

Forum:[Hack Forums](#)

Thread: [Filling a struct with data \(scanf problems\)](#)

SirSyn - January 5, 2016, 11:40 am

Hi,

I am practicing the use of structs and malloc. The code below works fine whenever I try to read the data of a single person.

However, when I use a loop to fill my array with multiple people, the "read_new" function doesn't work anymore.

The second time, it skips the scanf that reads a name, and immediately wants to read the email.

Can anyone explain to me why this happens? And how I can fix / prevent this.

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

```
#define BUFFER 200
#define MAX 2
```

```
typedef struct{
    char *name;
    char email[25];
    int age;
    double weight;
}person;
```

```
person* read_new();
void print_people(person *people);
```

```
int main(int argc,char **argv){
    int i;
    person *people[MAX];
    for(i=0;i<MAX;i++){
        people[i] = read_new();
        printf("Person %d has now been added to the array\n",i);
    }
    print_people(*people);
    return 0;
}
```

```
person* read_new(){
    person* temp;
    char *buffer;
    int length,age,weight;

    temp = (person*)malloc(sizeof(person));
```

```

buffer = (char*)malloc(BUFFER*sizeof(char)+1);

printf("Name :");
scanf("%s",buffer);
length = strlen(buffer);
temp->name = (char*)malloc(length*sizeof(char)+1);
strcpy(temp->name,buffer);

printf("\nEmail :");
scanf("%s",buffer);
length = strlen(buffer);
strcpy(temp->email,buffer);

printf("\nAge :");
scanf("%d",&age);
temp->age = age;

printf("\nWeight :");
scanf("%l",&weight);
temp->weight = weight;

free(buffer);

return temp;
}

void print_people(person *people){
    person *indexer = people;
    while(indexer!=people+MAX){
        printf("\nName = %s, email = %s, age = %d, weight = %f",indexer->name,indexer->email,indexer->age,indexer->weight);
        indexer++;
    }
}

```

RISC - January 5, 2016, 11:57 am

This is all terrible practice.

SirSyn - January 5, 2016, 12:18 pm

(Jan 5, 2016 12:57 PM)RISC Wrote: [\[CITE\]](#) This is all terrible practice.

Can you be more specific? I'm here to learn and I can't do a lot with "this is all terrible practice".

Thanks for the fast reply though.

RISC - January 5, 2016, 1:16 pm

(Jan 5, 2016 01:18 PM)SirSyn Wrote: [\[CITE\]](#) Can you be more specific? I'm here to learn and I can't do a lot with "this is all terrible practice".

Thanks for the fast reply though.

Using scanf.

Not placing your zero terminator.

Not freeing all your memory.

zNinja - January 5, 2016, 2:14 pm

For every call of malloc or calloc you need to "free" allocated memory from heap.

Also for every new you need to delete memory,

for every new [], you need to delete [] memory.

Using scanf can lead to buffer overflow.

Did not seen your code detailed just giving you more details of what RISC has mean

0xDEAD10CC - January 5, 2016, 6:09 pm

(Jan 5, 2016 02:16 PM)RISC Wrote: [\[CITE\]](#) Using scanf.

Not placing your zero terminator.

Not freeing all your memory.

? Using scanf() is not bad practice.. :S It is improper usage of people who don't understand scanf() that makes it bad.

First, the problem with the struct itself is that the name parameter is just a pointer. You either need a long lasting character array on the stack that it'll point to so that you don't end up with a dangling pointer, or you'll have one pointer to some data on the heap, and the struct itself will hold the actual data for the email, which doesn't make sense.

Your read function allocates a buffer on the heap to copy data over to another memory block allocated on the heap for the name field of the struct, and both of these operations are using buffer overflow vulnerable function calls; scanf() with "%s" and strcpy() instead of strncpy().

Your print function requires a pointer, and you pass a dereferenced array-of-pointers pointer:

Code print_people(*people);

And you don't have a free_people function or anything of that nature to free the memory you allocated on the heap for each of the names.

Really you should be passing a pointer to a pointer to iterate over the array of pointers.

You're also writing this as C++ code I would assume as casting from a void pointer to a pointer of another type does not require a cast in C, you don't initialize your pointers that aren't immediately assigned to NULL, and sizeof(char) is redundant as it's guaranteed to be 1.

The other thing I see is that you are mixing floating point and integers. You declare weight as a double (floating point) in the struct. Your read function reads weight into an integer variable, then assigns it to the object on the heap, before returning a pointer to that object, then you print the result as a floating point number. ?

Theres still more wrong with this code too that I can see

(Jan 5, 2016 03:14 PM)zNinja Wrote: [\[CITE\]](#) For every call of malloc or calloc you need to "free" allocated memory from heap.

Also for every new you need to delete memory,

for every new [], you need to delete [] memory.

Using scanf can lead to buffer overflow.

Did not seen your code detailed just giving you more details of what RISC has mean

No.. using scanf() improperly will lead to a potential buffer overflow. If you haven't been working with C for very long you probably shouldn't be using it carelessly, otherwise there are still ways to prevent it from being vulnerable to a buffer overflow by modifying the function call.

For those of you who would suggest fgets(), it's still just as easy to run into a buffer overflow with a function that allows you to specify the size as a parameter if you don't know what you're doing.

Blame your own stupidity though not the function.