Nombre completo: **Posadas Flores Armin**

Código de la materia: **IPV 6576**

Nombre de la carrera: **Ingeniería en Programación de Videojuegos**

Fecha de elaboración: **09/03/2024**

Trabajo: **Resumen del capítulo 1 del libro**

Introduccion al libro

Para empezar el libro nos da a entender desde su introduccion que las lecciones que se encuentran dentro son mas para optimizar codigo de la mejor forma posible, poniendonos como ejemplo el problema de selección que es determinar el numero mas grande de un grupo N del cual si bien la solucion es sencilla el metodo que se suele usar tarda horas a diferencia de lo que viene en el libro pero con la desventaja de que puede ser inpractico.

Problemas matematicos

En la programacion es importante conocer sobre matematicas y el libro nos enseña cuales son las formulas que mas se necesitan dentro del campo como lo son:

Exponentes: lo basico si dos bases con exponentes distintos se estan multiplicando los exponentes se suman, si se dividen es resta, si se encuentran entre parentesis se multiplican y si son iguales no cambian.

Logaritmos: Como sabemos es el exponente al cual se necesita elevar una cantidad positiva para obtener como resultado un cierto número, por ejemplo: 7^2=49 pero si queremos encontrar el exponente usamos log(7)*49=2 cabe resaltar que en programacion todos los logaritmos tienen como base un 2 a no ser que se diga lo contrario.

Series: Es la generalización de la noción de suma, aplicada a los infinitos términos de una sucesión, su formula siendo

$\sum_{i=0}^{N} 2^i = 2^{N+1} - 1$ y la otra siendo $\sum_{i=0}^{N} A^i = \frac{A^{N+1} - 1}{A - 1}$ y en caso de que suceda el caso de 0<A<1 se usa este $\sum_{i=0}^{N} A^i \leq \frac{1}{1 - A}$.

Aritmetica modular: a veces cuando dividimos dos numeros lo unico que quedremos sera el residuo y no el coeciente asi que para eso esta la aritmetica por ejemplo si tenemos 13/5 para sacar unicamente el residuo usamos 13 mod 5 que nos termina dando un 3, cabe resaltar que mod no hace nada por si mismo por lo que si necesita de almenos dos numeros.

Las palabras con P:

Prueba por induccion: esta se divide en dos partes la primera se encarga de proveer las base del caso, por ejemplo tenemos el caso de una serie numerica que va del F0=1 a Fi=Fi-1 +Fi-2 y tenemos que satisfacer la formula a Fi < (5/3)^i cuando i >1 para todo esto agarramos cada numero de la serie en orden, reemplazando i en las ecuaciones y asegurandonos que Fi < (5/3)^i, por ejemplo F1=1<(5/3), F2=2<(25/9), en ambos casos la base se cumple ahora tenemos que comprovar el teorema, para ello necesitamos comprovar que Fk+1 < (5/3)k+1, para ello podemos usar la hipotesis inductiva donde

$$\begin{aligned} F_{k+1} &< (5/3)^k + (5/3)^{k-1} \\ &< (3/5)(5/3)^{k+1} + (3/5)^2(5/3)^{k+1} \\ &< (3/5)(5/3)^{k+1} + (9/25)(5/3)^{k+1} \end{aligned}$$ el cual simplificado pasa a ser

$$\begin{aligned} F_{k+1} &< (3/5 + 9/25)(5/3)^{k+1} \\ &< (24/25)(5/3)^{k+1} \\ &< (5/3)^{k+1} \end{aligned}$$

Prueba de Contraejemplos: en este caso el enunciado que dice que Fk ≤ k2 es falso pues F11 = 144 > 112.

Prueba de Contradiccion: procede asumiendo que el teorema es falso y que alguna propiedad tambien es falsa, esto suele ocurrir cuando hay una cantidad infinita de numeros primos

Introduccion breve a la recursion

Funciones que se definen por sus propios terminos se les conoce como recursivas y el lenguaje de programacion C++ permite a las funciones a ser recursivas, claro no todas las matematicas recursivas son eficases o correctas pero el lenguaje ayuda a que la operación sea mas expresiva en menos codigo, por ejemplo si queremos llamar a un valor de 4 en una funcion f(4) entonces el lenguaje pedida la funcion de f(3) y si no la encuentra se ira al f(2) y asi hasta encontrar una funcion completa aun si esta llegara a los numeros negativos asi que lo que se hace es realizar una sola ecuacion, de preferencia f(1) o f(0) para que el programa automaticamente se encarge de hacer el resto si mandamos a llamar a una funcion mayor a 1 o 0.

Imprimiendo numeros

Cunado se desee imprimir un numero en la terminal o cualquier cosa en el que la vayamos a usar utilizamos “printOut(n)” con n siendo el numero de digitos que hay en la consola, por ejemplo si tenemos 76234 tendriamos que poner printOut(5) ya que posee cinco numeros ese valor claro tambien hay que especificar en que parte de la ecuacion va pues si va en una ecuacion con una posibilidad de 10 numeros tendriamos que poner algo asi 0<n<10.

Classes del C++

Todas las estructuras de datos seran objetos que almacenaran datos y proveeran funciones que manipulan la coleccion

Syntaxis basica de una clase

Una clase en C++ usualmente se conforma de miembros como datos o miembros de funciones en este ultimo caso tenemos que cada instancia de una clase es un objeto cada objeto tiene componentes de datos especificados en una clase (a no ser que los datos sean estaticos) y los miembros de una funcion se usan para actuar sobre una funcion, por ejemplo en una clase de intCell cada objeto tiene un miembro nombrado storedvalue mientras que el resto solo son metodos y de esos metodos dos son “write” y “read” mientras que los otros dos son “public” y “private” los cuales determinan la visibilidad de un miembro de la clase dentro de los metodos util para ocultar informacion que tal vez sea importante para lo que sea que se vaya a usar.

Separacion de interfaz e implementacion

Comandos de preprocesado

La interfaz usualmente se encuentra en un archivo que lleve .h y que debe ser mencionado en el archivo principal con un #include estos archivos suelen llevar .cpp, .cc o .C y cada miembro de su funcion debe ser capaz de identificar la clase a la que pertenece o si no asumiran que la funcion es global, tambien es importante que cada miembro de la funcion concuerde con la clase dentro del interfaz incluso si forma parte de un accessor o un

mutator por ultimo es importante que cuando se vaya a declarar un objeto sea por un zero parameter o one parameter

IntCell obj1; // Zero parameter constructor

IntCell obj2(12); // One parameter constructor

De lo contrario sera incorrecto como estos ejemplos

IntCell obj3 = 37; -> esto debido a que el constructor es explicito

IntCell obj4( ); -> y esto debido a que es mas una declaration de la funcion que un constructor.

Vectores y strings

Los vectores se encargan de conocer el tamaño de los arrays y strings, mientras que los strings pueden ser comparados con un == o < por eso catalogan a estos como objetos de primera clase, son faciles de usar pues incluso un vector es capaz de guarda 100 cuadros perfectos e imprimirlos.

Detalles del C++

Punteros

La variable de punteros son variables que almacenan la direccion de donde pertenece otro objeto, es usualmente utilizado para almacenar una lista de objetos.

Cuando se va a declarar un puntero se utiliza "*" al inicio de una declaracion (ejemplo: intCell: *m;) pero es importante que el puntero sea utilizado una vez declarado pues de lo contrario crasheara el programa al compilarlo debido a que como un puntero accesa a la memoria al declararlo pero no usarlo el puntero busca la localizacion de una memoria que no existe por lo que es importante darle un valor inicial al inicio del codigo. (ejemplo)

1-IntCell *m;

2-

3-m = new IntCell{ 0 };

Creacion dinamica de objetos

In C++ existe la funcion “new” el cual regresa un puntero a un objeto recien creado, la mejor forma de escribirlo es:

m = new IntCell;

Colección y eliminacion de basura

Cuando un objeto deja de ser referenciado es puesto de forma automatica a una colection de basura, pero cuando un objeto que fue asignado por “new” ya no es referenciada la operación “delete” debe ser aplicada al objeto pues de lo contrario la memoria usada se pierde, esto se le comone como una fuga de memoria (memory leak) por lo que la mejor forma de evitar estos problemas es no usar “new” cuando se puede usar una variable automatica en su lugar.

Asignacion y comparacion de punteros

Estos se basan en los valores del puntero lo que significa la direccion de la memoria que guarda, por ende dos punteros donde las variables son iguales si estas apuntan al mismo objeto si no no son iguales aun si los objetos a los que apuntan lo son.

Acceder a miembros de un objeto a travez de un puntero

Si la variable de un puntero apunta a un tipo de clase entonces se podra cceder a un miembro de la clase a traves del operador “->”

Lvalues, Rvalues y referencias

Lvalue es una expresion que identifica un objeto no temporal mientras que Rvalue es una expresion que identifica al objeto temporal o es un valor que no esta asociada a ningun objeto, por ejemplo tenemos que:

```
vector<string> arr( 3 );
const int x = 2;
int y;
int z = x + y;
string str = "foo";
vector<string> *ptr = &arr;
```

donde arr, str, arr[x], &x, y, z, ptr, *ptr, (*ptr)[x] son valores Lvalue debido a que si tienen nombre esas variables son Lvalue aun si no son modificables, mientras que "foo", x+y, str.substr(0,1) son valores Rvalue por que sus valores son temporales (en el caso de x o y son valores que se van a usar para z por ende son considerados como Rvalues para este caso).

Parametros “Passing”

En lenguajes de programacion todos los parametros pasan por la funcion “call-by-value”(1) mientras que el argumento actual es copiado a un parametro formal pero cuando se trata de C++ este lenguaje a pasado por varias versiones de dicho argumento.

La razon por la que call-by-value no es suficiente como el unico mecanismo del parametro passing en c++ se representa por el siguiente ejemplo:

```
double average( double a, double b );
void swap( double a, double b );
string randomItem( vector<string> arr );
```

En donde call-by-value copia x en a, y en b, para luego ejecutar el codigo para la funcion “average” donde se especifica mas a fondo despues pero presumiendo que x o y sean variables inaccesibles para average esta garantizaro que cuando se regrese a average x o y no cambien en lo absoluto lo que demuestra que call-by-value no podria servir como un swap.

Return Passing

En C++ hay varios mecanismos de regresar de una funcion la mas directa siendo “return-by-value” mostrado en el siguiente ejemplo:

```
double average( double a, double b );
LargeType randomItem( const vector<LargeType> & arr );
vector<int> partialSum( const vector<int> & arr );
```

lo que hace es que todas las funciones regresan un objeto de un tipo apropiado que puede ser usado a llamar y en todos los casos termina siendo un “Rvalue” como sea “randomItem” y “partialSum” pueden llegar a ser incorrectos sl regresar una expresion de

tipo “Lvalue” o elementos copiados, para ello se puede usar “auto &” para evitar una copia inmediata.

std::swap y std::move

La funcion std::swap es muy obvia y se encarga de agarrar dos valores que esten seterminados como inputs e intercambiarloes entre ellos, mientras que la funcion std::move convierte cualquier valor “Lvalue” a un “Rvalue” y viceversa.

Destructor

El destructor es llamado cuando un objeto se sale del alcance o esta sujeto a una eliminacion y se encarga de liberar recursos que fueron adquiridos durante el uso de un objeto.

Copy Constructor y Move Constructor

Son dos constructores especiales que son requeridos para construir un nuevo objeto, el constructor de copia inicia si se encuentra en el mismo estado que otro objeto del mismo tipo, mientras que el constructor de movimiento inicia si el objeto ya existente es de tipo “Rvalue”

Copy Assignment y Move Assignment (operator=)

Estos son llamados cuando el signo = es aplicado a dos objetos que fueron previamente contruidos, el copy se encarga de copiar un estado a otro solo si el estado original es de tipo “Lvalue” mientras que move se encarga de lo mismo pero solo si es un “Rvalue”

Defaults

Asi se les conoce si una clase consiste de miembros de datos que son exclusivamente primitivos con objetos que tendrian sentido llevar defaults como lo son los int, double, vector<int>, string, e incluso “vector<string>”. Pero hay casos donde los default no funcionan, por ejemplo cuando los miembros de datos son de tipo pointer con este ultimo localizado en la funcion de un objeto.

Arrays y Strings de estilo C

C++ posee un sistema de arrays C-style ya preinstalado y se declara asi: int arr1[n]; con “n” siendo el numero de integrales, arrl en si es un puntero a la memoria que es lo suficientemente grande para almacenar 10 integrales, tratar de aplicarle un = a un arral es imposible pues es un puntero constante, cuando el arral es pasado a una funcion solo el valor del puntero es heredado pues se perdera la informacion sobre el tamaño del arral por lo que si se quiere pasar debera ser por medio de un parametro adicional.

Templates

Function Templates

Una function de template no es una funcion de verdad si no que son patrones de lo que podria ser una funcion, por ejemplo tenemos el siguiente codigo:

```
6 template <typename Comparable>
7 const Comparable & findMax( const vector<Comparable> & a )
8 {
9 int maxIndex = 0;
10
11 for( int i = 1; i < a.size( ); ++i )
12 if( a[ maxIndex ] < a[i])
13 maxIndex = i;
14 return a[ maxIndex ];
15 }
```

Aqui tenemos un template de una funcion “findMax”, la linea que contiene la declaracion del template indica que “comparable” (Linea 6) es el argumento del template y esta puede ser reemplazada con cualquier tipo para generar una funcion por lo que si se manda a llamar “findMax” con un vector string como parametro la funcion sera generada remplazando “Comparable” con string.

Class Template

Los class template funcionan casi identicos que un fucntion template con la diferencia que class template se encarga de generar templates de clases y que no puedes declarar un objeto que pertenezca a una clase del template.

Objetos y comparables

El objeto se asume que tiene un constructor de zero-parametro, un operador= y un copy constructor, mientras que comparables tiene funciones extras como lo son “operator<” que se pueden usar para tener completo orden.

Funcion de objetos

“function object” son una forma de pasar funciones como parametros de forma que definimos una clase sin datos y una funcion de miembros para luego pasarla como una instancia de una clase de forma que pasamos una funcion poniendola dentro del objeto.

Usando Matrices

Una matriz o “matrix” en c++ es un array bi-dimensional (2d), estos pueden ser cualquier tipo de integer, character o float dependiendo de la inicializacion que se le de.

Miembros de datos, constructores y accesors basicos

La matrix es representada por un array de miembros de datos y es declarado a ser un vector<object>, el constructor construye arrays con entradas de “rows”, cada una siendo de tipo vector<object> construidos con un constructor de zero-parameter.
Despues dentro del cuerpo del constructor cada hilera es reescalara para tener columnas, por ende el constructor termina con arrays bi-dimensionales para terminar con los accesors “numrows” y “colrows” siendo facilmente implementables.

Operator[]

La idea del operator[] es si tenemos un matrix “m”, entonces m[i] deberia regresar un vectorcorrespondiendo a la hilera “i” de la matrix m de forma de que el operator[] regresa un vector<object> en vez de solo un object

Citas

(1): Cuando una funcion es llamada, nuevos elementos son creados en una pila de memoria que almacena informacion importante sobre las funciones asi como asignar espacios de memoria para parametros y regresar valores

Bibliografia

1- “¿Qué es la aritmética modular? (artículo) | Khan Academy,” *Khan Academy*. https://es.khanacademy.org/computing/computer-science/cryptography/modarithmetic/a/what-is-modular-arithmetic

2-GfG, “C Function Call by value,” *GeeksforGeeks*, Sep. 15, 2023. https://www.geeksforgeeks.org/cpp-function-call-by-value/

3-“IBM documentation.” https://www.ibm.com/docs/en/zos/2.2.0?topic=only-class-templates-c

4-Sneh, “Two dimensional array in C++,” *DigitalOcean*, Aug. 03, 2022. https://www.digitalocean.com/community/tutorials/two-dimensional-array-in-c-plus-plus