

Forum:[Hack Forums](#)

Thread: [Bubble Sort C program](#)

madera - November 5, 2015, 9:08 am

I'm not sure what I'm doing wrong with this program...first and foremost it gets locked in an infinite loop whenever the function "swap()" is called. I'm used to java so its possible im overlooking something simple ?

```
Code #include <stdio.h>
#include <stdlib.h>
```

```
int swap(int arg0, int arg1)
{
    int temp;

    temp = arg0;
    arg0=arg1;
    arg1=temp;

    return 0;
}
```

```
int main(int argc, const char * argv[]) {
    int i;
    int arr[i];
    int newi;

    printf("input number of elements in list: \n");
    scanf("%i", &i);

    int origi = i;

    for (int j=0; j < i; j++)
    {
        printf("input element: \n");
        scanf("%i", &arr[j]);
    }

    while(i!=0)
    {
        newi = 0;
        for (int x=0; x<i; x++)
        {
            if (arr[x-1] > arr[x])
            {
```

```

        swap(arr[x],arr[x-1]);
        newi = i;
    }
}
i = newi;
}

```

```

printf("sorted list is:\n");
for (int k = 0; k < origi; k++)
    printf("%i \n", arr[k]);

return 0;
}

```

bbbysmth - November 5, 2015, 9:14 am

In your swap function, try passing the values as pointers.

madera - November 5, 2015, 9:28 am

Thanks for the replies, I fixed it :)

```

Code #include <stdio.h>
#include <stdlib.h>

```

```

int swap(int *arg0, int *arg1)
{
    int temp;

    temp = *arg0;
    *arg0=*arg1;
    *arg1=temp;

    return 0;
}

```

```

int main(int argc, const char * argv[]) {
    int i;
    int newi;

    printf("input number of elements in list: \n");
    scanf("%i", &i);

    int arr[i];
    int origi = i;

    for (int j=0; j < i; j++)
    {
        printf("input element: \n");
    }
}

```

```

scanf("%i", &arr[j]);
}

while(i!=0)
{
    newi = 0;
    for (int x=0; x<i; x++)
    {
        if (arr[x-1] > arr[x])
        {
            swap(&arr[x-1], &arr[x]);
            newi = i;
        }
    }
    i = newi;
}

printf("sorted list is:\n");
for (int k = 0; k < origi; k++)
    printf("%i \n", arr[k]);

return 0;
}

```

Any explanation as to why I had to use pointers instead of an immediate array value?

madera - November 5, 2015, 9:50 am

Imao exact same example...ok, I get that. What I still don't understand is why direct (non pointer) variables are not "permanently" changed when they are passed to a function? I'm sure its possible that the (non pointer) variables being passed to the function are stored in a stack before they are passed to the function so that they can be retrieved when the function is returned.

More specifically, is this quirk just a characteristic of the C language in order to make it more versatile?

bbbysmth - November 5, 2015, 10:08 am

(Nov 5, 2015 10:50 AM)madera Wrote: [CITE](#) Imao exact same example...ok, I get that. What I still don't understand is why direct (non pointer) variables are not "permanently" changed when they are passed to a function? I'm sure its possible that the (non pointer) variables being passed to the function are stored in a stack before they are passed to the function so that they can be retrieved when the function is returned.

More specifically, is this quirk just a characteristic of the C language in order to make it more versatile?

Values are copied when they are passed to a function. This is why you have to use pointers if you want to change the original.

morph - November 5, 2015, 11:46 am

Read up on pointers. ?

0xDEAD10CC - November 7, 2015, 11:41 am

By passing a pointer, the pointer itself is copied but regardless it doesn't matter -- you are passing

an address to a specific location in memory where the original variable exists in the caller's scope and the address is all we care about for information, to which you can dereference and edit directly. If you pass a value, the value is a copy, you edit the copy and the copy is discarded at the return from the function with the original data remaining unchanged.

These are pointer fundamentals.