C-STRINGS AND SCOPE

An Interview Question

Delroy A. Brinkerhoff

WHAT'S WRONG WITH THIS CODE?

```
char* get_name()
{
   char name[100];
   cin.getline(name, 100);
      .
      .
      .
   return name;
}
```

TESTS OUR UNDERSTANDING OF FUNDAMENTAL CONCEPTS

- How to define and read C-strings
  - As a character array
  - Read with the getline function
- When is the memory for automatic variables allocated and deallocated?
  - Allocated when the variable comes into scope
  - Deallocated when the variable goes out of scope
- The relationship between pointers and arrays
  - The name of an array is address of the array
- How arrays are passed to and returned from functions
  - Always passed in and returned by pointer

WHAT'S WRONG WITH THE CODE?

```
char* get_name()
{
    char   name[100];
    cin.getline(name, 100);
        .
        .
        .
    return name;
}
```

- Correct
  - Definition of name
  - Return type
  - getline and all arguments
  - return statement
- Wrong
  - Returning the address of deallocated memory

CORRECTING THE *LOGICAL* ERROR

- Knowing *how* to correct the error is not the same as knowing *what* the error is
- Three standard corrections
  - static data
  - dynamic data
  - calling-scope data

STATIC DATA

```
char* get_name()
{
    static  char  name[100];
    cin.getline(name, 100);
        .
        .
        .
    return name;
}
```

- The static keyword makes data that is just the opposite of automatic data
- The static keyword does not effect scope – "name" is only accessible in "get_name"
- Memory is allocated when the program is first loaded into memory for execution
- Memory is not deallocated until the program exits

ADVANTAGES AND DISADVANTAGES

ADVANTAGES

- Simple to understand
- Easy to use
- Potentially the fastest

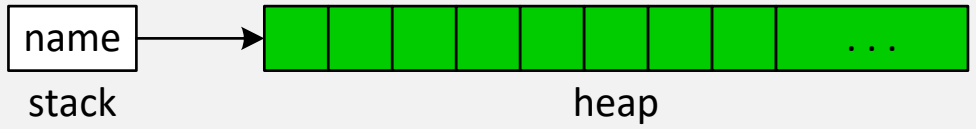
DISADVANTAGES

- Increases the total memory requirement of a program
- Functions w/static variables can't be recursive or reentrant
- Each call overwrites previous data
  - Must finish all data processing
  - Cannot store data without copying

DYNAMIC DATA

```
char* get_name()
{
    char*  name = new char[100];
    cin.getline(name, 100);
        .
        .
        .
    return name;
}
```

- Memory allocated with new is only deallocated with delete
- The variable "name" is deallocated but not the memory to which it points



ADVANTAGES AND DISADVANTAGES

ADVANTAGES

- Each returned C-string has it own memory
  - Can process multiple data items at the same time
  - Can store the data directly

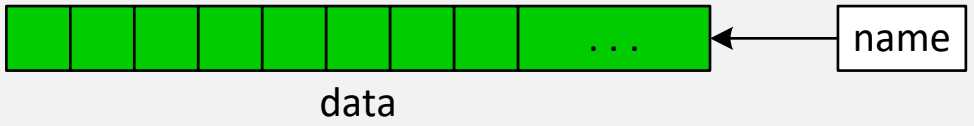
DISADVANTAGES

- Potentially slower
- Must remember to delete the returned data to avoid a memory leak

CALLING-SCOPE DATA

```
void client()
{
    char data[100];
    get_name(data);
        .
        .
        .
    // use data
}
```

```
char* get_name(char* name)
{
    cin.getline(name, 100);
        .
        .
        .
    return name;
}
```



ADVANTAGES AND DISADVANTAGES

ADVANTAGES

- Makes the client function responsible for memory management
- Easy to understand and to use
  - All the advantages of automatic data
  - All of the flexibility of dynamic data
- Can be combined with dynamic option

DISADVANTAGES

- The function call has one or two more arguments than other solutions
- The defined size of the array must agree with the second getline argument
  - `char data[100];`
  - `cin.getline(name, 100);`

COMBINING THE SCOPE AND DYNAMIC SOLUTIONS

```
char* get_name(char* name, int size)
{
    if (name == NULL)
        name = new char[size];

    cin.getline(name, size);
        .
        .
        .
    return name;
}
```

```
void client()
{
    char* data = get_name(NULL, 100);

    // use data
        .
        .
        .
        .
}
```