Práctica 2:
Aplicaciones de los apuntadores

Estructuras de Datos y Algoritmos I

Autor: José Mauricio Matamoros de Maria y Campos

1. Objetivo

Utilizar apuntadores en lenguaje C para acceder a las localidades de memoria tanto de datos primitivos como de arreglos.

2. Material

Se asume que el alumno cuenta con un una computadora con arquitectura Intel x86 o compatible, sistema operativo Linux/Unix, y compilador `gcc` instalado.

3. Instrucciones

1. Lea detenidamente los antecedentes teóricos sobre el uso de apuntadores, su aritmética, y sus aplicaciones en la Sección 4.

2. Realice cada una de las actividades detalladas en la Sección 5 apoyándose en el código de ejemplo del Apéndice A y responda las preguntas de las mismas.

4. Antecedentes

Un apuntador no es sino un tipo de variable especial que almacena una dirección en memoria. Es decir, cuando se crea una variable tipo apuntador se está reservando un espacio en memoria que contiene una dirección física en la memoria donde se pueden encontrar los datos. El compilador se encarga de reservar espacio suficiente para poder direccionar cualquier dirección en memoria dependiendo de la arquitectura para la cual se compila el programa. Así, una variable tipo apuntador puede ser de 8, 16, 32, o 64 bits y siempre será representado como un número entero no signado en notación hexadecimal.

Por ejemplo, considere el código de la Figura 1 compilado para una arquitectura de 16 bits. El apuntador `ptr`, declarado como un apuntador a entero, permite almacenar la dirección en memoria de de la variable `var`, para poder acceder a los datos directamente. A esta operación se le conoce como **indirección** y se realiza con el operador ámpersand (&).

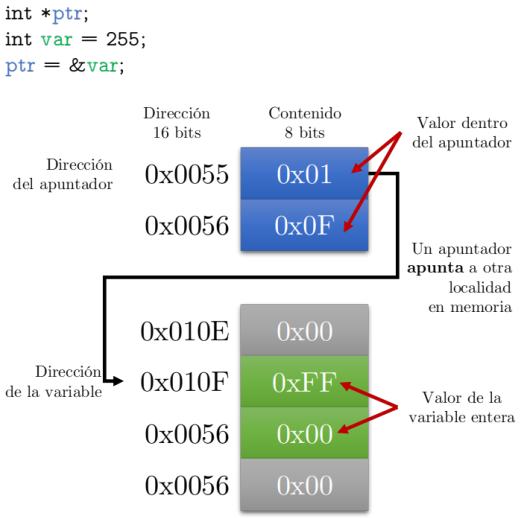

Figura 1: Indirección de los datos de una variable con apuntadores en un programa de 16 bits.

Los apuntadores le da gran poder al lenguaje C. Al almacenar direcciones en memoria, un programa escrito en C no sólo puede acceder directamente a los datos contenidos en cualquier dispositivo de hardware (para el procesador todos los datos se acceden mediante direcciones de memoria), sino también le control total al programador respecto a cómo operan las funciones.

En C, los parámetros que recibe una función se pasan **por copia**; es decir, toda función recibirá una copia de los parámetros que se le suministren. Al recibir copias, las funciones no pueden modificar los datos originales, lo cual no sólo permite realizar llamadas recursivas, sino que añade una capa de seguridad a los programas. Sin embargo, esto haría imposible crear funciones tales como $swap(int\ x, int\ y)$ que intercambia los valores de las variables x y y, pues cualquier modificación a éstas se perdería al ser ejecutada en copias. Aquí es donde entran los apuntadores. Si la función se reescribe como $swap(int\ *x, int\ *y)$, la función no esperará recibir enteros, sino apuntadores a números enteros que serán pasados por copia y no podrán modificarse. Pero esto ya no importa, al conocerse la dirección en memoria donde se almacena la información, esta podrá accederse y modificarse directamente de la siguiente manera:

```
1 void swap(int *x, int *y){
2   int tmp = *x;
3   *x = *y;
4   *y = tmp;
5 }
```

Nótese que con esta notación $*x$ y $*y$ son variables de tipo entero, mientras que x y y son apuntadores a estas variables. Esta notación puede expandirse sin límite, lo que permite crear apuntadores que apuntan a otros apuntadores que a su vez apuntan a otros apuntadores, y así sucesivamente. No obstante, en la práctica es raro encontrar más de 3 niveles de indirección con apuntadores.

Por otro lado, los apuntadores son, también, un elemento fundamental en el manejo de arreglos. Un arreglo no es sino un apuntador que apunta al primer elemento de un segmento de memoria reservado (es por esto que en C se requiere almacenar en otra variable el tamaño del areglo). Esto abre dos posibilidades, la primera es que se puede acceder a los datos apuntados por un apuntador tipado mediante el operador de acceso a arreglos[1] `[]` o bien usando aritmética de apuntadores. Luego entonces las siguientes instrucciones son válidas:

```
1 void foo(){
2   int bar[10];
3   zero(bar, 10);
4   printf("%d\n", bar[0]);
5 }
6
7 void zero(int* a, size_t len){
8   for (size_t i = 0; i < len; ++i)
9     *(a+i) = 0;
10 }
```

Cabe recalcar que cuando se usa aritmética de apuntadores, un incremento de 1 es equivalente a desplazar el apuntador una localidad para el tipo de dato utilizado, por ejemplo 1 byte en apuntadores tipo $char^*$ y 4 bytes en apuntadores tipo $float^*$.

5. Desarrollo de la práctica

Lea cuidadosamente cada una de las actividades propuestas antes de realizar el programa o las modificaciones indicadas.

5.1. Actividad 1: Apuntadores como direcciones en memoria

El programa del Apéndice A.2 genera número pseudo-aleatorio con base en la semilla 69 y posteriormente imprime este número y su dirección en memoria.

Compile y ejecute dicho programa **tres veces** y reporte el número generado y su dirección de memoria.

[1] El nombre en inglés del operador `[]` es *array subscript*.

	Dirección	Número
Ejecución 1		
Ejecución 2		
Ejecución 3		

[2 puntos] ¿Las direcciones coinciden? Explique: _____

A continuación, compile y ejecute el programa del Apéndice A.3 que genera e imprime un arreglo de números pseudo-aleatorios y sus direcciones en memoria. Utilice la información para completar la siguiente tabla:

Índice	Dirección	Número
0		
1		
2		
3		
4		
5		
6		
7		
8		
9		

[2 puntos] Las direcciones en memoria, ¿son contiguas? ¿Están espaciadas a intervalos regulares? Explique: _____

5.2. Actividad 2: Aritmética de apuntadores

El programa del Apéndice A.4 ordena un arreglo de valores tipo `char` generados aleatoriamente utilizando el algoritmo *Bubble Sort* implementado con aritmética de apuntadores.

Modifique el programa anterior para que:

i) Ordene enteros de 32 bits (`int`) en lugar de enteros de 8 bits (`char`).

ii) Utilice arreglos en lugar de aritmética de apuntadores

[1 punto] ¿Qué salida produce el programa modificado?: _____

[3 puntos] Explique qué cambios tuvieron que hacerse a la función `bubble_sort` para que ésta usara arreglos en lugar de aritmética de apuntadores:

5.3. Cadenas

El propósito del programa incompleto del Apéndice A.5 es el de cifrar y descifrar una cadena de texto usando el algoritmo del César, aplicando un corrimiento cíclico de tamaño K a las letras del alfabeto. Nótese cómo en esta ocasión la función `main` del programa recive dos parámetros: un entero y un doble apuntador a caracter. Estos se usan para pasar argumentos al programa ejecutable.

Complete el programa, compílelo y ejecútelo con la línea:

```
./crypto "Hola mundo!!!"
```

[1 punto] ¿Qué salida produce el programa?:

[1 punto] ¿Qué salida produce el programa si se usa un corrimiento de 7 unidades?:

A. Códigos de ejemplo

A.1. Archivo `src/Makefile`

src/Makefile

```
CC       = gcc
# CFLAGS   = -Wall
SILENT   = @
PROGRAMS = random randomv bubble crypto
.PHONY: all clean $(PROGRAMS)

all: $(PROGRAMS)

$(PROGRAMS): %: %.c
  $(SILENT) $(CC) $(CFLAGS) -o $@ $<

clean:
  rm -f *.o
```

A.2. Archivo `src/random.c`

src/random.c

```c
#include <stdlib.h>
#include <stdio.h>
#include <limits.h>

int main(void){
  int num;
  // Set seed to 69
  srand(69);
  // Generate random number
  num = 1 + rand() % 1000;
  // Print random number and its
    address
  printf("num (%p) = %d\n", &num, num);
  return 0;
}
```

A.3. Archivo `src/randomv.c`

src/randomv.c

```c
#include <stdlib.h>
#include <stdio.h>
#include <limits.h>

#define A_SIZE 10


#define A_SIZE 10

int main(void){
  int a[A_SIZE];
  // Set seed to 69
  srand(69);

  // Generate vector of random numbers
  printf("a (%p)= {\n", a);
  for(int i = 0; i < A_SIZE; ++i){
    a[i] = 1 + rand() % 1000;
    // Print number and address
    printf("    %3d (%p)\n", a[i], &(a[i]));
  }

  return 0;
}
```

A.4. Archivo `src/bubble.c`

src/bubble.c

```c
#include <stdlib.h>
#include <stdio.h>
#include <limits.h>

#define A_SIZE 10

// Function prototypes
int main(void);
void swap(char* a, char* b);
void populate(char* array, int seed);
void bubble_sort(char a[A_SIZE], size_t a_size);

// Functions
void populate(char* array, int seed){
  srand(seed);
  for(size_t i = 0; i < A_SIZE; ++i)
    array[i] = 1 + rand() % 256;
}

void swap(char* a, char* b){
  int temp = *a;
  *a = *b; *b = temp;
}

void bubble_sort(char a[A_SIZE], size_t a_size){
  for (int i = a_size-1; i >= 0; --i){
    for (char *cc = a, *bcc = a; cc < (bcc + i); ++cc){
      if (*cc > *(cc+1)) swap(cc, cc+1);
    }
  }
}

// Main
int main(void){
  char a[A_SIZE];

  // Populate the array with random numbers
  populate(a, 69);

  // Sort array
  bubble_sort(a, A_SIZE);

  // Print array
  printf("a:");
  for(size_t i = 0; i < A_SIZE; ++i) printf(" %d", a[i]);
  printf("\n");

  return 0;
}
```

A.5. Archivo `src/crypto.c`

src/crypto.c

```c
#include <stdlib.h>
#include <stdio.h>

#define SHIFT 4

void encrypt(char* str);
void decrypt(char* str);
int main(int argc, char** argv);

void encrypt(char* str){
  // Read until the end of the string
  while(*str != 0){
    // To upper case
    if((*str >= 'a') && (*str <= 'z')) *str += 'A' - 'a';
    // Shift caps and numbers, preserves the rest
    if((*str >= 'A') && (*str <= 'Z')){
      *str = 'A' + (*str - 'A' + SHIFT) % (1 + 'Z' - 'A');
    }
    if((*str >= '0') && (*str <= '9')){
      *str = '0' + (*str - '0' + SHIFT) % (1 + '9' - '0');
    }
    ++str;
  }
}

void decrypt(char* str){
  /* Student's Code Here */
  /* Student's Code Here */
  /* Student's Code Here */
}

int main(int argc, char** argv){
  if(argc < 2){
    fprintf(stderr, "No input string provided!\n");
    fprintf(stderr, "Usage:\n\t%s string\n", argv[0]);
    return -2;
  }
  printf("Original  string: \"%s\"\n", argv[1]);
  encrypt(argv[1]);
  printf("Encrypted string: \"%s\"\n", argv[1]);
  decrypt(argv[1]);
  printf("Decrypted string: \"%s\"\n", argv[1]);
  return 0;
}
```

B. Reporte Escrito

El reporte de la práctica deberá ser entregada en un archivo en formato `PDF` siguiendo las siguientes especificaciones:

- La primera página del documento deberá ser la carátula oficial para prácticas de laboratorio disponible en `lcp02.fi-b.unam.mx/`
- El nombre del documento `PDF` deberá ser `nn-XXXX-L02.pdf`, donde:
  - `nn` es el número de lista del alumno a dos dígitos forzosos (ej. 01, 02, etc.).
  - `XXXX` corresponden a las dos primeras letras del apellido paterno seguidas de la primera letra del apellido materno y la primera letra del nombre, en mayúsculas y evitando cacofonías; es decir, los cuatro primeros caracteres de su RFC o CURP.
- El reporte consiste en un documento de redacción propia donde el alumno explica de forma concisa y a detalle las actividades realizadas en la práctica, y reportando los resultados obtenidos.
- La longitud del reporte no deberá exceder las 3 páginas, sin contar la carátula.
- El reporte deberá seguir todos los lineamientos para documentos escritos establecidos al inicio del curso.
- Todas las referencias deberán estar debidamente citadas.

IMPORTANTE: No se aceptan archivos en otros formatos ni con nombres distintos a los especificados.