More C Strings

Sunday, September 27, 2020 12:51 PM

Searching in strings

strchr. Returns pointer to first occurrence of char in string, or NULL if not found.

ex.
```
char daisy[6];
strcpy(daisy, "Daisy");
char* letterA = strchr(daisy, 'a');
printf("%s\n", daisy);   // Daisy
printf("%s\n", letterA); // aisy
```

Note: strrchr does this in reverse

strstr Returns pointer to first occurrence of string in string, or NULL if not found.

strspn Returns length of initial part of first string which contains only characters in the second string.

ex.
```
char daisy[10];
strcpy(daisy, "Daisy Dog");
int spanLength = strspn(daisy, "aDeoi"); // 3
```

Note: strcspn does this but for characters not in the second string

Buffer overflows and Valgrind

Buffer overflow. Copying object in space too small leads to overwriting other program memory.

Valgrind. Memory analysis utility to help catch these errors

Pointers

Defn. A pointer is a variable type that stores a memory address.

One 8-byte pointer can refer to any size memory location.

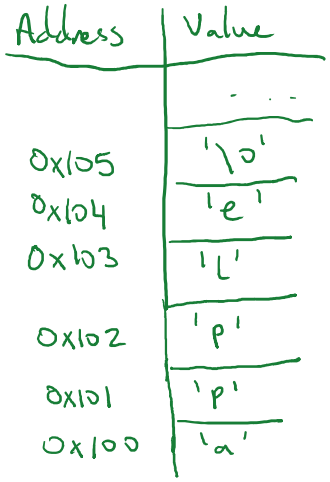
Essential to heap allocation

Can address memory generically.

Memory. Array of bytes.

Each byte has unique numeric index written in hexadecimal.

Pointers store these addresses



Passing by value. When passing a parameter in C, we pass a copy rather than a reference.

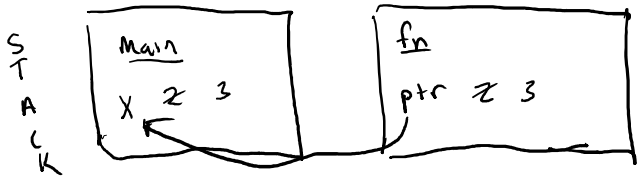
Instead of passing directly, pass memory address (the pointer)

Syntax.

```
int x = 2;
// &- get memory address of (takes T, yields *T)
int* ptr = &x;
// * - dereference/get value at addr (takes *T, yields T)
printf("%d", *xPtr);
```

ex.
```
void fn(int* ptr) {
    *ptr = 3;
}

void main(int argc, char* argv[]) {
    int x = 2;
    fn(&x);
    printf("%d\n", x);   // 3
}
```



Strings in Memory

Behavior 1. If we create a string as a char[], we can modify its characters because its memory lives within our stack space.

Why? When declaring char[], contiguous memory allocated on stack to store array contents, can modify stack memory.

ex.
```
char str[6];
strcpy(str, "apple");
```

Behavior 2. We can't set a char[] equal to another value, because it refers to a block of memory reserved for the original array (i.e., it is not a pointer).

ex.
```
char str[6];
strcpy(str, "apple");
char str2[7];
strcpy(str2, "orange");
str = str2; // not allowed!
```

Note: cannot resize array after creation.

Behavior 3. If we pass a char[] as a parameter, set something equal to it, or perform arithmetic with it, it is automatically converted to a char*.

ex.
```
void fn(char* str) {
    ...
}

int main(int argc, char* argv[]) {
    char str[5];
    strcpy(str, "rice");
    fn(str);
    char* str2 = str;
    char* str3 = str+2;
    return 0;
}
```

Behavior 4. If we create a new string with new characters as a char*, we cannot modify its characters because its memory lives in the data segment.

ex. `char* str = "Hello, world!";`

`str[0] = 'h'; // cannot modify it!`

Note: pointer refers to address of the first character of the string in the data segment.

Note: Only applies to creating a new string out of a string literal

Note: cannot check in code if string is modifiable - must state this in documentation

Behavior 5. We can set `char*` equal to another value, because it is a reassignable pointer.

ex. `char* str = "apple";`

`char* str2 = "orange";`

`str = str2;`

Behavior 6. Adding an offset of n to a C string gives us a substring n places past the first character.

ex.
```
char* str = "apple";  // 0xff0
char* str2 = str + 2; // 0xff2

printf("%s\n", str);  // apple
printf("%s\n", str2); // ple

char thirdLetter = str[3];  // 'l'
char thirdLetter = *(str+3); // 'l'
```

Behavior 7. If we change characters in a string parameter, these changes will persist outside the function.