Write your own memcpy() and memmove()

The memcpy function is used to copy a block of data from a source address to a destination address. Below is its prototype.

```
void * memcpy(void * destination, const void * source, size_t num);
```

The idea is to simply typecast given addresses to char *(char takes 1 byte). Then one by one copy data from source to destination. Below is implementation of this idea.

```
// A C implementation of memcpy()
#include<stdio.h>
#include<string.h>

void myMemCpy(void *dest, void *src, size_t n)
{
   // Typecast src and dest addresses to (char *)
   char *csrc = (char *)src;
   char *cdest = (char *)dest;

   // Copy contents of src[] to dest[]
   for (int i=0; i<n; i++)
       cdest[i] = csrc[i];
}

// Driver program
int main()
{
   char csrc[] = "GeeksforGeeks";
   char cdest[100];
   myMemCpy(cdest, csrc, strlen(csrc)+1);
   printf("Copied string is %s", cdest);

   int isrc[] = {10, 20, 30, 40, 50};
   int n = sizeof(isrc)/sizeof(isrc[0]);
   int idest[n], i;
   myMemCpy(idest, isrc,  sizeof(isrc));
   printf("\nCopied array is ");
   for (i=0; i<n; i++)
     printf("%d ", idest[i]);
   return 0;
}
```

Run on IDE

Output:

```
Copied string is GeeksforGeeks
Copied array is 10 20 30 40 50
```

What is memmove()?

memmove() is similar to memcpy() as it also copies data from a source to destination. memcpy() leads to problems when source and destination addresses overlap as memcpy() simply copies data one by one from one location to another. For example consider below program.

```c
// Sample program to show that memcpy() can loose data.
#include <stdio.h>
#include <string.h>
int main()
{
   char csrc[100] = "Geeksfor";
   memcpy(csrc+5, csrc, strlen(csrc)+1);
   printf("%s", csrc);
   return 0;
}
```

Run on IDE

Output:

```
GeeksGeeksGeek
```

Since the input addresses are overlapping, the above program overwrites the original string and causes data loss.

```c
// Sample program to show that memmove() is better than memcpy()
// when addresses overlap.
#include <stdio.h>
#include <string.h>
int main()
{
   char csrc[100] = "Geeksfor";
   memmove(csrc+5, csrc, strlen(csrc)+1);
   printf("%s", csrc);
   return 0;
}
```

Run on IDE

Output:

```
GeeksGeeksfor
```

How to implement memmove()?

The trick here is to use a temp array instead of directly copying from src to dest. The use of temp array is important to handle cases when source and destination addresses are overlapping.

```
//C++ program to demonstrate implementation of memmove()
#include<stdio.h>
#include<string.h>

// A function to copy block of 'n' bytes from source
// address 'src' to destination address 'dest'.
void myMemMove(void *dest, void *src, size_t n)
{
   // Typecast src and dest addresses to (char *)
   char *csrc = (char *)src;
   char *cdest = (char *)dest;

   // Create a temporary array to hold data of src
   char *temp = new char[n];

   // Copy data from csrc[] to temp[]
   for (int i=0; i<n; i++)
       temp[i] = csrc[i];

   // Copy data from temp[] to cdest[]
   for (int i=0; i<n; i++)
       cdest[i] = temp[i];

   delete [] temp;
}

// Driver program
int main()
{
   char csrc[100] = "Geeksfor";
   myMemMove(csrc+5, csrc, strlen(csrc)+1);
   printf("%s", csrc);
   return 0;
}
```

Run on IDE

Output:

```
GeeksGeeksfor
```

Optimizations:

The algorithm is inefficient (and honestly double the time if you use a temporary array). Double copies should be avoided unless if it is really impossible.

In this case though it is easily possible to avoid double copies by picking a direction of copy. In fact this is what the memmove() library function does.

By comparing the src and the dst addresses you should be able to find if they overlap.

– If they do not overlap, you can copy in any direction

– If they do overlap, find which end of dest overlaps with the source and choose the direction of copy accordingly.

– If the beginning of dest overlaps, copy from end to beginning

– If the end of dest overlaps, copy from beginning to end

– Another optimization would be to copy by word size. Just be careful to handle the boundary conditions.
– A further optimization would be to use vector instructions for the copy since they're contiguous.

This article is contributed by Saurabh Jain. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Recommended Posts:

How to write a running C code without main()?
Commonly Asked C Programming Interview Questions | Set 1
Implement Your Own sizeof
Quine – A self-reproducing program
Implement your own itoa()
Fork() – Practice questions
Undefined Behavior in C and C++
Print 1 2 3 infinitely using threads in C
Heap overflow and Stack overflow
How to clear console in C language?