Algoritmos y estructuras de datos

Aritmética de direcciones y cadenas

Francisco Javier Zaragoza Martínez

Universidad Autónoma Metropolitana Unidad Azcapotzalco
Departamento de Sistemas

Universidad Autónoma Metropolitana
Casa abierta al tiempo Azcapotzalco

Posgrado en Optimización

7 de abril de 2021

Citas

Perry Farrar

Si le mientes a la computadora, te atrapará.

John Barnes

Jugar con apuntadores es como jugar con fuego. El fuego es tal vez la más importante herramienta conocida por la humanidad. Usado con cuidado trae enormes beneficios; pero si se sale de control el desastre ataca.

Charles Babbage

En dos ocasiones me han preguntado “Díganos Sr. Babbage, si le mete a la máquina los datos equivocados ¿saldrán las respuestas correctas?” No soy capaz de apreciar correctamente la clase de confusión de ideas que pudiera provocar tal pregunta.

Apuntadores y memoria

1. Si `tipo v` es una variable, entonces `&v` es su referencia.
2. Además `tipo *p` declara a `p` como un apuntador a `tipo`.
3. Si hacemos `p = &v`, entonces `*p` es `v`.

Apuntadores y arreglos

1. Si `tipo a[N]` es un arreglo, entonces `a` es su referencia.
2. Si hacemos `p = a`, entonces `p+i` vale `&a[i]` y `*(p+i)` es `a[i]`.
3. Los apuntadores a elementos de un arreglo se pueden incrementar `p++`, decrementar `p--` y comparar para moverse en un arreglo.

Cadenas

Cadenas en la memoria

En C las cadenas se implementan como arreglos de caracteres (**char**) que terminan en el carácter nulo `'\0'` (que vale `0`). La memoria

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
69	74	69	77	80	76	79	83	32	68	69	32	85	65	77	0

se interpreta como la cadena `"EJEMPLOS DE UAM"`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
E	J	E	M	P	L	O	S	`' '`	D	E	`' '`	U	A	M	`'\0'`

Cadenas

Declaración e inicialización

Declaración de cadenas

Una cadena se declara como un arreglo de **char** que tenga espacio suficiente para todos los caracteres y el carácter nulo.

```
char s[10]; // cadena de hasta  9 caracteres
char t[11]; // cadena de hasta 10 caracteres
```

Inicialización de cadenas

Una cadena se puede inicializar desde la declaración:

```
char s[10] = "ejemplo"; // 7+nulo y sobran 2
char t[] = "ejemplo";   // arreglo de 8 char
```

También se puede inicializar posteriormente, pero se debe recordar que antes de eso contiene basura y no se debe olvidar colocar el carácter nulo.

Cadenas

Lectura

Palabras

Si `s` es una cadena suficientemente larga, entonces `scanf("%s", s)` lee en `s` una palabra sin blancos (espacios, saltos de línea, etc.) de la entrada estándar.

Líneas

A diferencia de esto `gets(s)` lee en `s` una línea de la entrada estándar, remplazando el carácter `\n` por un nulo. Evita leer cadenas de esta forma.

Entrada limitada

Finalmente `fgets(s, n, stdin)` lee en `s` hasta `n-1` caracteres de la entrada estándar (puede detenerse antes si encuentra un `\n` o se acaba la entrada). No remplaza `\n`, pero sí agrega un `\0`. Si no pudo leer nada, regresa `NULL`. Esta es la forma correcta de leer una cadena en un arreglo.

Cadenas

Escritura

Escritura normal

Si `s` es una cadena, entonces `printf("%s", s)` escribe el contenido de la cadena `s` en la salida estándar. Lo mismo hace `fputs(s, stdout)`.

Escritura con fin de línea

Por otro lado `puts(s)` escribe el contenido de la cadena `s` en la salida estándar y agrega el carácter `\n` al final.

Cadenas

Cosas que tal vez no sabías

`printf` y compañía

`printf` regresa la cantidad de caracteres que imprimió (o un número negativo si algo falló). Además, `sprintf` sirve para generar una cadena con formato (en lugar de imprimir a la salida estándar).

`scanf` y compañía

`scanf` regresa la cantidad de variables que leyó (o `EOF` si algo falló). Además, `sscanf` lee de una cadena (en lugar de la entrada estándar).

`getchar` y `putchar`

Para leer o escribir caracteres individuales se usa `getchar` y `putchar`.

Cadenas

Biblioteca estándar

El lenguaje C no tiene operaciones de cadenas. Sin embargo, tiene una biblioteca a la que se tiene acceso usando `#include <string.h>` con decenas de funciones que operan sobre cadenas.

Operaciones de cadenas

Nosotros implementaremos algunas funciones de cadenas para practicar el uso de arreglos, apuntadores y aritmética de direcciones:

1. Longitud de una cadena (`int strlen(char *s)`).
2. Copia de cadenas (`char* strcpy(char *s, char *t)`).
3. Concatenación de cadenas (`char* strcat(char *s, char *t)`).
4. Comparación de cadenas (`int strcmp(char *s, char *t)`).

Longitud de una cadena

Con arreglos

```
int longitud(char s[]) {
  int   i = 0;          // al principio
  while (s[i] != '\0') // si no es nulo
    i++;                // cuenta
  return i;
}
```

Longitud de una cadena

Con apuntadores

```
int longitud(char *s)  {
  char *p = s;         // al principio
  while (*p != '\0')   // si no es nulo
    p++;               // cuenta
  return p-s;          // resta
}
```

Longitud de una cadena

Con apuntadores, sin comparación explícita

```
int longitud(char *s)  {
  char *p = s;         // al principio
  while (*p)           // si no es nulo
    p++;               // cuenta
  return p-s;          // resta
}
```

Copia de cadenas

Con arreglos

```
void copia(char s[], char t[]) {
  int i = 0;                        // al principio
  while ((s[i] = t[i]) != '\0') // si no es nulo
    i++;
}
```

Copia de cadenas

Con apuntadores

```
void copia(char *s, char *t)   {
  // los apuntadores ya apuntan al lugar correcto
  while ((*s = *t) != '\0') {   // si no es nulo
    s++; t++;}
}
```

Copia de cadenas

Con apuntadores, sin comparación explícita

```
void copia(char *s, char *t)   {
  // los apuntadores ya apuntan al lugar correcto
  while (*s = *t) {              // si no es nulo
    s++; t++;}
}
```

Copia de cadenas

Con apuntadores, incremento inmediato

```
void copia(char *s, char *t)   {
  // los apuntadores ya apuntan al lugar correcto
  while (*s++ = *t++)            // si no es nulo
    ;
}
```

Concatenación de cadenas

Con arreglos

```
void concatena(char s[], char t[]) {
  int i = 0, j = 0;     // al principio de s y t
  while (s[i] != '\0')   // busca el final de s
    i++;
  while ((s[i++] = t[j++]) != '\0')  // copia t
    ;
}
```

Concatenación de cadenas

Con apuntadores

```
void concatena(char *s, char *t)   {
  // los apuntadores ya apuntan al lugar correcto
  while (*s)              // busca el final de s
    s++;
  while (*s++ = *t++)                // copia t
    ;
}
```

Concatenación de cadenas

Con llamadas a funciones

```
void concatena(char *s, char *t)   {
  // los apuntadores ya apuntan al lugar correcto
  s += longitud(s);      // busca el final de s

  copia(s, t);                        // copia t

}
```

Comparación de cadenas

Con arreglos

```
int compara(char s[], char t[]) {
  int i = 0;            // al principio de s y t
  while (s[i] == t[i]) // si s y t son iguales
    if (s[i] != '\0')          // si no es nulo
      i++;                              // avanza
    else                       // de otra manera
      return 0;            // s y t son iguales
  return s[i]-t[i];     // s y t son diferentes
}
```

Comparación de cadenas

Con apuntadores

```
int compara(char *s, char *t)   {
  // los apuntadores ya apuntan al lugar correcto
  while (*s == *t)      // si s y t son iguales
    if (*s) {                  // si no es nulo
      s++; t++;                       // avanza
    } else                     // de otra manera
      return 0;            // s y t son iguales
  return *s - *t;       // s y t son diferentes
}
```

Cadenas

Ejercicios

1. Escribe la función `int busca(char s[], int c)` que regresa la posición de la primera aparición de `c` en la cadena `s` (es decir, `s[i] == c`) o -1 si no está. Ahora escribe la función `char* busca(char *s, int c)` que regresa un apuntador a la primera aparición de `c` en la cadena `s` o `NULL` si no está.
2. Escribe la función `void invierte(char s[])` que invierte la cadena `s`. Ahora escribe una versión con apuntadores `void invierte(char *s)`. Ejemplo: `roma -> amor`.
3. Escribe la función `void borra(char s[], int c)` que borra todas las apariciones de `c` en la cadena `s`. Ahora escribe una versión con apuntadores `void borra(char *s, int c)`. Ejemplo: `catarata -> ctrt`.

Arreglos de cadenas

Arreglos de arreglos de caracteres

Imagine que se quiere declarar un arreglo de cadenas que contenga los nombres de los siete días de la semana. Como "miercoles" tiene nueve caracteres, entonces debemos pedir diez caracteres para cada cadena:

```
char dias[7][10] = {"lunes", "martes", "miercoles",
        "jueves", "viernes", "sabado", "domingo"};
```

Esto se vería así como arreglo:

```
dias[0] = "lunes"; dias[1] = "martes";
dias[2] = "miercoles"; ... ; dias[6] = "domingo";
```

Y así en la memoria:

```
"lunes0....martes0...miercoles0jueves0...viernes0..sabado0...
   domingo0.."
```

Arreglos de cadenas

Inconvenientes

Almacenar arreglos de cadenas así tiene serios inconvenientes:

1. En un arreglo de este tipo, todas las cadenas ocupan la misma cantidad de caracteres. Esto quiere decir que cada una debe ser al menos tan larga como la cadena más larga. Por otro lado, las cadenas cortas desperdiciarán mucho espacio.
2. Con frecuencia se desea mover una cadena de un lugar a otro de un arreglo (por ejemplo, para hacer copias o mantenerlas ordenadas). En este caso se deberá procesar cada carácter de la cadena y entre más largas será más lento.

Arreglos de cadenas

Arreglos de apuntadores a caracteres

Una alternativa es declarar un arreglo de apuntadores a caracteres. El arreglo de nuestro ejemplo anterior podría declararse así:

```
char *dias[7] = {"lunes", "martes", "miercoles",
        "jueves", "viernes", "sabado", "domingo"};
```

Esto se interpretaría exactamente igual que un arreglo:

```
dias[0] = "lunes"; dias[1] = "martes";
dias[2] = "miercoles"; ... ; dias[6] = "domingo";
```

Pero en la memoria se vería distinto: `dias[0]` sería un apuntador a la cadena `"lunes"` que ocupa sólo seis caracteres, etc.

Intercambio de cadenas

Con arreglos de caracteres

Si a y b son arreglos de caracteres y queremos intercambiar sus contenidos, no tenemos más remedio que copiarlos de un lado a otro:

```
char a[] = "primero"
char b[] = "segundo";
char t[] = "temporal";

copia(t, a); // copia primero a t
copia(a, b); // copia segundo a a
copia(b, t); // copia primero a b
```

En este caso, las tres cadenas deben ser suficientemente largas.

Intercambio de cadenas

Con apuntadores a caracteres

Si a y b son apuntadores de caracteres y queremos intercambiar las cadenas a las que apuntan, podemos hacer algo mucho más rápido:

```
char *a = "primero"
char *b = "segundo";
char *t;

t = a; // t apunta a primero
a = b; // a apunta a segundo
b = t; // b apunta a primero
```

En este caso, sólo se intercambian los valores de los apuntadores.

Clasificación de caracteres

Funciones de biblioteca

Con frecuencia se necesita saber si un carácter es letra, dígito o alguna otra clasificación especial. Usando `#include <ctype.h>` obtenemos lo siguiente:

```
int isdigit(c); // c es un digito
int isupper(c); // c es una mayuscula
int islower(c); // c es una minuscula
int isalpha(c); // c es una letra
int isalnum(c); // c es letra o digito
int isspace(c); // c es un blanco
```

Además obtenemos:

```
int tolower(c); // convierte c a minuscula
int toupper(c); // convierte c a mayuscula
```

Conversión cadena a decimal

Con arreglos

```
int decimal(char s[]) {
  int n = 0;
  for (int i = 0; '0' <= s[i] && s[i] <= '9'; i++)
    n = 10*n + (s[i] - '0');
  return n;
}
```

Conversión cadena a decimal

Con arreglos y funciones

```
int decimal(char s[]) {
  int n = 0;
  for (int i = 0; isdigit(s[i]); i++)
    n = 10*n + (s[i] - '0');
  return n;
}
```

Conversión cadena a decimal

Con apuntadores y funciones

```
int decimal(char *s)  {
  int n = 0;
  for (; isdigit(*s); s++) // s ya apunta al lugar correcto
    n = 10*n + (*s - '0');
  return n;
}
```

Conversión cadena a decimal

Con apuntadores y funciones

```
int decimal(char *s)  {
  int n = 0;
  while (isdigit(*s))       // s ya apunta al lugar correcto
    n = 10*n + (*s++ - '0');
  return n;
}
```

Cadenas

Ejercicios

1. Escribe funciones `int binario(char *s)` y `int octal(char *s)` que conviertan una cadena escrita en binario u octal a un `int`.
2. Escribe una función `int hexadecimal(char *s)` que convierta una cadena escrita en hexadecimal a un `int`. Los dígitos hexadecimales del 10 al 15 se pueden representar con `abcdef` o con `ABCDEF`.
3. Escribe una función `void adecimal(char *s, int n)` que convierta `n` a una cadena escrita en decimal.
4. Escribe `int abinario(char *s, int n)` y `int aoctal(char *s, int n)` que conviertan `n` a una cadena escrita en binario u octal.
5. Escribe una función `void ahexadecimal(char *s, int n)` que convierta `n` a una cadena escrita en hexadecimal.