Algoritmos y estructuras de datos

Memoria, apuntadores y arreglos

Francisco Javier Zaragoza Martínez

Universidad Autónoma Metropolitana Unidad Azcapotzalco
Departamento de Sistemas

Universidad Autónoma Metropolitana
Casa abierta al tiempo Azcapotzalco

Posgrado en Optimización

5 de abril de 2021

Citas

Alexander Chase

La memoria es aquello con lo que olvidas.

Xu Liu

Nuestra memoria cambia cada vez que la grabamos.

Jonah Keri

La memoria humana es uno de los peores dispositivos de recolección de datos que exista en el mundo.

Donald Knuth

La gente que está más que casualmente interesada en las computadoras debería tener al menos una vaga idea de cómo es el hardware subyacente. De otra manera los programas que escriban serán muy extraños.

Memoria y direcciones

La memoria de una computadora es un arreglo de celdas numeradas consecutivamente. Cada celda contiene un byte (de 8 bits) y está numerada por una dirección (de d bits). En los procesadores modernos d suele ser 32 o 64, es decir, procesadores de 32 o 64 bits. Las direcciones de memoria inician en la 0 y terminan en la $M - 1$ (donde $M \leq 2^d$ es el tamaño de la memoria).

Ejemplo (con $d = 4$ y $M = 2^4$)

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

Interpretación de la memoria

Como enteros de 4 bytes (`int`)

Ejemplo (con $d = 4$ y $M = 2^4$)

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

Los procesadores de Intel son *little endian*: El primer `int` del ejemplo se calcularía como $69 + 74 \times 256 + 69 \times 256^2 + 77 \times 256^3 = 1296386629$:

bytes 0 a 3	bytes 4 a 7	bytes 8 a 11	bytes 12 a 15
1296386629	1397705808	541410336	5062997

Interpretación de la memoria

Como caracteres (char)

Ejemplo (con $d = 4$ y $M = 2^4$)

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

El juego de caracteres estándar de C es ASCII:

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'

Interpretación de la memoria

Un ejemplo más complicado

Suponga que se hicieron las siguientes declaraciones de variables:

```
char a, b, c, d;
int  n, m;
char w, x, y, z;
```

entonces las variables a, b, c y d podrían ocupar las direcciones 0, 1, 2 y 3, las variables n y m las direcciones 4 a 7 y 8 a 11, y las variables w, x, y y z las direcciones 12, 13, 14 y 15.

En este caso, los valores de las variables serían los siguientes:

```
char a = 'E', b = 'J', c = 'E', d = 'M';
int  n = 1397705808,   m = 541410336;
char w = 'U', x = 'A', y = 'M', z = '\0';
```

Operadores de referencia y desreferencia

Operador de referencia

El operador prefijo & obtiene la dirección de inicio de una variable. A esto también se le llama una referencia a la variable. Si declaramos `int n` entonces &n es una referencia a n.

Operador de desreferencia

El operador prefijo * obtiene la variable a la que se hace referencia. A esto también se le llama una desreferencia. En otras palabras, *&n es n.

Direcciones y apuntadores

Direcciones

Las direcciones donde quedan almacenadas las variables en la memoria son determinadas por el compilador y el sistema operativo. Esto nos obliga a usar el operador & para obtener referencias a las variables.

Variables de tipo apuntador

Un apuntador es una variable que puede almacenar una referencia a una variable de tipo específico. Los apuntadores se declaran así:

```
char *s, *t;
int  *p, *q;
```

Toda variable puede contener basura y los apuntadores no son excepción.

Direcciones y apuntadores

Ejemplo de uso de apuntadores

Considere la declaración de variables y apuntadores:

```
int n = 1, m = 2; // inicializados
int *p, *q;       // sin inicializar
```

Si ahora hacemos

```
p = &n; // direccion de n
q = &m; // direccion de m
```

diremos que `p` apunta a `n` y que `q` apunta a `m` (`*p` es `n` y `*q` es `m`).

```
*p = 3; // ahora n vale 3
m = *p; // ahora m vale 3
q = p;  // ahora q apunta a n
*q = 4; // ahora n vale 4
```

Paso de parámetros a funciones

Paso de parámetros por valor

Todas las funciones en C reciben sus parámetros por valor. Esto significa que se hace una copia en una variable local y se trabaja con ella.

Paso de parámetros por referencia

Es evidente que algunas funciones deben poder modificar variables externas (y no solo copias locales de ellas). Por ejemplo, la función de biblioteca `scanf` debe poder leer datos de la entrada para colocarlos en variables del programa. Esto se hace así:

```
int n;
scanf("%d", &n);
```

Observa que `scanf` recibió una referencia a `n` (y no una copia de `n`).

Función de intercambio

Ejemplo (no funciona)

```
void intercambia(int a, int b) {
  int t;
  t = a;
  a = b;
  b = t;
}
```

Cuando llamamos a esa función de esta manera

```
int n = 3, m = 4;
intercambia(n, m);
```

lo que ocurre es que se hacen copias de `n` y `m` en `a` y `b` (ahora `a = 3` y `b = 4`) y se intercambian los valores de `a` y `b` (ahora `a = 4` y `b = 3`), pero `n` y `m` no se modifican.

Función de intercambio

Ejemplo (sí funciona)

```
void intercambia(int *a, int *b) {
  int t;
  t = *a;
  *a = *b;
  *b = t;
}
```

Cuando llamamos a esa función de esta manera

```
int n = 3, m = 4;
intercambia(&n, &m);
```

lo que ocurre es que `a = &n` y `b = &m` (es decir, `a` apunta a `n` y `b` apunta a `m`) y se intercambian los valores de `*a` y `*b` (es decir, de `n` y `m`).

Memoria y arreglos

Representación de arreglos en la memoria

Considere la declaración del arreglo `int a[4]` y el contenido de 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

Como vimos antes, esto se interpreta como cuatro enteros de esta forma:

`a[0]` (0 a 3)	`a[1]` (4 a 7)	`a[2]` (8 a 11)	`a[3]` (12 a 15)
1296386629	1397705808	541410336	5062997

En general, `T a[N]` declara un arreglo de `N` elementos de tipo `T`. Si una variable de tipo `T` requiere `B` bytes para ser almacenada, entonces el arreglo `a` ocupa `N*B` bytes consecutivos.

Dirección de los elementos

Si declaramos `int a[4]` entonces las direcciones de sus elementos son `&a[0]`, `&a[1]`, `&a[2]` y `&a[3]`. También estará definida `&a[4]` aunque el elemento `a[4]` no exista.

El nombre del arreglo

El nombre del arreglo `a` es un sinónimo de la dirección de su primer elemento. De esta manera, hacer `p = &a[0]` es lo mismo que hacer `p = a`.

Apuntadores e índices

Si declaramos `int *p = &a[0]`, entonces `p+1` vale `&a[1]`, `p+2` vale `&a[2]`, etc. En general, si `i` es un `int` entre `0` y el tamaño del arreglo `a`, entonces `p+i` vale `&a[i]`. En otras palabras `p+i` vale `&a[i]` y `*(p+i)` es `a[i]`.

Incremento y decremento

Si `p` apunta a un elemento de un arreglo, entonces `p++` hace que `p` apunte al siguiente elemento, mientras que `p--` hace que `p` apunte al anterior elemento (cuando existan).

Comparación de apuntadores

Si `p` y `q` apuntan a elementos del mismo arreglo, entonces se pueden comparar. El resultado de la comparación es el mismo que el resultado de la comparación de los índices correspondientes.

Mínimo de un arreglo

Con arreglos e índices, regresando el índice

Encuentra el índice del menor valor en el arreglo `int a[]` de `n` elementos.

```
int  minimo(int n, int a[]) {
  int  m = 0;      // el menor es el primero
  for (int i = 1; i < n; i++)
    if (a[i] < a[m])
      m = i;
  return m;
}
```

Mínimo de un arreglo

Con arreglos e índices, regresando un apuntador

Encuentra un apuntador al menor valor en el arreglo `int a[]` de `n` elementos.

```
int* minimo(int n, int a[]) {
  int *m = &a[0]; // el menor es el primero
  for (int i = 1; i < n; i++)
    if (a[i] < *m)
      m = &a[i];
  return m;
}
```

Mínimo de un arreglo

Con aritmética de apuntadores, regresando un apuntador

Encuentra un apuntador al menor valor en el arreglo que empieza en `int *a`.

```
int* minimo(int n, int *a) {
  int *m = a;      // el menor es el primero
  for (int i = 1; i < n; i++)
    if (*(a+i) < *m)
      m = a+i;
  return m;
}
```

Mínimo de un arreglo

Sólo con apuntadores, regresando un apuntador

Encuentra un apuntador al menor valor en el arreglo que empieza en `int *a`.

```
int* minimo(int n, int *a) {
  int *m = a;      // el menor es el primero
  for (int *p = a+1; p < a+n; p++)
    if (*p < *m)
      m = p;
  return m;
}
```

Ejercicios

1. Escribe una función `void dos(int *a, int *b)` que ordene `*a` y `*b` de modo que `*a <= *b`.
2. Escribe una función `void tres(int *a, int *b, int *c)` que ordene `*a`, `*b` y `*c` de modo que `*a <= *b <= *c`.
3. Escribe una función `int* minimo(int *a, int *b)` que regrese un apuntador al mínimo entero en el arreglo que empieza en `a` y acaba justo antes de `b`. Pista: modifica la última versión de `minimo`.
4. Reescribe todas las versiones de `minimo` pero para el máximo.