استثناها و خطاها

استثناها خطاهای درحال اجرا هستند. به طور معمول خطاها توسط دستور if تشخیص و در صورت برقرار بودن شرط عمل مناسب، که اغلب نمایش پیغام و اتمام برنامه بود، انجام می گرفت. راه دیگر هنگام مواجه شدن با خطا گیر انداختن آن هنگام پیش آمدن شرایط خطاست. کدی که ممکن است باعث خطا شود اجرا می شود و هر خطائی که ایجاد می شود گرفته می شود و عمل مناسب انجام می گیرد. بلاک های try-catch بر اساس این مفهوم عمل می کنند. مدیریت خطاها به طور سطحی در اینجا معرفی می شود.

خطاها
مدیریت استثناها
بلاک های try-catch

مدیریت خطاها از موضوعات مهم در برنامه نویسی است. حالات متعددی وجود دارند که ممکن است ایجاد خطا کند. مثلا باز کردن فایلی که وجود ندارد یا دریافت ورودی نادرست از کاربر. کنترل این خطاها هنگام اجرای برنامه اهمیت زیادی دارد. اگر برنامه این گونه خطاها را در دست نگیرد برنامه سقط می کند و نارضایتی کاربر را باعث می شود. روش قدیمی بررسی خطا توسط if و دادن یک پیغام دوستانه به کاربر و اتمام برنامه بود. C++ راه دیگری را توسط بلاک های try-catch برای مدیریت خطاهای در حال اجرا در اختیار مي گذارد.

خطاها

خطاها را به سه دسته می توان تقسیم کرد:

- خطاهای گرامری (syntax error). خطاهائی که با رعایت نکردن قواعد زبان برنامه نویسی توسط کامپایلر تشخیص داده می شوند. مثلا بجای if(x!=y) بنویسید if(x<>y). کامپایلر کد برنامه را تا وقتی خطای گرامری وجود دارد کامپایل نمی کند.
- خطاهای زمان اجرا (runtime error). هر وقفه ای که در جریان عادی اجرای برنامه پیش آید که معمولا باعث سقط برنامه می شود. مثلا باز کردن فایلی که وجود ندارد یا تقسیم بر صفر. این خطاها استثنا (exception) نامیده می شوند.
- خطاهای منطقی (login error). وقتی برنامه کامپایل و اجرا می شود اما به دلیل خطائی در منطق برنامه نتایج غلطی تولید می کند. سخت ترین نوع خطا است که معمولا به آن bug هم گفته می شود.

برای رفع خطا ابتدائی ترین کاری که می شود کرد تست کلیه برنامه و کشف و برطرف سازی خطا است. کنترل مناسب خطاها از سقط برنامه جلوگیری می کند. کامپایلر خطاهای گرامری را مشخص می کند. مدیریت استثناها خطاهای زمان اجرا را بر عهده دارند بنابراین خطاهای منطقی بزرگترین مشکل برنامه نویس خواهد بود.

مدیریت استثنا

قبل از اینکه سیستم مدیریت استثناها معرفی شود برنامه های کامپیوتری کاملا بدون خطا نبودند. البته برای گرفتن خطاها روش هائی وجود داشت که معمول ترین آن بررسی مقداری بود که توسط توابع برگردانده می شد و نحوه اجرای آن را بیان می کرد. کدی که تابع را فراخوانی می کرد مقدار بر گشتی را چک می کرد و مطابق با آن عمل می کرد (اگر با استفاده از Windows API برنامه نوشته باشید با این روش آشنا هستید).

مفهوم کلی مدیریت استثناها ساده است. هر زمان که استثنائی شناسائی می شود یک فلگ فرضی خطا بالا می رود. سیستمی همواره مراقب این فلگ خطاست. و درصورت بالا رفتن فلگ کد مدیریت خطا فراخوانی می شود. بالا رفتن فلگ خطا را اصطلاحا گیر انداختن (throwing) خطا می نامند. وقتی خطائی به دام انداخته می شود سیستم با گرفتن خطا (catching) عکس العمل نشان می دهد. محاصره کردن بلاکی از کد حساس به خطا و مدیریت استثنا را سعی کردن (trying) به اجرای بلاک می نامند.

دلایل خوبی برای استفاده از مدیریت استثناها وجود دارد. مهمترین آن جداکردن کد مدیریت خطا از کد برنامه است. وقتی استثنائی گرفته می شود مسیری موازی اجرا طی می شود و با اجرای نرمال کد برنامه تداخلی ندارد. این باعث می شود نوشتن کد ساده تر

شود چون مجبور نیستید برای هر خطائی چک داشته باشید. علاوه بر این یک استثنا نمی تواند ندیده گرفته شود و تضمین می کند که با خطا برخورد کند. و چون جلوی سقط ناخواسته برنامه را می گیرد اتکاپذیری برنامه را بالا می برد.

از مزایای دیگر مدیریت استثناها این است که خطا می تواند خارج از محدوده تابع گیرانداخته شود. یعنی اگر درونی ترین تابع خطائی داشته باشد این خطا می تواند تا تابع بالائی که دارای بلاک trying است منتشر شود که به برنامه نویس اجازه می دهد کد مدیریت خطا را در هر جا مثل تابع اصلی قرار بدهد.

بلاک های try-catch

مدیریت استثنا بر اساس مفاهیمی که گفته شد توسط سه دستور try، catch و throw عمل می کند. بلاک کدی می خواهیم استثناهای آن را بگیریم با دستور try مشخص می شود. درون بلاک می توان هر خطائی با دستور throw گیر افتاده و از بین می رود. بلاک catch که محلی برای کد مدیریت خطا است بلافاصله بعد از بلاک try قرار می گیرد.

فرم کلی به صورت زیر است:

```
try {
   ...
   ...
   throw Exception()
   ...
   ...
   }
catch( Exception e ) {
   ...
   ...
   }
```

مثال. هر استثنائی که موقع دسترسی به فایل test.txt رخ دهد گرفته و پیغام مناسب نمایش داده می شود.

```
#include <fstream.h>
#include <iostream.h>

int main () {
   try {
      char buffer[256];
      ifstream myfile ("test.txt");
      while (! myfile.eof() )  {
         myfile.getline (buffer,100);
         cout << buffer << endl;
         }
      }
  catch(…) {
     cout << "There was an error !\n";
     }
   return 0;
}
```

سه نقطه درون پرانتز catch به معنی اینستکه کلیه خطاها توسط این بلاک مدیریت می شود. می توان انواع مختلفی از خطاها را هندل کرد.

وقتی استثنائی رخ می دهد کنترل به بلاک catch منتقل می شود بنابراین دستورات بعد از throw اجرا نخواهد شد. در بلاک catch کد باید به نحوی باشد که اجرای برنامه را به صورت معمول ادامه یابد یا در صورت برطرف نشدن خطا با exit یا abort خاتمه پیدا کند. اگر بلاک catch اجرای برنامه را به پایان نرساند کنترل اجرا به دستور بعد از بلاک try-catch منتقل می شود.

نکته. بین دو بلاک هیچ چیز نباید باشد.
نکته. در صورت گیر انداختن یک استثنا دستورات بعد از throw اجرا نخواهد شد.
نکته. یک استثنا را می توان به تابعی که تابع جاری را صدا زده است انداخت تا مدیریت شود .

مثال. خطا در تابع گرفته می شود اما به برنامه اصلی برای مدیریت ارسال می شود.

```
#include <iostream.h>
float divide_number(float , float);

int main() {
   float dividend,divisor,answer;
   try {
      cout << "Please enter a number \n";
      cin >> dividend;
      cout << "Please enter a number \n";
      cin >> divisor;

      answer = divide_number(dividend,divisor);
      cout << dividend << " divided by ";
      cout << divisor << " is " << answer;
      }
    catch(...) {
      cout << "oops, there is an error!";
      }

      return 1;
}
float divide_number(float num1, float num2) {
  try  {
    float answer;
    answer = num1/num2;
    return answer;
    }
  catch(...) {
    throw;
    }
}
```

هر دستور try می تواند با چندین دستور catch در ارتباط باشد. هر کدام برای نوع مختلفی از استثنا. نوع داده ای که در دستور catch مشخص شده است با استثنای تطبیق داده می شود سپس بخش مربوطه به اجرا در می آید و از بقیه صرفنظر می شود. هر نوع داده ای از جمله کلاس های تعریف شده در برنامه می تواند بدام انداخته شود.

مثال. اعداد صحیح توسط catch گرفته می شود. اگر تقسیم بر صفر باشد یک عدد صحیح به بلاک catch داده می شود. بلاک catch استثنا از نوع صحیح را می گیرد.

```
#include <iostream.h>

int main(){
   int answer, divisor, dividend;

   try  {
      cout << "Please enter an integer \n";
      cin >>divisor;

      cout << "Please enter another integer \n";
      cin >> dividend;

      if(dividend ==0) throw 0;

      answer = divisor/dividend;
      cout << answer;

      return 0;
      }
   catch (int i) {
      cout << "You cannot divide by zero";
      }
```

```
}
```

مثال. کاربرد مدیریت استثناها در استفاده از حافظه پویا.

```
try {
   NodePtr = new ListNodeClass(Item, NextPtr);

   if (NodePtr == NULL) {
      throw "Cannot allocate memory";
      }
   NodePtr = GetNode(Item, Front->Next);
   }
catch(char * Str) {
   cout << "Error:  Could not insert at front -- " << Str
      << endl;
   throw;
   }
```

مثال. مدیریت استثنا با چند بلاک catch.

```
try{
   cin >> test;
   if (test == 0 ) throw test;
   if (test == 1 ) throw 'a';
   if (test == 0 ) throw 123.5;
   }
catch(float FloatNum){
   cout << "Caught an exception with float value: " << FloatNum << endl;
   }
catch(int IntNum) {
   cout << "Caught an exception with int value: " << IntNum << endl;
   }
catch(...){   // catch anything else
   cout << "Caught an exception with type not int or float" << endl;
   }
```

وقتی چندین خطا را پاسخ می دهید سلسله مراتب خطاها اهمیت دارد. خطاها با ترتیبی که در سلسله مراتب نشان داده شده است باید در تله انداخته شوند یعنی اگر شما run-time-error را گیر انداختید نمی توانید عبارت catch دیگری برای file access error داشته باشید چون خطاهائی که در سلسله مراتب بالاتر هستند کلیه خطاهای پایینی را در بر دارند.

```
Exception - the parent class for all exception class
   Login-error - the parent class of a variety of logic error
      invalid-argumet
      out-of-range
   Runtime-error - the parent of class for a variety of rumtime error
      overflow-error
      FileAccessError
      range-error
```