realloc() — Change reserved storage block size

Last Updated: 2021-06-25

Standards

Standards / Extensions	C or C++	Dependencies
ISO C POSIX.1 XPG4 XPG4.2 C99 Single UNIX Specification, Version 3	both	

Format

```
#include <stdlib.h>

void *realloc(void *ptr, size_t size);
```

General description

Changes the size of a previously reserved storage block. The *ptr* argument points to the beginning of the block. The *size* argument gives the new size of the block in bytes. The contents of the block are unchanged up to the shorter of the new and old sizes.

If the *ptr* is NULL, realloc() reserves a block of storage of *size* bytes. It does not give all bits of each element an initial value of 0.

If *size* is 0 and *ptr* is not NULL, the storage pointed to by *ptr* is freed and NULL is returned.

If you use realloc() with a pointer that does not point to a *ptr* created previously by malloc(), calloc(), or realloc(), or if you pass *ptr* to storage already freed, you get undefined behavior—usually an exception.

If you ask for more storage, the contents of the extension are undefined and are not guaranteed to be 0.

The storage to which the returned value points is aligned for storage of any type of object. Under z/OS® XL C only, if 4K alignment is required, the __4kmalc() function should be used. (This function is only available to C applications in stand-alone System Programming C (SPC) Facility applications.) The library functions specific to the System Programming C (SPC) environment are described in z/OS XL C/C++ Programming Guide.

To investigate the cause of realloc() running out of heap storage, see z/OS Language Environment Programming Reference

> **Note:** The environment variable *_CEE_REALLOC_CONTROL* controls reallocation that can lead to improved application performance. For more information about the *_CEE_REALLOC_CONTROL* environment variable, see z/OS XL C/C++ Programming Guide.

Special behavior for C++

The C++ keywords new and delete are not interoperable with calloc(), free(), malloc(), or realloc().

Returned value

If successful, realloc() returns a pointer to the reallocated storage block. The storage location of the block might be moved. Thus, the returned value is not necessarily the same as the *ptr* argument to realloc().

The returned value is NULL if *size* is 0. If there is not enough storage to expand the block to the given size, the original block is unchanged and a NULL pointer is returned. If realloc() returns NULL because there is not enough storage, it will also set errno to one of the following values:

Error Code
Description

ENOMEM
Insufficient memory is available

Example

CELEBR06

```
/* CELEBR06

   This example allocates storage for the prompted size of array
   and then uses &realloc. to reallocate the block to hold the
```

```
   new size of the array.
   The contents of the array are printed after each allocation.

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

int main(void)
{
  long * array;    /* start of the array */
  long * ptr;      /* pointer to array   */
  int    i;        /* index variable     */
  int  num1, num2; /* number of entries of the array */

  void print_array( long *ptr_array, int size);

  printf( "Enter the size of the array\n" );
  scanf( "%i", &num1 );

  /* allocate num1 entries using malloc() */
  if ( (array = (long *)malloc( num1 * sizeof( long ))) != NULL ) {
     for ( ptr = array, i = 0; i < num1 ; ++i ) /* assign values */
        *ptr++ = i;
     print_array( array, num1 );
     printf("\n");
  }
  else { /*  malloc error  */
    printf( "Out of storage\n" );
    abort();
  }

  /* Change the size of the array ... */
  printf( "Enter the size of the new array\n" );
  scanf( "%i", &num2);

  if ( (array = (long *)realloc( array, num2* sizeof( long ))) != NULL )
  {
     for ( ptr = array + num1, i = num1; i <= num2; ++i )
        *ptr++ = i + 2000;  /* assign values to new elements */
     print_array( array, num2 );
  }

  else { /* realloc error */
    printf( "Out of storage\n" );
```

```
    abort();
  }
}

void print_array( long  * ptr_array, int size )
{
  int i;
  long * index = ptr_array;

  printf("The array of size %d is:\n", size);
  for ( i = 0; i < size; ++i )           /* print the array out    */
    printf( "  array[ %i ] = %li\n", i, ptr_array[i] );
}
```

Output: If the initial value entered is 2 and the second value entered is 4, then expect the following output:

```
Enter the size of the array
The array of size 2 is:
  array[ 0 ] = 0
  array[ 1 ] = 1

Enter the size of the new array
The array of size 4 is:
  array[ 0 ] = 0
  array[ 1 ] = 1
  array[ 2 ] = 2002
  array[ 3 ] = 2003
```

Related information

- “System Programming C (SPC) Facilities” in z/OS XL C/C++ Programming Guide
- spc.h – System library functions and storage allocation
- stdlib.h – Standard library functions

- calloc() – Reserve and initialize storage
- free() – Free a block of storage
- malloc() – Reserve storage block

Parent topic:

→ Library functions

Previous
readv() – Read data on a file or socket and store in a set of buffers

Next
realpath() – Resolve path name