmost parameter in the variable parameter list in the function definition (the one just before the ...).

**If the parameter argN is declared with the register storage class, with a function type or array type, or with a type that is not compatible with the type that results after application of the default argument promotions, the behavior is undefined.**

The va\_arg() macro shall return the next argument in the list pointed to by ap. Each invocation of va\_arg() modifies ap so that the values of successive arguments are returned in turn. The type parameter shall be a type name specified such that the type of a pointer to an object that has the specified type can be obtained simply by postfixing a '\*' to type. If there is no actual next argument, or if type is not compatible with the type of the actual next argument (as promoted according to the default argument promotions), the behavior is undefined, except for the following cases:

One type is a signed integer type, the other type is the corresponding unsigned integer type, and the value is representable in both types.

One type is a pointer to void and the other is a pointer to a character type.

Both types are pointers.

Different types can be mixed, but it is up to the routine to know what type of argument is expected.

The va\_end() macro is used to clean up; it invalidates ap for use (unless va\_start() or va\_copy() is invoked again).

Each invocation of the va\_start() and va\_copy() macros shall be matched by a corresponding invocation of the va\_end() macro in the same function.

Multiple traversals, each bracketed by va\_start() ... va\_end(), are possible.

**More about va\_arg**

type va\_arg (va\_list ap, type)

**It retrieves next argument.**

This macro expands to an expression of type type with the value of the current argument in the variable arguments list identified by ap.

Each call to this macro modifies the state of ap so that the next call to this macro will expand to the argument that follows the one it evaluates to.

**Notice that va\_arg cannot determine the actual type of the argument passed to the function, but uses whatever type is passed as the type macro argument as its type.**

Notice also that va\_arg does not determine either whether the retrieved argument is the last argument passed to the function (or even if it is an element past the end of that list). The function should be designed in such a way that the number of parameters can be inferred in some way by the values of either the named parameters or the additional arguments already read.

**Parameters**

**ap:** Object of type va\_list carrying information about the current retrieval state of a variable argument list. This object shall have been initialized by an initial call to va\_start or va\_copy and not have been released with va\_end.

**Type:** A type name. This type name is used as the type of the expression this macro expands to (i.e., its return type).

For a type expression to be suitable for its use with va\_arg, it must be such that type\* produces a pointer to type.

The type shall be compatible with type of the extracted argument (as promoted according to the default argument promotions), or one be the unsigned version of the other, or one be a void pointer and the other some other pointer type.

**Example:**

**/\* va\_arg example \*/**

**#include <stdio.h> /\* printf \*/**

**#include <stdarg.h> /\* va\_list, va\_start, va\_arg, va\_end \*/**

**int FindMax (int n, ...)**

**{**

**int i,val,largest;**

**va\_list vl;  
//list vl is initialized with the the variable list sent as argument to the findMax function**

**//n is the number of elements in the function**

**va\_start(vl,n);**

**largest=va\_arg(vl,int);**

**//largest is initialized to first argument of the variable length list argument**

**for (i=1;i<n;i++)**

**{**

**val=va\_arg(vl,int);**

**largest=(largest>val)?largest:val;**

**}**

**va\_end(vl);**

**//used for cleaning up purpose. Will there by a memory leak otherwise?**

**return largest;**

**}**

**int main ()**

**{**

**int m;**

**m= FindMax (7,702,422,631,834,892,104,772);**

**printf ("The largest value is: %d\n",m);**

**return 0;**

**}**

This FindMax function takes as its first parameter the number of additional arguments it is going to get. The first additional argument is retrieved and used as an initial reference for comparison, and then the remaining additional arguments are retrieved in a loop and compared to return the greatest one (which in this case is 892).

So, this can accept any type of argument and any number of arguments (the function

FindMax)

**More about va\_start**

**va\_start**

It initializes a variable argument list

Initializes ap to retrieve the additional arguments after parameter paramN.

A function that invokes va\_start, shall also invoke va\_end before it returns.

Parameters (of va\_start macro)

**ap:** Uninitialized object of type va\_list.

After the call, it carries the information needed to retrieve the additional arguments using va\_arg.

If ap has already been passed as first argument to a previous call to va\_start or va\_copy, it shall be passed to va\_end before calling this function.

**Paramn:** Name of the last named parameter in the function definition. The arguments extracted by subsequent calls to va\_arg are those after paramN.

**Return value:**

None

**Example:**

**/\* va\_start example \*/**

**#include <stdio.h> /\* printf \*/**

**#include <stdarg.h> /\* va\_list, va\_start, va\_arg, va\_end \*/**

**void PrintFloats (int n, ...)**

**{**

**int i;**

**double val;**

**printf ("Printing floats:");**

**va\_list vl;**

**va\_start(vl,n);**

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

**{**

**val=va\_arg(vl,double);**

**printf (" [%.2f]",val);**

**}**

**va\_end(vl);**

**printf ("\n");**

**}**

**int main ()**

**{**

**PrintFloats (3,3.14159,2.71828,1.41421);**

**return 0;**

**}**

The function PrintFloats takes the number of additional arguments as first argument (n), which are then sequentially read using the cstdarg macro and printed out with a specific format.  
  
%.2f: format specifier is used. It means we are not fixing the width of the whole float argument. But we are fixing the length of the number of digits printed after decimal point.

**Output:**

Printing floats: [3.14] [2.72] [1.41]

**More about va\_end**

**void va\_end (va\_list ap);**

End using variable argument list

Performs the appropriate actions to facilitate a normal return by a function that has used the va\_list object ap to retrieve its additional arguments.

This macro should be invoked before the function returns whenever va\_start has been invoked from that function.

**Parameters**

**va\_list:** previously initialized by va\_start or va\_copy

**Return value**

None

**Example**

**/\* va\_end example \*/**

**#include <stdio.h> /\* puts \*/**

**#include <stdarg.h> /\* va\_list, va\_start, va\_arg, va\_end \*/**

**void PrintLines (char\* first, ...)**

**{**

**char\* str;**

**va\_list vl;**

**str=first;**

**va\_start(vl,first);**

**do {**

**puts(str);**

**str=va\_arg(vl,char\*);**

**} while (str!=NULL);**

**va\_end(vl);**

**}**

**int main ()**

**{**

**PrintLines ("First","Second","Third","Fourth",NULL);**

**return 0;**

**}**

The PrintLines function takes a variable number or arguments. The first argument passed becomes parameter first, but the remaining are retrieved sequentially in the do-while loop using va\_arg until a null pointer is retrieved as the next argument.

**Output:**

First

Second

Third

Fourth

**Macro:**

**va\_copy**

**void va\_copy (va\_list dest, va\_list src);**

Copy variable argument list

dest as a copy of src (in its current state).

The next argument to be extracted from dest is the same as the one that would be extracted from src.

A function that invokes va\_copy, shall also invoke va\_end on dest before it returns.

**Parameters:**

**dest:** Uninitialized object of type va\_list.

After the call, it carries the information needed to retrieve the same additional arguments as src.

If src has already been passed as first argument to a previous call to va\_start or va\_copy, it shall be passed to va\_end before calling this function.

Src: Object of type va\_list that already carries information to retrieve additional arguments with va\_arg (i.e., it has already been passed as first argument to va\_start or va\_copy ans has not yet been released with va\_end).

**Question answers from it:**

**1.The standard header \_\_\_\_\_\_\_ is used for variable list arguments (…) in C.**

a) <stdio.h >

b) <stdlib.h>

c) <math.h>

d) <stdarg.h>

**Answer is d) <stdarg.h>**

**2.va\_end does whatever**

a) Cleanup is necessary

b) Must be called before the program returns.

c) Both a & b

d) None of the mentioned

**Answer is c) Both a & b**

**3.What is the output of this C code?**

#include <stdio.h>

#include <stdarg.h>

void func(int, ...);

int main()

{

func(2, 3, 5, 7, 11, 13);

return 0;

}

void func(int n, ...)

{

int number, i = 0;

va\_list start;

va\_start(start, n);

while (i != 3)

{

number = va\_arg(start, int);

i++;

}

printf("%d", number);

}

a) 3

b) 5

c) 7

d) 11

**Answer is c) 7**

**4.Which of the following function with ellipsis are illegal?**

a) void func(…);

b) void func(int, …);

c) void func(int, int, …);

d) Both (a) and (c)

**Answer is a)**

There should be at least one argument present in the function call which is of non variable length type.

**5.Which of the following data-types are promoted when used as a parameter for an ellipsis?**

a) char

b) short

c) int

d) None of the mentioned

**Answer is a) char**

**6.Which of the following macro extracts an argument from the variable argument list (i.e. ellipsis) and advance the pointer to the next argument?**

a) va\_list

b) va\_arg

c) va\_end

d) va\_start

**Answer is b) va\_arg**

**7.The type va\_list is used in an argument list**

a) To declare a variable that will refer to each argument in turn;

b) For cleanup

c) To create a list

d) There is no such type

**Answer is a) To declare a variable that will refer to each argument in turn**

**8.The declaration … can**

a) Appear anywhere in the function declaration

b) Only appear at the end of an argument list.

c) Nothing

d) Both a & b

**Answer is b) Only appear at the end of an argument list**

**9.Each call of va\_arg**

a) returns one argument

b) Steps va\_list variable to the next

c) Both a & b

d) None of the mentioned

**Answer is c) Both a & b**

**10.What is the output of this C code?**

#include <stdio.h>

int f(char chr, ...);

int main()

{

char c = 97;

f(c);

return 0;

}

int f(char c, ...)

{

printf("%c\n", c);

}

a) Compile time error

b) Undefined behaviour

c) 97

d) a

**Answer is d) a**

**11.What is the output of this C code?**

#include <stdio.h>

#include <stdarg.h>

int f(...);

int main()

{

char c = 97;

f(c);

return 0;

}

int f(...)

{

va\_list li;

char c = va\_arg(li, char);

printf("%c\n", c);

}

a) Compile time error

b) Undefined behaviour

c) 97

d) a

**Answer is a) Compile time error**

Because ... should be written at the end of the argument lists. But ... : it expects at least one or more arguments before it. That's why, this program would cause compilation error.

Compiler would say the same thing for line 3 and line 10 that:

ISO C requires a named argument before ‘...’

int f(...);

**12.What is the output of this C code?**

#include <stdio.h>

#include <stdarg.h>

int f(char c, ...);

int main()

{

char c = 97, d = 98;

f(c, d);

return 0;

}

int f(char c, ...)

{

va\_list li;

va\_start(li, c);

char d = va\_arg(li, char);

printf("%c\n", d);

va\_end(li);

}

a) Compile time error

b) Undefined behaviour

c) a

d) b

**Answer is b) Undefined behaviour**

Compiler would throw a warning saying: warning: ‘char’ is promoted to ‘int’ when passed through ‘...’ [enabled by default]

in char d = va\_arg(li,char);

1. **What is the output of this C code?**

#include <stdio.h>

#include <stdarg.h>

int f(char c, ...);

int main()

{

char c = 97, d = 98;

f(c, d);

return 0;

}

int f(char c, ...)

{

va\_list li;

va\_start(li, c);

char d = va\_arg(li, int);

printf("%c\n", d);

va\_end(li);

}

a) Compile time error

b) Undefined behaviour

c) a

d) b

**Answer is d) b**

Since, char is actually promoted to int when passed through '...'

**14.What is the output of this C code?**

#include <stdio.h>

#include <stdarg.h>

int f(int c, ...);

int main()

{

int c = 97;

float d = 98;

f(c, d);

return 0;

}

int f(int c, ...)

{

va\_list li;

va\_start(li, c);

float d = va\_arg(li, float);

printf("%f\n", d);

va\_end(li);

}

a) Compile time error

b) Undefined behaviour

c) 97.000000

d) 98.000000

**Answer is b) Undefined behaviour.**

Compiler would say ‘float’ is promoted to ‘double’ when passed through ‘...’ [enabled by default] in statement:float d = va\_arg(li, float);

so,we should pass ‘double’ not ‘float’ to ‘va\_arg’

**Extras:  
  
sample printf implementation using variable length argument:**

#include <stdarg.h>

void minimal\_printf(char \*fmt, ...)

{

va\_list ap; /\* points to each unnamed arg in turn \*/

char \*p, \*sval;

int ival;

double dval;

va\_start(ap, fmt); /\* make ap point to 1st unnamed arg \*/

for (p = fmt; \*p; p++)

{

if (\*p != '%')

{

putchar(\*p);

continue;

}

switch (\*++p)

{

case 'd':

ival = va\_arg(ap, int);

printf("%d", ival);

break;

case 'f':

dval = va\_arg(ap, double);

printf("%f", dval);

break;

case 's':

for (sval = va\_arg(ap, char \*); \*sval; sval++)

putchar(\*sval);

break;

default:

putchar(\*p);

break;

}

}

va\_end(ap); /\* clean up when done \*/

}

# Variable length arguments for Macros

Like functions, we can also pass variable length arguments to macros. For this we will use the following preprocessor identifiers.

To support variable length arguments in macro, we must include ellipses (…) in macro definition. There is also “\_\_VA\_ARGS\_\_” preprocessing identifier which takes care of variable length argument substitutions which are provided to macro. Concatenation operator ## (aka paste operator) is used to concatenate variable arguments.

Let us see with example. Below macro takes variable length argument like “printf()” function. This macro is for error logging. The macro prints filename followed by line number, and finally it prints info/error message. First arguments “prio” determines the priority of message, i.e. whether it is information message or error, “stream” may be “standard output” or “standard error”. It displays INFO messages on stdout and ERROR messages on stderr stream.

**#include <stdio.h>**

**#define INFO 1**

**#define ERR 2**

**#define STD\_OUT stdout**

**#define STD\_ERR stderr**

**#define LOG\_MESSAGE(prio, stream, msg, ...) do {\**

**char \*str;\**

**if (prio == INFO)\**

**str = "INFO";\**

**else if (prio == ERR)\**

**str = "ERR";\**

**fprintf(stream, "[%s] : %s : %d : "msg" \n", \**

**str, \_\_FILE\_\_, \_\_LINE\_\_, ##\_\_VA\_ARGS\_\_);\**

**} while (0)**

**int main(void)**

**{**

**char \*s = "Hello";**

**/\* display normal message \*/**

**LOG\_MESSAGE(ERR, STD\_ERR, "Failed to open file");**

**/\* provide string as argument \*/**

**LOG\_MESSAGE(INFO, STD\_OUT, "%s Geeks for Geeks", s);**

**/\* provide integer as arguments \*/**

**LOG\_MESSAGE(INFO, STD\_OUT, "%d + %d = %d", 10, 20, (10 + 20));**

**return 0;**

**}**

Now, \_\_VA\_ARGS\_\_ for variable length arguments.  
  
,##\_\_VA\_\_ARGS\_\_ : will trim the last comma if there is no argument.