TRABAJO PRACTICO N:2

TEMA: Punteros. Arreglos dinámicos. Relación entre puntero y arreglos. Arreglos de punteros. Métodos: procedimientos y funciones, invocación de métodos desde otra función. Puntero a funciones. Métodos como parámetros. Argumentos predeterminados

Objetivos:

* Aprender a crear y utilizar arreglos dinámicos.
* Comprender la vinculación que existe entre el nombre o identificador de un arreglo y los punteros
* Aprender a construir programas de manera modular, a partir de piezas pequeñas llamadas métodos.
* Entender los mecanismos que tienen el lenguaje para pasar información entre funciones.
* Comprender la diferencia que existe entre procedimiento y funciones, y su implementación en C.
* Aprender a usar argumentos predeterminados

LENGUAJE: C/ C++

VERCTORES DINAMICOS

Los vectores dinámicos son los que se crean en tiempo de ejecución, a diferencia de aquellos que se crean en tiempo de compilación. Tanto en uno como en otro caso, la reserva de memoria de sistema que se realiza para ellos es físicamente contigua, es decir hecha la reserva de memoria, esta no puede ser físicamente extendida o liberada parcialmente. Por lo tanto si necesito ampliar la posibilidad de almacenamiento de un arreglo dinámico o estático, se deberá crear otro más amplio en espacio que el anterior y transferir los datos que aquel tenía al nuevo arreglo.

En el lenguaje en C los arreglos dinámicos se crean usando:

1. un método, llamado void \* malloc(size-t size )
2. Un operador , llamado new

METODO

El prototipo de malloc está en stdlib.h

**<stdlib.h>**(funciones para conversión de números y reserva dinámica de memoria: atof, atoi, \_itoa, malloc, free, etc)

La función malloc ( ) devuelve un puntero al primer byte de la colección de byte reservados según el tipo de dato requerido. Si hubiese insuficiente cantidad de memoria para cubrir los requerimientos, malloc ( ) retorna un puntero nulo (null). Es importante verificar cual es el estado de este puntero después de que la instrucción que contiene la invocación al método malloc ( ) se haya ejecutado.

Para una variable primitiva o simple: int, char, float, doublé, la sintaxis es:

Void \*malloc ( size-t numero \_byte)

Tener en cuenta que no es necesario realizar un casteo entre el tipo de dato que retorna la función malloc y el tipo de dato asociado al puntero que señala la memoria reservada por la gestión del método malloc. Esto se debe a que malloc retorna un puntero de tipo void, el cual puede ser asignado a cualquier puntero sin importar el tipo del puntero con el cual ha sido declarado, la conversión la realiza el compilador automáticamente.

En el caso de los arreglos, se utiliza el método sizeof ( ) para determinar el tamaño de la reserva de memoria, para lo cual se hace:

int \*ip;

ip = (int \*) malloc(100 \* sizeof(int) ); // reserve de 200 byte de memoria físicamente consecutiva

float \*p = (float \*) malloc(N \* sizeof(float) ); // crea reserve de memoria de 4 x N byte

Ejemplo1:

El siguiente método escribe al revés una cadena ingresada por teclado

#include “stdio.h”

# include “stdlib.h”

# include “string.h”

main ( ) {

char \*s;

register int I;

s = malloc ( 80);

if ( !s) { printf( “ no se gestiono la memoria”);

return 1;

}

gets (s );

for ( t = strlen ( s) -1; t> = 0; t--) printf ( “%c”, s [t]);

free ( s);

return 0;

}

Ejemplo2:

El siguiente método reserva espacio de memoria dinámica para crear un arreglo de 10 enteros.

#include “stdio.h”

# include “stdlib.h”

int \* function (int can ){ int \* ptro;

ptro = ( int \*)malloc ( can \* sizeof (dato)); ) return ptro;}

void carga-arreglo ( int t [ ], int can) // el compilador cree que es un arreglo

{int I;

for ( I =0; I < can; i++)

scanf ( “%d”, &t);

}

void muestra\_array ( int a[ ], int tam )

{ int iI;

for ( i =0; i< tam; i++) printf ( “%d”, a [ i]);}

int main ( ) {

int can, \*p;

printf ( “ ingresar cantidad de elementos”);

scanf ( “ % d”, &can);

p = función (can);

if ( !p) return 1;

carga-arreglo ( p, can );

muestra\_array ( p,can);

free ( p);

return 0;

}

OPERADOR

Otra forma de crear un arreglo en tiempo de ejecución es por medio del operador new. El operador new crea un objeto y el operador delete lo borra (en el sentido de liberación de memoria).

Su sintaxis es más sencilla, puesto que new no es un procedimiento sino un operador del C++ que está en el entorno del programa, no hay que utilizar una librería determinada para acceder a su código. El operador new devuelve la dirección de memoria donde comienza el almacenamiento dinámico creado, en caso de no poder hacer la reserva de memoria complete, new devuelve null; por lo tanto hay que averiguar si el espacio de memoria necesario se pudo gestionar antes de usarlo.

La sintaxis para este tipo de operador es:

puntero= new tipo-de-dato // crea un tipo primitivo

puntero = new (tipo de dato) // forma funcional

delete puntero

delete [ ] puntero // para un arreglo unidimensional

Ejemplos de uso del operador new:

int \*t = new int; // asignación para un entero

int \* ptro = new int [5 ]; // crea un arreglo dinámico de 5 elementos de tipo entero

char \* ptro2 = new char [ N]; // crea un arreglo de tipo char de tamaño N

Las dimensiones para crear un arreglo bidimensional pueden ser expresiones:

double \* pp,

int fila =5;

int colum = 2;

double (\* pp) [ colum] =new double [fila \* colum];

la instrucción anterior crea reserva de memoria dinámica para un arreglo bidimensional de tamaño fila por colum. Para acceder a los elementos de esta table se puede usar la sintaxis pp [0][1], pp [0][2], etc

Si la especificación del tipo de dato para el que queremos crear memoria dinámica es complicada, hay que utilizar la forma funcional del operador new, se usan los paréntesis para forzar el orden en el que debe realizarse la interpretación del tipo.

Por ejemplo para la siguiente sintaxis.

int (\* \*pfn )( );

pfn = new ( int (\* [ 3]) ( ) ); \ \* se asigna memoria para un arreglo de 3 punteros a funciones que devuelven enteros y que no aceptan parámetros \* \

C++ también permite inicializar las variables que se crean dinámicamente, para ello hay que realizar lo siguiente:

puntero = new tipo-de-dato ( inicializadores)

por ejmplo:

float \* p ;

p = new float ( 2.5);

Ejemplo:

Implementa un programa en C++ que permita crear un puntero a un tipo de dato char y que lo inicialize con el nombre de una persona entre comillas doble, en la misma instrucción. Este nombre será pasado por parámetro a un método el cual creará un arreglo dinámico con el operador new, para albergar dicho nombre.

# include “iostream.h”

# include “string.h”

# include “assert.h” // código para el método assert

void crear\_vector\_dinamico ( char \*nom)

{char \*vector;

vector= new char [ strleng (( nombre ) +1)];

assert (vector !=0); // termina si no se asigna memoria

strcpy ( vector, nombre); // copia contenido de cadenas

cout << vector;

}

int main ( )

{ char nombre [ ] = “Oscar”;

Crear\_vector\_dinamico ( nombre);

Return 0;

}

Assert es una macro, sirve para determinar si se signo la memoria para vector. De no ser así el programa termina con un mensaje de error que indicó la condición probada, el número de línea donde sucedió la condición y el archivo donde se encuentra dicha condición.

Punteros a Funciones. Ejemplo.

//# include ….

h(int x) {return 4;}

int g (int n, int (\*f) (int))

{int u = f(3); // también válido (\*f)(3)

return u;

}

main ( ) {

int k=g(5, h); //también válido “g(5,&h);”

//……..}

EJERCICIOS

1. Desarrollar un programa que permita declarar un arreglo dinámico de 10 elementos de tipo entero. Se desea recorrerlo y mostrar por pantalla los elementos múltiplos del tamaño.
2. Desarrollar un programa que permita declarar un arreglo dinámico de 20 elementos de tipo entero. Se desea generar otro con los contenidos del arreglo origen cuyos elementos sean impares positivos. Mostrar el contenido de los dos arreglos por pantalla.
3. Desarrollar un programa que permita declarar un arreglo dinámico de 20 elementos de tipo entero con repetición. Se desea generar otro sin las repeticiones del arreglo origen. Mostrar ambos arreglos por pantalla.
4. Desarrollar un programa que permita declarar un arreglo dinámico de 15 elementos, el contenido del arreglo son caracteres ingresados desde el teclado. Se desea recorrerlo y mostrar por pantalla solamente las letras minúsculas.
5. Desarrollar un programa que permita mediante un método ordenar los elementos de un arreglo dinámico de tamaño N, se desea mostrar por pantalla.
6. Se desea desarrollar un programa que permita declarar una tabla de 3 filas por 3 columnas de números enteros. Mediante un método se desea recorrerla y generar un arreglo con los elementos de la fila F1, este valor se pasará como parámetro. Mostrar por pantalla desde el programa principal los elementos del arreglo generado.
7. Se desea desarrollar un programa que permita declarar una tabla de 3 filas por 3 columnas de números enteros. Se desea recorrerla mediante un procedimiento y generar un arreglo con los elementos que están dispuestos en la diagonal secundaria. El arreglo se muestra desde el programa principal.
8. Se desea desarrollar un programa que permita declarar una tabla de 4 filas por 4 columnas de números enteros. Se desea recorrerla mediante un método para detectar el mayor elemento y el menor, esta información será devuelta al programa principal a través de una instrucción y desde allí mostrada por pantalla.
9. Se desea desarrollar un programa que permita declarar una tabla de N xN de números reales de simple precisión.

Se desea desarrollar métodos para:

1. Seleccionar el tamaño NxN
2. Cargar la tabla desde el teclado
3. Mostrar por pantalla los elementos de la tabla
4. Detectar el menor valor y generar un arreglo con los elementos que estén sobre su columna.
5. Se desea desarrollar un programa que permita declarar una tabla de N xN de caracteres. Se desea desarrollar métodos para:
6. Cargar la tabla, se admiten repeticiones
7. Recorrer la tabla
8. Generar otra tabla en la cual se hallan reemplazado las repeticiones por ceros o blancos o cualquier marca de su elección, mediante la cual se pueda identificar el lugar en el cual estaba la repetición.

Mostrar por pantalla toda la información necesaria para comprender el proceso que se hace sobre los datos.

1. Implementar un programa que permita mediante una función calcular el volumen de una caja. El método dispone de argumentos predeterminados para calcular el volumen de una caja de 1x1x1.

int calcula\_volumen ( int largo =1, int ancho =1,int alto = 1); /\*prototipo de una

función con parámetros

predeterminados\*/

Se desea invocar el método de la siguiente manera:

1. Llamar a la función sin indicación de parámetros.

vol = calcula\_volumen ( );

1. Llamar a la función con valor para largo tomado desde el teclado en el main.

vol = calcula\_volumen ( a);

1. Llamar a la función con valor para largo y ancho tomado desde el teclado en el main.

vol = calcula\_volumen ( a,b);// el parámetro alto es el predeterminado

1. Llamar a la función pasando los 3 parámetros con valores tomados desde el teclado desde el main.

vol = calcula\_volumen (a, b, c );

***Nota****: Las llamadas a funciones pueden pasar un valor particular para un argumento. El programador puede especificar que un parámetro es un argumento predeterminado, con lo cual indica un valor predeterminado para él. Cuando se omite un argumento predeterminado en una llamada de función, el compilador inserta automáticamente el valor predeterminado de dicho argumento, el cual se pasa en el momento de la llamada.*

*Los argumentos predeterminados deben ser los de la derecha (últimos en la lista) de todos los parámetros que se disponga en un método. Al invocar un método con dos o más parámetros predeterminados, si uno de los parámetros omitidos no es el de más a la derecha de la lista de parámetros, todos los que se encuentren a la derecha de dicho parámetro también deben omitirse.*

*Los parámetros predeterminados deben especificar con la primera ocurrencia del nombre de la función (generalmente en el prototipo). Los valores predeterminados pueden ser constantes, variables globales o invocaciones a funciones.*

*Es un error de sintaxis especificar e intentar usar un argumento predeterminado que*

*no sea el de más a la derecha (último), sin tomar los predeterminados de todos los argumentos a su derecha.*

1. Inventar un programa en C/C++ donde usted deba usar los siguientes elementos de código:

(\*ptr) [ ] // ptr es un puntero a un arreglo

Por ejemplo usando la notación anterior para tipos permitidos, logramos las siguientes construcciones:

int \* (\*ptr) [ ] // ptr es un puntero a un arreglo de punteros a variables enteras

int (\*ptr)[ ] //ptr es un puntero a un arreglo de variables de tipo int

1. Se desea inicializar un arreglo de punteros a cadenas como se muestra a continuación:

Char \*nombre\_mes [12] = { “Enero”, “Febrero”, “Marzo”, “Abril”, “Mayo”, “Junio”, “Julio”, “Agosto”, “Septiembre”, “Octubre”, “Noviembre”, “Diciembre”,};

Se desea mostrar por pantalla los meses largos (31 días) y los meses cortos (30 días), por separado.

1. Una máquina herramienta lee temperaturas provenientes de un sensor. Se desea implementar un método (A) que reciba como parámetro una estructura de dos miembros, uno de ellos representa la temperatura máxima y el otro miembro la mínima. Este método (A) invocará otro método (B) llamado leer\_ temperatura, cuyo único parámetro es la temperatura que proviene del sensor. Se desea que el método (A) actualice la información de los miembros de la estructura.

Por ejemplo:

……………………..

void MetodoA ( temperatura \* t)

{ float actual;

Leer\_temp ( actual);

if ( actual > t-> alta)

t->alta = actual;

else if ( actual < t-> baja)

t-> baja = actual;

}

………………………

1. Implementar un programa que permita declarar un punto a un método. El método recibirá dos parámetros que representan el extremo inferior y el extremo superior de un intervalo de letras que se generará desde dentro del método. Deberá asegurar que los parámetros representen ambos información válida (representan letras minúsculas o letras mayúsculas).

Mostrar el contenido del arreglo generado desde el método en el main.

Por ejemplo el formato del método puede ser:

char ( \*función) ( char inf, char sup);

1. Crear un enunciado para las siguientes declaraciones y compilar los códigos que respondan a ellos.
2. Void \*función ( int t, int n);
3. Unsigned \* función ( int t, int + a[ ], unsigned n);
4. Diseñar una función A que permita realizar la suma de n términos. Este método A tienen dos parámetros, int n que representa el número de términos de la suma, y otro parámetro que es un método ( B o C). El método B, recibe un solo parámetro k que representa el número de términos a generar y calcula lo siguiente: inverso (k) = 1.0/ k (k =1, 2, 3…)

El método C, recibe un solo parámetro k que representa el número de términos a generar y calcula lo siguiente: cuadrado (k) = k \* k (k =1, 2, 3…)

En C:

…….

double inverso ( int k);

double cuadrado ( int k);

doublé funcsuma ( int n, doublé ( +f) ( int k);

…….

Main ( )

{ …….

….. funcsuma ( n, inverso);

……..

…….funcsuma ( n, cuadrado);

…..

}

…….

….