

MANEJO DE EXCEPCIONES

¿Que se hace cuando en un programa de computadora se produce un error de ejecución?

Por: Lic. María Angélica Mendoza C.
Lic. José Lauro Cortés Carrera
Docentes I.T. Orizaba

Desde sus inicios, la programación de computadoras ha originado el surgimiento de varios problemas. Problemas que de alguna forma, mediante los diferentes lenguajes de programación se han tratado de resolver; desafortunadamente no todos los problemas que han surgido, han sido resueltos, debido a diversos factores: tecnológicos, funcionales, de implementación, nuevos paradigmas de programación, etc. Algunos de estos problemas son, por ejemplo, la implementación de datos. Debido a que internamente los tipos de datos (int, flota, char, etc.) se representan en la memoria de las computadoras con diferentes longitudes (bits) dependiendo del compilador utilizado, existen diferentes compiladores de un mismo lenguaje de programación ó a la existencia de diferentes versiones de un compilador. La liberación de memoria es otro problema común. En algunos lenguajes de programación ésta se realiza de forma automática, y en algunos otros se debe liberar de forma explícita, cuando ya no se utilizan algunos recursos. Otro problema muy importante, es el manejo de algunos errores que pueden ocurrir durante la ejecución de un programa de computadora. En todos los lenguajes de programación, a dichos errores se les conoce como *excepciones*.

¿QUE ES UNA EXCEPCIÓN?

Cuando se ejecuta un programa de computadora, lo ideal es que concluya sin ningún contratiempo. Si ocurriera algún error que impidiera que siga su ejecución, se dice que ha ocurrido una “condición excepcional”; es decir, una excepción. Tales excepciones pueden ser por ejemplo: hacer referencia a un elemento fuera de los límites de un arreglo; intentar realizar una división por cero o de condiciones impredecibles, como el indicador de que se ha alcanzado el fin de archivo en un archivo de entrada. Existen algunos lenguajes de programación, como por ejemplo, C y Pascal, donde es posible insertar pruebas explícitas en el programa para buscar condiciones de excepción, en donde se utilizan códigos de retorno para informar al usuario que ha ocurrido un error, durante la ejecución de un programa. Pero normalmente estos códigos de retorno se implementan como variables globales esto es, pueden ser accedidas y manipuladas desde cualquier parte de un programa, lo que permite que el valor de una variable sea sobrescrito, de manera que el valor original de estas variables se modifica varias veces; así, cuando estas variables son recuperadas, su valor ha sido alterado,

dificultando la solución al problema ocurrido. Además, estos lenguajes no ofrecen al programador una forma de definir excepciones propias o de usuario. Pero también existen lenguajes de programación que proporcionan mecanismos propios para el manejo de excepciones, como por ejemplo PL/1, Ada y recientemente el Java, los cuales proporcionan una forma de llamar a un subprograma mediante llamadas explícitas cuando ocurre un suceso ó condición excepcional. Además de ofrecer al programador mecanismos para la definición de excepciones propias.

CLASES DE EXCEPCIONES.

Existen básicamente dos clases de excepciones:

Excepciones del sistema (default). Son las excepciones que están predefinidas en un lenguaje de programación, por ejemplo: OVERFLOW, UNDERFLOW ó END-OF-FILE y éstas se crean implícitamente cuando se incurre en la excepción correspondiente.

Excepciones definidas por el programador (propias). Son creadas por el programador e insertadas en el código, por ejemplo: “MI_EXCEPCION”, “DIVISION_POR_CERO”, “VALOR_DE_DATO_MALO” donde cada tipo de excepción definida, debe coincidir con el manejador de excepciones que va a capturar.

¿QUE ES UN MANEJADOR DE EXCEPCIONES?

Cuando sucede una excepción, la ejecución del programa se detiene, en espera de que el problema sea solucionado en alguna parte. La parte del programa a la que se le pasa el control para que lleve a cabo la solución del problema se le llama “manejador (gestor) de excepciones”. El manejador de excepciones contiene instrucciones que especifican las acciones en particular que se deben tomar antes de que pueda continuar el procesamiento normal del código.

Un manejador de excepciones puede aparecer en cualquier bloque del programa que esté dentro del alcance de la declaración correspondiente a la excepción. Además, pueden aparecer diferentes manejadores de excepciones en distintos bloques de código.

Debido a que un manejador de excepciones, es invocado sin una llamada explícita, no requiere de paso de parámetros ó de un nombre.

Un manejador de excepciones contiene típicamente:

Un conjunto de declaraciones de variables locales.
Una serie de enunciados ejecutables.

PROPAGACIÓN DE EXCEPCIONES.

Muy frecuentemente en un programa, el lugar donde se origina una excepción, no siempre es el lugar indicado para manejarla. Cuando una excepción es manejada en un subprograma, que no es el mismo en el cuál surgió, se dice entonces que la excepción se ha propagado desde el punto en el cuál ha surgido hacia el punto donde fue manejada. La determinación de cuál manejador de excepciones es el indicado para manejar una excepción en particular, está definida en término de la cadena dinámica de activaciones del subprograma que ha originado la excepción. Por ejemplo: Si una excepción “e” surge en un subprograma “A”, entonces la excepción “e” será manejada por el manejador definido en “A”, si es que lo hay; sino existe ningún manejador entonces termina el subprograma. Otro ejemplo, es el que se muestra en la siguiente figura, en donde el subprograma “B” llama al subprograma “A”, ahora entonces, si la excepción “e” surgida “A”, no se maneja ahí mismo, se propagará la excepción al subprograma “B”, en busca de un manejador de excepciones apropiado. En este caso el subprograma “B” proporciona un gestor de excepciones para la excepción “e”. Por tanto, se dice, que la excepción “e” se ha propagado desde el punto “A” en que surgió hasta el punto “B” en que fue manejada. Si no se proporciona un manejador de excepciones apropiado el programa termina y se invoca un manejador estándar definido por el sistema.

¿QUE SUCEDE DESPUÉS DE QUE HA SIDO MANEJADA LA EXCEPCIÓN?

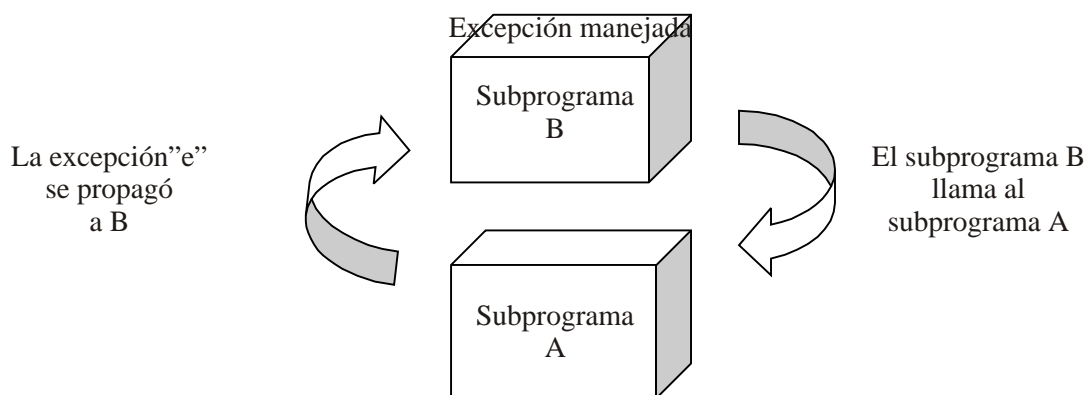
- ¿Hacia dónde se debe transferir el control una vez que se ha completado el proceso del manejo de una excepción?
- ¿Debe de regresar al punto desde donde se propagó la excepción?
- ¿Debe de terminar la ejecución del programa después de que fue manejada la excepción?
- ¿Debe de regresar el control al subprograma en donde se manejó la excepción?
- ¿Debe terminarse el subprograma que contiene el manejador de excepciones, de tal forma que su llamador continúe como si nada hubiera pasado?

Esto dependerá del lenguaje de programación elegido. Por ejemplo, en Ada, la ejecución del programa termina una vez que se ha manejado una excepción. Otros lenguajes proporcionan otras alternativas, como por ejemplo Java, que también permite que continúe la ejecución del programa una vez que se ha manejado la excepción. La verificación de excepciones se puede implementar “por medio de interrupciones de hardware ó trampas” ó “pueden llevarse en el software de soporte del sistema operativo”, pero regularmente se implementan “por medio de códigos especiales insertados por el traductor del lenguaje en el código ejecutable”.

LAS EXCEPCIONES EN JAVA.

El manejo de las excepciones en Java, lleva el manejo del error en tiempo de ejecución al paradigma de la orientación a objetos. Una excepción en Java, es un objeto que describe una condición excepcional que se ha producido en un fragmento de código.

La excepción “e” es manejada en B



En Java, una excepción, siempre es desencadenada cuando “alguien” (otro método ó un usuario), en alguna parte durante la ejecución de un programa Java, llama a un método. Cuando surge una condición excepcional, se crea un objeto Exception (excepción) que es enviado al método que provocó la excepción. Ese método puede capturar la excepción en base a su tipo concreto. Los métodos también se pueden proteger de una salida prematura mediante una excepción y hacer que se ejecute un bloque de código justo antes de que una excepción provoque que termine el método. Las excepciones pueden aparecer de manera implícita en un método o pueden ser creadas manualmente y enviadas para informar de alguna condición de error al método llamante.

¿EN QUE MOMENTO SE DEBE LANZAR UNA EXCEPCIÓN?

La respuesta puede resumirse en una directiva. Si un método encuentra una condición anormal que no pueda manejar, debería lanzar una excepción. Las excepciones son “lanzadas” por la máquina virtual de Java. En Java, los métodos que llaman a otros métodos, no comprueban valores de retorno. Si el método que ha sido invocado se ejecuta sin ningún contratiempo, quién llamó al método está seguro de que no ocurrió nada anormal y no tendrá nada de que preocuparse. Algunos ejemplos de excepciones, que pueden ocurrir durante la ejecución de un programa en Java, se muestran a continuación, así como su significado.

¿COMO SE SOLUCIONAN LAS EXCEPCIONES?

Al igual que otros lenguajes que proporcionan manejo de excepciones, se deben solucionar mediante un manejador de excepciones. Una vez que ha ocurrido una excepción para indicar que hay un problema, en alguna parte se espera que esta excepción sea capturada y el problema pueda ser tratado. Las excepciones son capturadas por gestores de excepciones ubicados a lo largo de los hilos de invocación

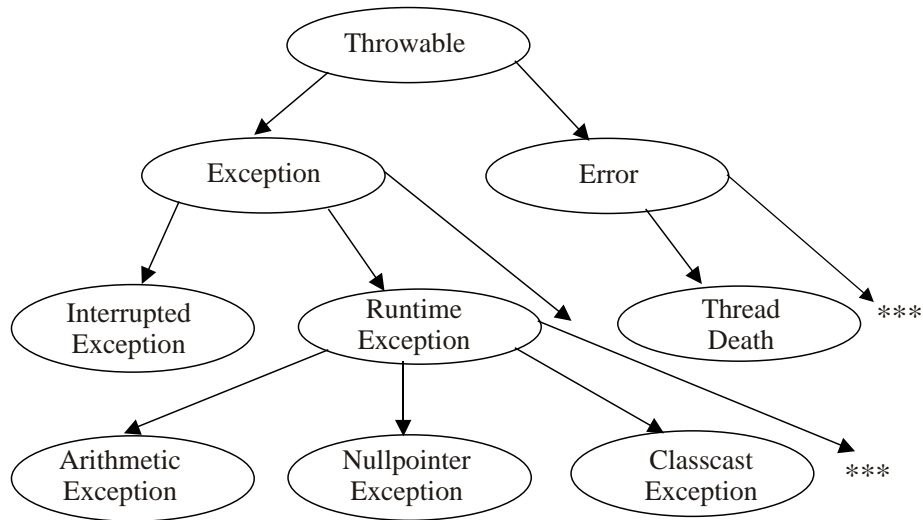
de métodos de pila. Un manejador de excepciones contiene un conjunto de sentencias ejecutables que tratan el problema para darle solución. Si el método llamado no esta preparado para capturar la excepción, éste lanza la excepción hacia arriba hasta su método llamado, y así sucesivamente hasta encontrar un gestor apropiado para la excepción. Cuando se programa, se deben ubicar los manejadores de excepciones (debería haber uno para cada tipo de excepción surgida) estratégicamente, así un programa capturaré y manejará todas las excepciones mediante el sistema en tiempo de ejecución.

¿QUE SUCEDE CUANDO UNA EXCEPCIÓN NO ES CAPTURADA POR ALGUN GESTOR DE EXCEPCIONES?

Si una excepción no es capturada y tratada por algún gestor se excepciones en particular, se ejecutará el gestor en tiempo de ejecución por defecto, el cual imprimirá un mensaje indicando el tipo de excepción ocurrido, así como el trazado de la pila (que contiene un registro de las llamadas a método) del lugar el donde ocurrió la excepción y el programa terminará abruptamente.

Las excepciones son objetos, los objetos de excepción son una instancia de la clase Throwable (lanzable) o cualquiera de sus instancias. Cuando se lanza una excepción, en realidad se está lanzando un objeto. La figura anterior, muestra una parte de la clase Throwable contenida en el paquete java.lang de Java. Como se puede observar existen dos clases que descienden directamente de esta clase: la clase Exception y la clase Error. Sin embargo solo se pueden lanzar los objetos cuyas clases desciendan directamente de la clase Throwable. Esta clase sirve como base para una gran cantidad de objetos de excepción que se pueden lanzar en un programa en Java.

Clase de excepción	Significado
ArithmeticException	Una condición aritmética excepcional ha ocurrido. Por ejemplo, una división por cero.
ArrayIndexOutOfBoundsException	Una matriz fue accedida con un índice ilegal (fuera de los límites permitidos)
NullPointerException	Se intentó utilizar null donde se requería un objeto
IOException	Ocurrió una excepción de I/O (entrada/salida)



Las excepciones derivadas de la clase de la clase Exception son las que los programas de usuario deberían capturar y manejar mediante un gestor de excepciones. Esta clase también es la base para aquellas excepciones que nuestros propios programas pueden lanzar, es decir, para elaborar nuestras propias excepciones.

USANDO EXCEPCIONES CON C++

El manejo de excepciones en C++ aplicado adecuadamente, hará el código mas seguro y sencillo den entender, debido a que no es necesario estar manejando los errores en varias partes del programa. Una de las nuevas características definidas por ANSI para lograr que el lenguaje C++ sea el mas seguro en su programación es el uso de excepciones, este mecanismo permite que se puedan detectar y en determinados casos recuperar errores al tiempo de ejecución de un programa

El compilador Borland C++ 4.0 es uno de los primeros en proveer soporte a esta nueva característica del lenguaje C++. Cuando un error es detectado se puede enviar una excepción con la información necesaria para poder diagnosticar las causas que generaron el error así como el punto en que este ocurrió. De esta forma el programador puede definir las rutinas que se pueden ejecutar después de ocurrido el error y antes de la terminación del programa.

Suponiendo que se desea escribir un programa para reservar varios arreglos de enteros, en los cuales el primer elemento es el numero de elementos en el arreglo y los elementos restantes son inicializados en diferentes formas. L implementación del programa podría hacerse de la siguiente forma:

```

Int * Reserva(int num)
{ int *arreglo=new int(num+1);
  arreglo [0]=num;
  return arreglo; }

int *Generalguales(int num, int valor)
{ int *arreglo=Reserva(num);
  for (int i=1; i<=num;i++)
  arreglo[i]=valor;
  return arreglo;}

int *GeneraSecuencia(int num)
{ int *arreglo=reserva(num);
  for (int i=1;i<=num;i++)
  arreglo[i]=i;
  return arreglo; }

void Algo(void)
{ int*arreglo=generalguales(2,3);
  //mas código... }
  
```

Obviamente esta implementación tiene el problema de no verificar que la inicialización de los elementos ocurra satisfactoriamente. La forma más fácil de hacer la validación es haciendo que la función reserva() revise si el valor de arreglo es igual a 0 o no. Igualmente a cada una de las funciones que utilicen el arreglo se les debe poner el código para revisar si el valor de la variable arreglo sea válido.

La solución anterior llevaría a tener que repetir código en cada una de las funciones en las cuales reservar memoria sea una operación crítica. Creo que todos los programadores además de poseer sistemas seguros buscamos hacer código de una forma en que no sea tediosa ni repetitiva.

EL MECANISMO DE LAS EXCEPCIONES

Escribir código que utilice las excepciones es relativamente sencillo, para ello se definieron 3 nuevas palabras reservadas: try, catch y throw. Cuando en el programa existe algún problema se envía (“throw”) una excepción. Cuando se desea manejar las excepciones, el código del cual se espera que ocurra una excepción se pone en un bloque try, seguido de una o varias cláusulas match para procesarlas, por ejemplo:

```
Struct Error
{Error(int num):Num(num){ }
Int Num;};

int *Reserva(int num)
{int *arreglo=new int[num+1]
if(arreglo=0)
throw Error(num);
else
{arreglo[0]=num;
return arreglo;}}
```

```
int *GeneralGuales(Int num, Int valor)
{int *arreglo=Reserva(num)
for(int i=1;i<=num;i++)
arreglo [i]=valor;
return arreglo}
```

```
void Algo (void)
{try {int *arreglo=GeneralGuales (2,3);
//mas código... }
match (const Error &Falla)
{cout << “Falla en generacion del
arreglo<<falla num<<”elementos”;
exit(1);}
match (...)
{cout <<”error desconocido”;
exit (19;}}
```

La estructura error fue creada para guardar la información del problema en el programa. Esta estructura puede ser tan compleja como se desee. New pudo reservar memoria, si hubo error se crea un objeto de tipo error y se envía la excepción con este tipo de dato.

La función generaIguale () permanece sin cambios. Si tuviéramos que usar las técnicas del lenguaje C, la función tendría que revisar el valor regresado por reserva para detectar el error. Con el uso de excepciones esto no es necesario. En este caso la función GeneralGuales no necesita procesar el error y solo pasa el error a la función anterior, por lo tanto no debe ser modificada cuando se usan excepciones.

En la función Algo() es donde se desea manejar los errores que puedan ocurrir, por lo tanto se introdujo un bloque try. Este bloque le indica al compilador que estamos interesados en manejar las excepciones que puedan ser enviadas durante la ejecución del bloque de instrucciones.

Un bloque try debe ser seguido siempre de por lo menos una cláusula match. Cada cláusula debe iniciar con la palabra match. Cada cláusula debe iniciar con la palabra match y continuar con el nombre del tipo de excepción que recibe, en nuestro ejemplo la línea match (Const Error &Falla) indica que el bloque siguiente será ejecutado si un objeto del tipo error fue enviado durante la ejecución del bloque try anterior. Tal como si fuera una llamada a una función, nosotros podemos hacer una instancia este objeto para poder usarlo dentro de la cláusula catch.

La segunda cláusula match en la función algo () especifica un tipo “...” lo cual significa que puede recibir cualquier error. Cuando una excepción es enviada durante la ejecución de un bloque try, las cláusulas match son examinadas en el orden en que aparecen en el programa y la primera que pueda recibir el tipo de objeto que fue enviado será ejecutada.

En nuestro ejemplo, cuando un objeto de tipo error es enviado, la primera cláusula match es ejecutada por que ella procesa objetos del mismo tipo. Si un objeto de otro tipo es enviado, la segunda cláusula match es ejecutada. Existen muchas opciones más en el manejo de excepciones, las cuales iremos aplicando paulatinamente en artículos posteriores.

RESPUESTAS AL ACERTIJO DEL NÚMERO ANTERIOR:

Problema A. Los números de mi casa pueden ser __, __, __ y __.

Problema B. El área sombreada vale __ unidades cuadradas