Tema 4: Cadenas en profundidad

El siguiente ejercicio está pensado para poner en práctica las técnicas enseñadas en los temas anteriores. En concreto, se centrará en el tema 4, y se pedirá usar las funciones vistas en ese tema.

Enunciado

Se pide implementar un programa que analice los parámetros de entrada del programa pasados por la cabecera del main (int argc, char* argv[]), y que muestre los resultados acordes con el tipo de dato introducido. Está prohibido usar scanf, getchar u otra función de lectura de termina, sólo se leerán los datos introducidos por argc/argv. En concreto, se pide implementar lo siguiente:

Programa principal:

Se pide implementar un programa que reciba por parámetros una lista de datos de un tipo introducido por el usuario. Los datos introducidos son de tipo cadena (se usa la variable "char* argv[]"), habrá que analizar esos datos introducidos, transformarlos al tipo de dato pedido, y mostrarlos por pantalla.

Datos de entrada:

La especificación de datos de entrada en programas de terminal (los usados en clase) suele seguir el siguiente formato (sin espacios, sin los carácteres "<>"):

<Dos guiones><nombre de variable>=<valor>

Por ejemplo el siguiente comando:

$> ./programaPrueba --Tamanio=10

Tiene el siguiente significado: "Ejecuta el programa llamado "programaPrueba" y le pasas por argv el parámetro "--Tamanio=10" . Ese parámetro indicaría a un programa que la variable "tamanio" tiene el valor numérico 10. Se va a seguir este formato de especificación de valores de entrada en esta práctica.

El programa que se implementará en esta práctica recibirá por parámetros la siguiente lista de opciones (sin los carácteres “<” “>”):

--numDatos=número : Éste parámetro sirve para especificar el número de elementos que se introducirán, de un tipo dado.

--tipoDatos=tipo : Éste parámetro sirve para especificar el tipo de datos que se introducirán. Pueden ser de tipo char, int o float.

--listaDatos=<elementos separados por coma> : Éste parámetro indica la lista de valores a cargar dentro del programa, de un tamaño igual al indicado en “numDatos”, y del tipo indicado en “tipoDatos”.

Por ejemplo, el comando:

$> ./parseoCadenas --numDatos=5 --tipoDatos=char –listaDatos=a,b,c,d,e

Indicará que nuestro programa debe cargar 5 elementos de tipo char, que serán “a b c d e”.

Desarrollo de la práctica:

Paso 1: Definición de estructuras y tipos enumerados a usar

Se pide, al menos, definir un tipo de datos enumerado para identificar los tipos de datos con los que se trabajará (char, int, float). Ese tipo de datos se denominará “**tipoDatos_**e”. Se deja libertad para implementar el resto de las estructuras de datos que crea necesarias

Paso 2: Crear funciones que permitan acceder a los parámetros leídos y sus valores.

Se necesitan funciones que recorran el array de argumentos de entrada pasados por cabecera del main, y parseen los datos introducidos. Se pide crear las siguientes funciones (no se pueden variar los parámetros de cabecera ni retorno):

int getIntArgValue(char* arg, int argc, char* argv[]) :

Esta función devolverá el valor integer asignado a uno de los argumentos de entrada. Los parámetros de entrada son:

arg: Cadena de caracteres con el nombre del argumento cuyo valor se desea buscar

argc: Tamaño de la lista de argumentos recibidos por main

argv[]: Array de cadenas que contiene los argumentos recibidos por cabecera de main.

Tomando como ejemplo el comando usado anteriormente (--numDatos=5), un ejemplo de invocación desde el "main" sería el siguiente:

int numDatos= getIntArgValue("--numDatos", argc,argv);

Después de invocar la anterior función, la variable "numDatos" debería valer "5".

tipoDatos_e getTDatosArgValue (char* arg, int argc, char* argv[]) :

Esta función devolverá el valor enumerado "tipoDatos_e" asignado a uno de los argumentos de entrada. Los parámetros de entrada son:

arg: Cadena de caracteres con el nombre del argumento cuyo valor se desea buscar

argc: Tamaño de la lista de argumentos recibidos por main

argv[]: Array de cadenas que contiene los argumentos recibidos por cabecera de main.

Tomando como ejemplo el comando usado anteriormente (-- tipoDatos=char), un ejemplo de invocación desde el "main" sería el siguiente:

tipoDatos_e tipoDatos= getTDatosArgValue ("--tipoDatos", argc,argv);

Después de invocar la anterior función, la variable "numDatos" debería valer el equivalente enumerado "char".

char* getStrArgValue(char* arg, int argc, char argv)**

Esta función devolverá la cadena de caracteres separados por comas asignado a uno de los argumentos de entrada. Los parámetros de entrada son:

arg: Cadena de caracteres con el nombre del argumento cuyo valor se desea buscar

argc: Tamaño de la lista de argumentos recibidos por main

argv[]: Array de cadenas que contiene los argumentos recibidos por cabecera de main.

Tomando como ejemplo el comando usado anteriormente (--listaDatos=a,b,c,d,e), un ejemplo de invocación desde el "main" sería el siguiente:

char* listaDatos = getStrArgValue("--listaDatos", argc,argv);

Después de invocar la anterior función,el puntero "listaDatos" debería apuntar a la siguiente cadena : "a,b,c,d,e" (acabada en '\0')

Paso 3: Mostrar los valores de la lista de datos introducidos

Una vez cargados los parámetros, hay que parsear la lista de datos al tipo elegido (char, int ó float), y mostrarlos por pantalla. Para ello se pide implementar las siguientes funciones:

void printDatosChar(char* strDatos,int numDatos)

Esta función imprimirá los "numDatos" valores en formato "char" almacenados en "strDatos". Los parámetros de entrada son los siguientes:

strDatos: Cadena de caracteres en rango "a-z, A-Z, 0-9" acabados en '\0', separados por comas.

numDatos: El número de caracteres separados por comas que hay en strDatos.

Tomando como ejemplo los datos y variables creadas anteriormente, una llamada a esta función podría ser la siguiente:

printDatosChar(listaDatos,5)

Y escribirá por pantalla la siguiente salida:

“Dato 0: a”

“Dato 0: b”

“Dato 0: c”

“Dato 0: d”

“Dato 0: e”

void printDatosInt(char* strDatos,int numDatos)

Esta función imprimirá los “numDatos” valores en formato “int” almacenados en “strDatos”, separados por comas. Los números almacenados pueden ser de tamaño variable. Los parámetros de entrada son los siguientes:

strDatos: Cadena de caracteres en rango “0-9” acabados en ‘\0’, separados por comas.

numDatos: El número de caracteres separados por comas que hay en strDatos.

Suponiendo que el puntero a char “listaDatos” contenga la cadena de caracteres “10,225,3,1456,2”, una llamada a esta función podría ser la siguiente:

printDatosInt(listaDatos,5)

Y escribirá por pantalla la siguiente salida:

“Dato 0: 10”

“Dato 0: 225”

“Dato 0: 3”

“Dato 0: 1456”

“Dato 0: 2”

void printDatosFloat(char* strDatos,int numDatos)

Esta función imprimirá los “numDatos” valores en formato “float” almacenados en “strDatos”, separados por comas. Los números almacenados pueden ser de tamaño variable. Los parámetros de entrada son los siguientes:

strDatos: Cadena de caracteres en rango “0-9 . ” acabados en ‘\0’, separados por comas. (ojo, las comas son para separar números, los “puntos” indican el formato “coma flotante” numérico)

numDatos: El número de caracteres separados por comas que hay en strDatos.

Suponiendo que el puntero a char “listaDatos” contenga la cadena de caracteres “10.5,225.32,3.8,1456.956,2.0”, una llamada a esta función podría ser la siguiente:

printDatosChar(listaDatos,5)

Y escribirá por pantalla la siguiente salida:

“Dato 0: 10.500”

“Dato 0: 225.320”

“Dato 0: 3.800”

“Dato 0: 1456.956”

“Dato 0: 2.000”

Paso 4: Programa “main”

Una vez se tienen las anteriores funciones testeadas, se pasaría a implementar un programa “main” que realizara las laboraes pedidas:

- Recibir los parámetros indicados por argumentos de entrada en la cabecera de la función main
- Extraer el número de elementos de la lista de datos
- Extraer el tipo de los datos
- Extraer la lista de datos
- En función del tipo de datos, llamar al método adecuado para mostrarlos por pantalla.

Evaluación

Se tendrá en cuenta la calidad del código entregado:

Errores de ejecución

Detección de errores en parámetros de entrada

Uso de memoria dinámica de forma adecuada donde sea necesario

Liberación de memoria dinámica (memory leaks)

Parseo de cadenas (uso de funciones vistas en clase)

Parseo de cadenas a tipos de datos

No está permitido compartir código entre alumnos. Lo entregado debe ser código original creado por el alumno. Se permite consultar documentación de funciones y ejemplos de uso, pero no se permite copia de código. En caso de detección de copias, queda a disposición del profesor decidir el nivel de parecido entre códigos entregados.

Se permite la realización de los ejercicios en grupos de dos personas. En este caso , sólo se podrá compartir código entre los miembros del mismo grupo.