Centro de Investigación en Matemáticas, A.C.

Reporte tarea 2 - Programación y algoritmos

Erick Salvador Alvarez Valencia

27 de Agosto de 2017

Índice

1. Introducción

En el presente reporte se presentan textualmente el desarrollo y los resultados de los programas adjuntos a la tarea de la semana 2 de la materia de programación y algoritmos.
A lo largo del reporte se analizará por separado la forma en la que se desarrollaron los programas, incluyendo el pseudocódigo de algunas funciones. A su vez se añadirán los resultados de algunas ejecuciones que se realizaron, y, sus debidas conclusiones.

2. Programa 1: Funciones con punteros

2.1. Desarrollo

El programa se dividió en 4 funciones, todas ellas trabajaron con un puntero a un arreglo de tipo *char* creado en la función *main*.

La primer función se encargó de obtener el tamaño de un tipo pasado por argumento (como entero), tal y como lo haría el operador $sizeof$. Lo que hace la función es crear un arreglo con memoria estática para 2 posiciones de acuerdo al tipo provisto, posteriormente hace la resta de las direcciones de memoria y regresa el resultado. Esto siempre funciona debido a que el arreglo estático reserva memoria en posiciones contiguas de la RAM.

Input: Un entero representando el tipo de dato
Result: El tamaño en bytes del tipo de dato provisto
Function *funcion1 (tipo)*
    arr ← arreglo de tamaño 2 del tipo dado;
    tam ← arr[1] - arr[0];
    return *tam*;

Algorithm 1: Tamaño en bytes de un tipo dado.

La segunda función se encargó de llenar el arreglo pasado como argumento con valores específicos de acuerdo al tipo de dato proporcionado. Para realizar esto anterior, el programa se apoyó en la creación de punteros de datos específicos y un casteo con el arreglo original, lo cual permitió que al hacer un avance en la posición del puntero, dicho avance fuera de acuerdo al tipo. Como ejemplo, en el compilador de gcc, un puntero de tipo *entero* realiza un avance de 4 bytes, mientras que uno de tipo *double* realiza un avance de 8 bytes. Como se hace un llenado con otro puntero entonces dichas acciones se reflejarán en la memoria del arreglo original.

Input: Un entero representando el tipo de dato, un entero representando el tamaño del recorrido, un puntero hacia un arreglo de tipo char

Function *funcion2 (tipo, n, arr)*

```
if tipo == 0 then
    for i ∈ n do
        Llenar arr[i] con valores entre 48 y 122 (repitiéndose).
    end
else if tipo == 1 then
    aux ← puntero a enteros del arreglo original;
    for i ∈ n do
        Llenar aux[i] con valores entre 1 y n.
    end
else if tipo == 2 then
    aux ← puntero a flotantes del arreglo original;
    for i ∈ n do
        Llenar aux[i] con valores entre 0 y 1 con un paso de 1/(n − 1).
    end
else
    aux ← puntero a doubles del arreglo original;
    for i ∈ n do
        Llenar aux[i] con valores entre 0 y 1 con un paso de 1/(n − 1).
    end
```

Algorithm 2: Llenado de un arreglo utilizando punteros.

La tercer función imprime el contenido del arreglo de chars hasta un cierto n utilizando la estrategia de la función anterior, crear punteros del tipo específico para hacer un avance en memoria de acuerdo al tipo.

Input: Un entero representando el tipo de dato, un entero representando el tamaño del recorrido, un puntero hacia un arreglo de tipo char

Function *funcion3 (tipo, n, arr)*

```
if tipo == 0 then
    for i ∈ n do
        Imprimir el valor de arr[i].
    end
else if tipo == 1 then
    aux ← puntero a enteros del arreglo original;
    for i ∈ n do
        Imprimir el valor de aux[i].
    end
else if tipo == 2 then
    aux ← puntero a flotantes del arreglo original;
    for i ∈ n do
        Imprimir el valor de aux[i].
    end
else
    aux ← puntero a doubles del arreglo original;
    for i ∈ n do
        Imprimir el valor de aux[i].
    end
```

Algorithm 3: Impresión de un arreglo utilizando punteros.

La función 4 genera 3 valores aleatoriamente, uno representando a un tipo de dato, otro que está en el rango de $(-3, 3)$ el cual servirá para preguntarle al usuario sobre un desplazamiento de memoria, y el tercero un valor entre $(1, r)$, donde r es el límite de acuerdo al tipo de dato obtenido (char: 1600, int: 400, float: 400, double: 200). Al calcular estos valores se le muestra al usuario el tipo elegido y un valor en hexadecimal el cual representa la dirección de memoria de una posición del arreglo (más específicamente $arr[n]$), posteriormente se le pide al usuario que calcule la el desplazamiento de esa posición q veces, donde q es el valor entre $(-3, 3)$ calculado previamente, si este acierta, se llaman las funciones $f2$ y $f3$, de caso contrario, se detiene la ejecución del programa.

Input: Un puntero hacia un arreglo de tipo char
Result: EL tipo de dato calculado aleatoriamente
Function *funcion4 (arr)*

```
n ← 0;
rty ← número aleatorio entre 0 y 3;
rin ← número aleatorio entre -3 y 3;
if rty == 0 then
    Imprimir "char".;
    n ← número aleatorio entre 1 y 1600;
else if rty == 1 then
    Imprimir "int".;
    n ← número aleatorio entre 1 y 400;
else if rty == 2 then
    Imprimir "float".;
    n ← número aleatorio entre 1 y 400;
else
    Imprimir "double".;
    n ← número aleatorio entre 1 y 200;
esperado ← arr[n]_p + (rin * f1(rty));
Imprimir el tipo y la posición de memoria elegidos.;
Imprimir resp.;
if resp == esperado then
    Imprimir correcto.;
    f2(rty, n, arr);
    f3(rty, n, arr);
else
    Imprimir Incorrecto.;
```

Algorithm 4: Función principal.

Donde $arr[n]_p$ es la posición en memoria de la i-ésima posición del arreglo.

2.2. Pruebas realizadas

Para el presente programa se hicieron pruebas a las funciones por separado, y finalmente, cuando se determinó que trabajaban de manera deseada, se probaron en conjunto con la función cuatro.

Las pruebas consistieron en ejecutar el programa varias veces, analizar el tipo de dato y el valor en hexadecimal arrojados, y en ciertos casos teclear el valor correcto y en otros el incorrecto. Cabe concluir que el programa se comportó de manera deseada en todos los casos de prueba.

2.3. Conslusiones

Como se especificó en la sección anterior, el funcionamiento del programa fue correcto debido al buen tratamiento que se tuvo con los punteros, ya que hay que tener mucho cuidado sobre el acceso a la memoria en los mismos. Una buena manera de hacer esto es creando un puntero del tipo deseado y realizando un casting. Ya que al realizar un avance de puntero, este sabrá cuántas posiciones desplazarse de acuerdo al tipo del mismo, y nosotros no tendremos que lidiar con ese problema.

Para lo anterior también hay que cuidar la cantidad de avances que hacemos con los punteros de tipo diferente a *char*, ya que inicialmente tenemos reservados 1600 bytes, si hiciéramos por ejemplo un avance de esa cantidad con un puntero de tipo entero, después de 400 avances estaríamos accediendo probablemente a memoria que no es nuestra.

3. Programa 2: Procesamiento de texto

3.1. Desarrollo

La finalidad de este programa fue el procesamiento de archivos de texto, en general, tomar un archivo *.txt* y crear varios archivos con 5000 caracteres cada uno. Dichos caracteres pasan por un proceso de filtro, donde solo se aceptan los que son letras (mayúsculas - minúsculas) y números. Los separadores se tratan como un caso especial, si hay varios separadores juntos (espacio, tabulación, salto de línea), solo se procesa uno (se imprime en archivo y se cuenta como caracter más), ya sean espacios, saltos de línea o tabulaciones.

El proceso anterior se define a grandes rasgos en el siguiente pseudocódigo:

```
filesName ← arreglo de tipo char;
c ← char;
cntF ← 0;
cntChar ← 0;
isSep ← false;
Leer el nombre del archivo.;
Abrir el archivo.;
Crear puntero a archivo de salida.;
while Sig caracter != EOF do
    c ← Sig caracter en archivo.;
    if not isValidChar(c) then
        continue.
    if isSeparator(c) then
        if isSep then
            continue.
        isSep ← true;
        Imprime en archivo un espacio.;
        cntChar ← cntChar + 1;
    else
        Imprime en archivo c.;
        cntChar ← cntChar + 1;
        isSep ← false;
    if cntChar eq 5000; then
        Cierra el archivo actual y genera uno nuevo.
end
```

Algorithm 5: Procesamiento de texto.

Como se puede apreciar en el pseudocódigo anterior, el programa hace llamadas a las funciones $isValidChar$ y $isSeparator$ las cuales están definidas en un archivo especial para funciones. Esas funciones, se encargan de tomar un caracter y devolver si es válido o no, para lo cual hacen una comparación mediante el código ASCII.

3.2. Pruebas realizadas

El programa fue probado con los archivos de texto que se encuentran en la página de moodle de la clase 4. Se analizaron algunos de los archivos creados por el programa, y en general, todos cumplen con los requisitos solicitados (sólo contienen a lo mucho 5000 caracteres, sólo se procesan letras y números, etc.).

De la misma forma el programa fue probado con otros archivos, creados especialmente para ver qué hacía con ciertos casos especiales, y unos descargados para analizar el procesamiento con grandes cantidades de texto.

3.3. Conclusiones

Se concluye que el programa trabaja como es deseado, según los casos a los que fue probado. De la misma forma se pueden sugerir una pequeña mejora en el procesamiento del nombre de los archivos de salida, ya que se utiliza un arreglo auxiliar para realizar este proceso, se puede implementar un mejor algoritmo que trabaje con el mismo arreglo de lectura de la imagen. De la misma forma se puede extender el algoritmo para más casos.