C/C++資料結構與程式設計

例外處理

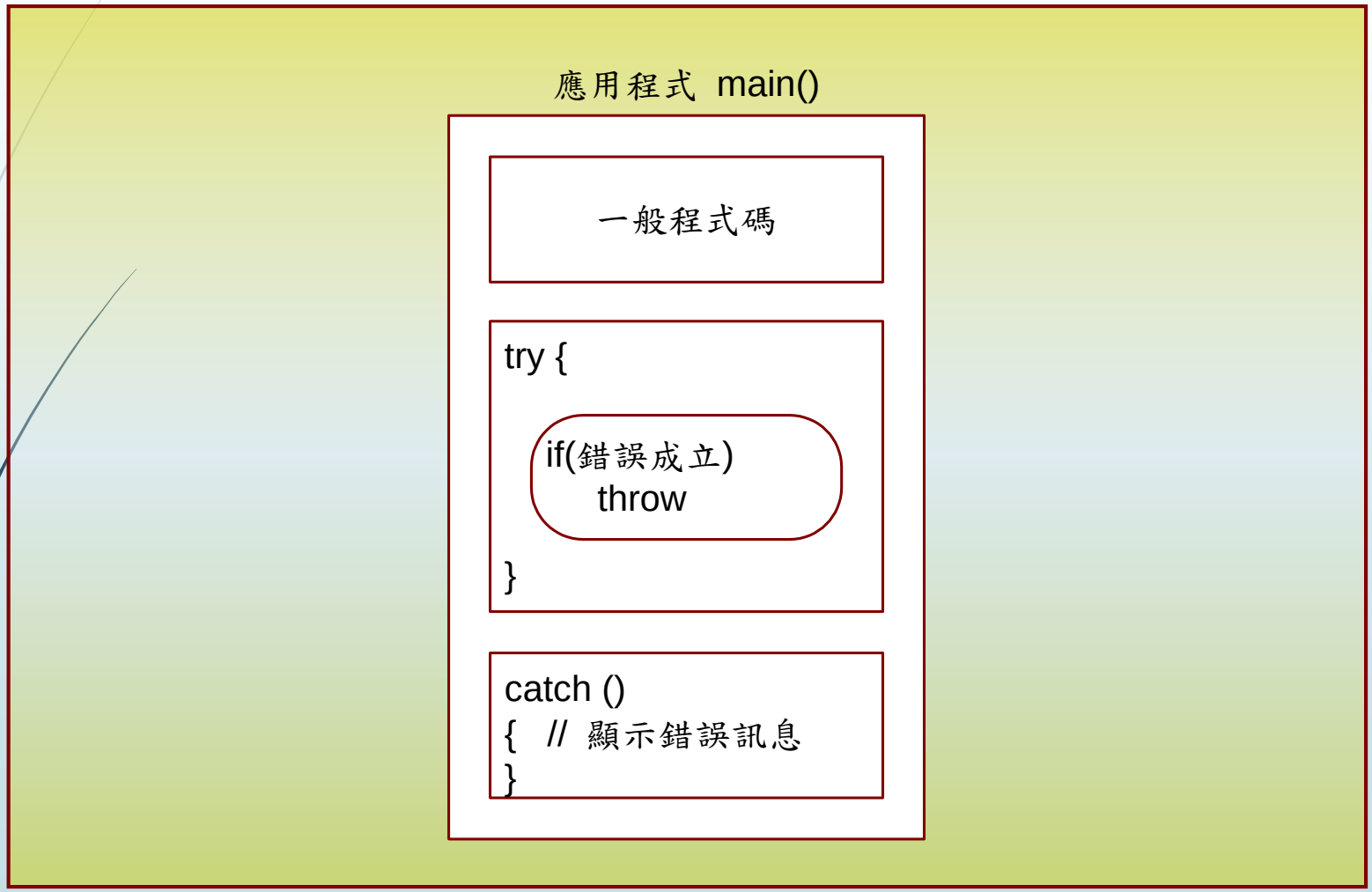
NTU CSIE

講師：張傑帆

錯誤處理

- 例外（exceptions）
  - 是執行程式發生錯誤或是非預期的事件時，發出警告或錯誤信號。
  - 使用例外處理則可以自動監控程式執行時是否產生錯誤。

簡單錯誤處理



簡單錯誤處理 (續)

```
try {
 if(錯誤條件)
      throw 參數;
}
catch(資料型態 參數)
{
 // 顯示錯誤訊息;
}
```

簡單錯誤處理 (續)

- 範例一

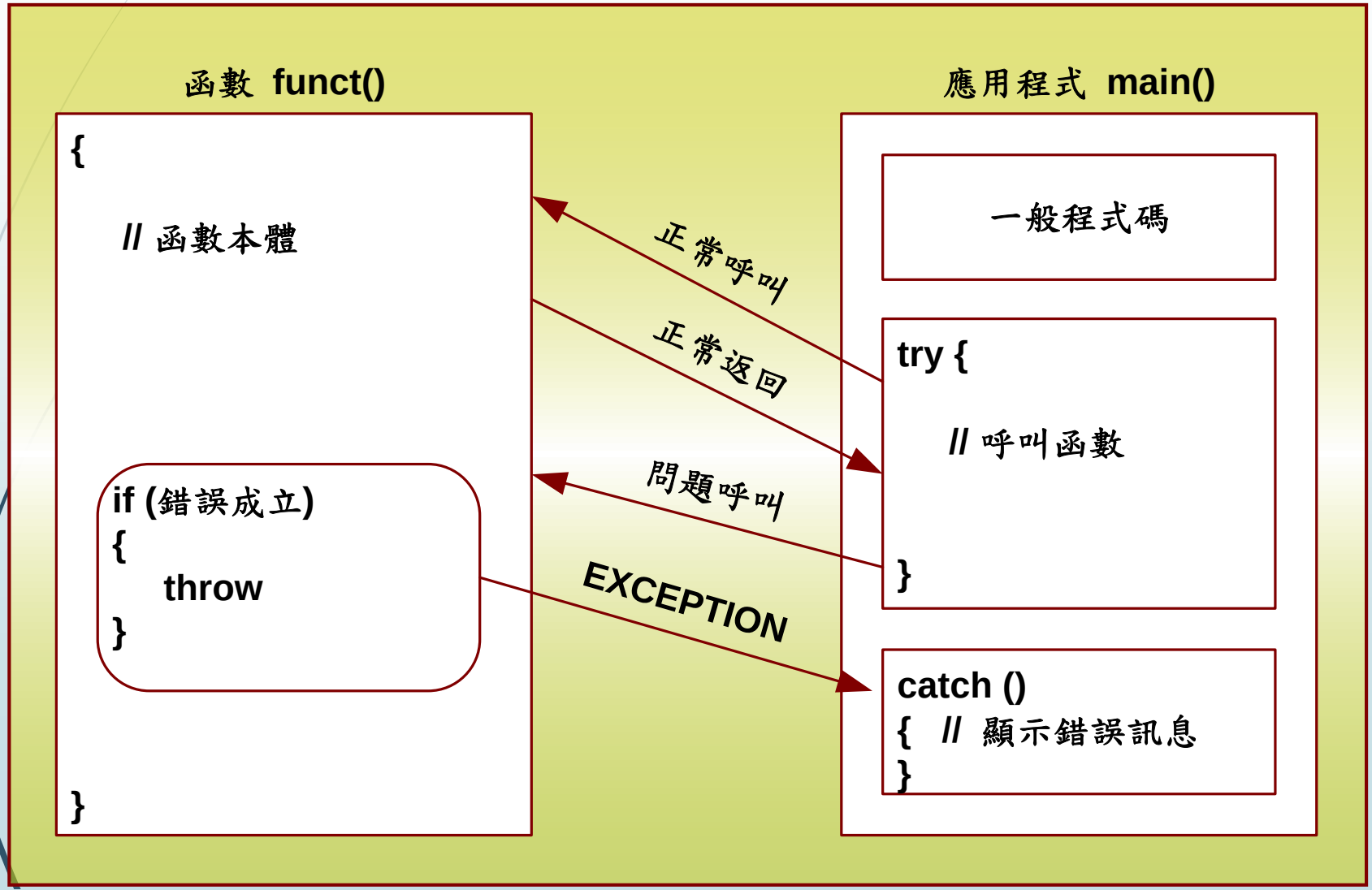
```
float nomer, denom;
cout << "請輸入被除數 = ";
cin >> nomer;
cout << "請輸入除數 = ";
cin >> denom;
if(denom == 0)                          //若除數等於0
      cout << "錯誤：除數為 0\n";          //則輸出錯誤訊息
else                                    //若除數不等於0
      cout << nomer/denom << endl;      //則輸出運算值
```

簡單錯誤處理 (續)

- 範例二

```
float nomer, denom;
cout << "請輸入被除數 = ";
cin >> nomer;
cout << "請輸入除數 = ";
cin >> denom;
try{
        if(denom == 0)                          //若除數等於0
                throw"錯誤：除數為 0\n";          //則輸出錯誤訊息
        else                                    //若除數不等於0
                cout << nomer/denom << endl;    //則輸出運算值
}catch(char *divideByZero) {                    //捕捉例外
                cout << “錯誤：除數為 0\n”; }     //則輸出錯誤
```

函數錯誤處理


函數 funct()
{
// 函數本體
if (錯誤成立)
{
throw
}
}
應用程式 main()
一般程式碼
try {
// 呼叫函數
}
catch ()
{ // 顯示錯誤訊息
}
正常呼叫
正常返回
問題呼叫
EXCEPTION

函數錯誤處理 (續)

- **傳回型態 func() {**
  if(錯誤條件)
  throw 參數; // throw 例外
  return 傳回值;
  }

- int main(int argc, char *argv[]) {
  try {
  func(); // 呼叫func() 函數
  }
  catch(資料型態 參數)
  {
  // 顯示錯誤訊息;
  }
  return 0;
  }

函數錯誤處理 (續)

- 範例一

```
float divide(float numer, float denom)
{
        if(denom == 0) {                          //若除數等於0
                cout << "錯誤：除數為 0\n";          //輸出錯誤訊息
                return 0;                         //並傳回數值0 (邏輯錯誤)
        } else {                                  //若除數不等於0
                return numer / denom;             //則傳回除法運算值
        }
}
```

函數錯誤處理 (續)

- 範例一 (續)

```
int main(int argc, char *argv[])
{
        float nomerator, denominator, quotient;
        cout << “請輸入被除數：” ;
        cin >> nomerator;
        cout << “請輸入除數：” ;
        cin >> denominator;
        quotient = divide(nomerator, denominator);          //呼叫divide函數
        cout << “除法運算值 = ” << quotient << endl;    //輸出除法運算值
        return 0;
}
```

函數錯誤處理 (續)

- 範例二

```
float divide(float numer, float denom)
{
        if(denom == 0) {                                    //若除數等於0
                throw “錯誤：除數為 0\n”;                    //投擲例外
        } else {                                            //若除數不等於0
                return numer / denom;                       //傳回除法運算值
        }
}
```

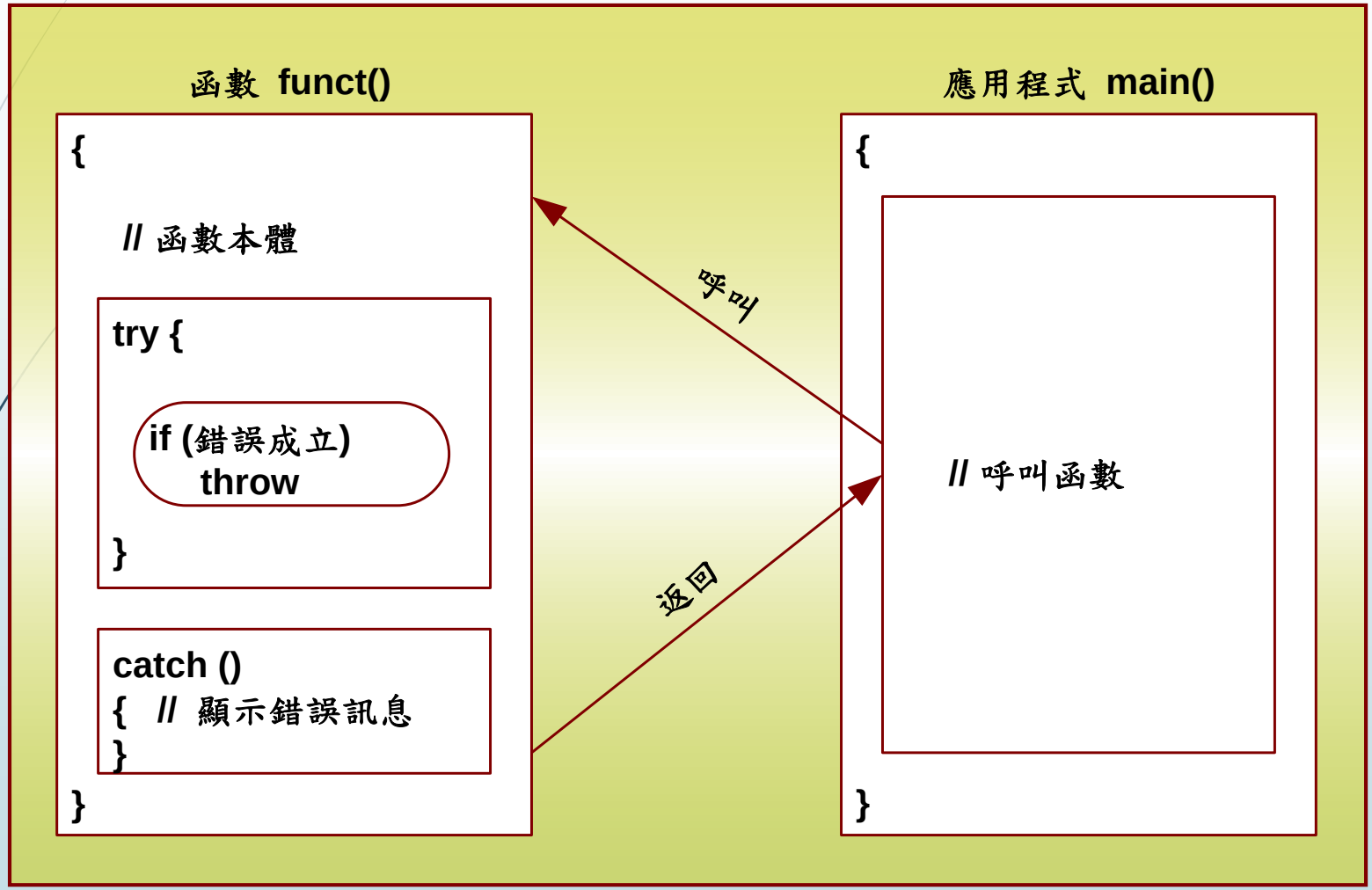
函數錯誤處理 (續)

- 範例二 (續)

```
int main(int argc, char *argv[]) {
        float nomerator, denominator, quotient;
        cout << “請輸入被除數：” ;
        cin >> nomerator;
        cout << “請輸入除數：” ;
        cin >> denominator;
        try {
                quotient = divide(nomerator,denominator);
                cout <<“除法運算值 = ” << quotient << endl;
        } catch(char *divideByZero) {                    //捕捉例外
                cout << “錯誤：除數為 0\n”;                    } //則輸出錯

        return 0;
}
```

函數錯誤處理 (續)


函數 funct()
{
// 函數本體
try {
if (錯誤成立)
throw
}
catch ()
{ // 顯示錯誤訊息
}
}
呼叫
返回
應用程式 main()
{
// 呼叫函數
}

函數錯誤處理 (續)

- **傳回型態 func()**

```
{
   try {
         if(錯誤條件)
               throw 參數;
         return 傳回值;
   }
   catch(資料型態 參數)
   {
         // 顯示錯誤訊息;
         return 傳回值;
   }
}
```

- int main(int argc, char *argv[])

```
{
   func();            // 呼叫func() 函數
   return 0;
}
```

函數錯誤處理 (續)

- 範例三

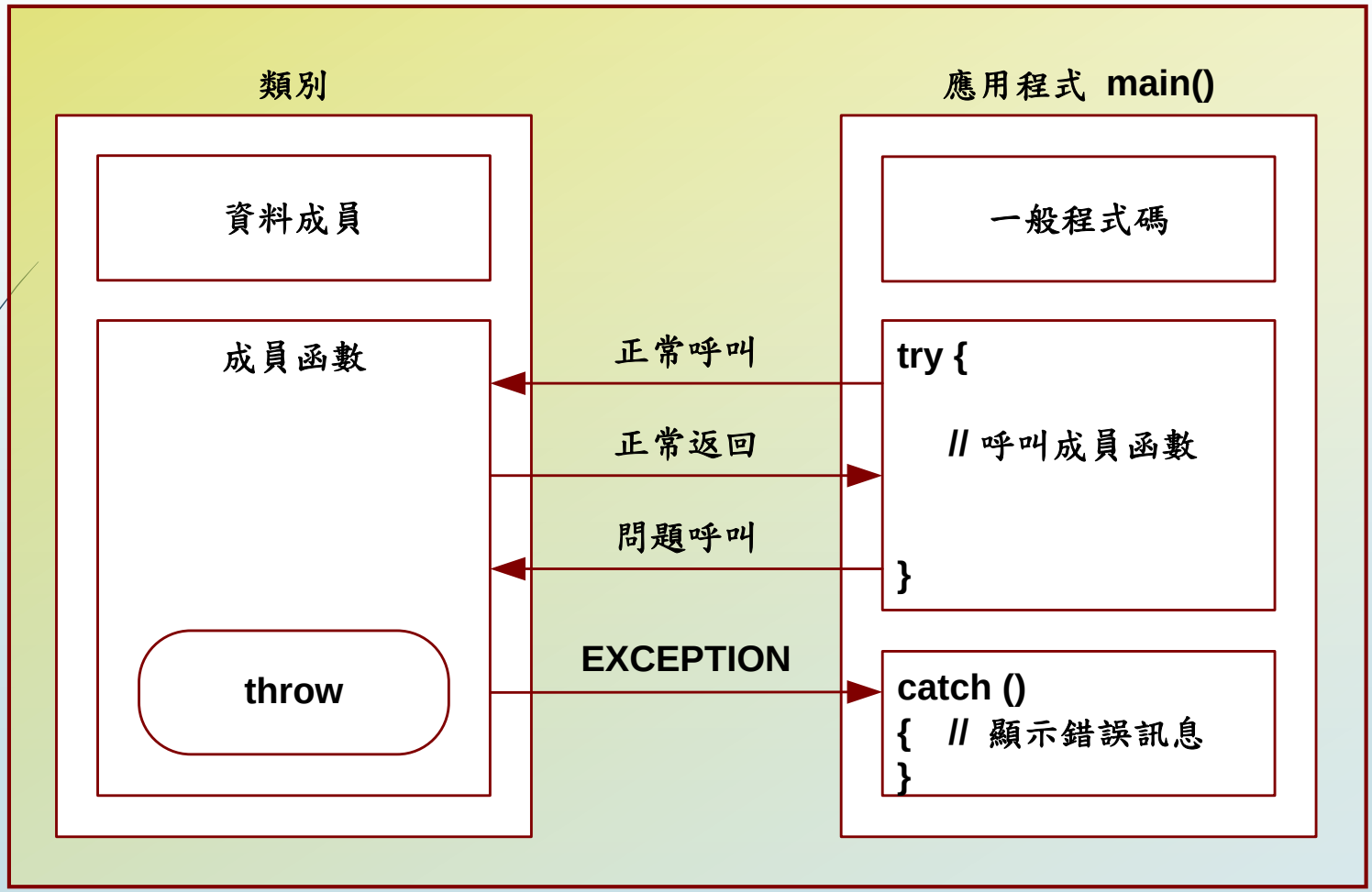
```
int check(int i)                                    //含有throw的函數
{
      try {                                         //try
            if (i == 0) throw i;                    //若輸入為0則throw
            cout << "輸入為 "   << i << endl;
            return i;
      } catch(int i) {                              //捕捉例外
            cout << "輸入為 "   << i << "，程式結束！\n";
            return i;
      }
}
```

函數錯誤處理 (續)

- 範例三 (續)

```
int main(int argc, char *argv[])
{
        int n;
        do
        {
                cout << “請輸入任意整數 (輸入 0 則結束)：”;
                cin >> n;
        } while(check(n));                    //呼叫check函數
        return 0;
}
```

類別錯誤處理


類別
應用程式 main()
資料成員
一般程式碼
成員函數
正常呼叫
try {
// 呼叫成員函數
正常返回
問題呼叫
}
EXCEPTION
throw
catch ()
{ // 顯示錯誤訊息
}

類別錯誤處理 (續)

```
class Test {
public:
   class AnError {                  // 自定類別型態
                                    // 沒有程式碼;
   };
   傳回型態 functA()      {
         if(錯誤條件)
               throw AnError;       // throw類別型態參數
         return 傳回值;
   }
};
int main(int argc, char *argv[]) {
   Test obj;
   try {
         obj.functA();
   } catch(Test::AnError) {         // catch類別型態參數
                                    // 顯示錯誤;
   }
   return 0;
}
```

類別錯誤處理 (續)

- 範例一

```
class Stack {                                   //自定Stack資料類別
        int st[MAX];
        int top;
public:
        class Range { };                        //ERROR類別
        Stack() { top = -1; }
        void push(int i){
                if(top >= MAX – 1)
                        throw Range();          //投擲例外
                st[++top] = i;                  //否則top=top+1
        }
        int pop() {
                if(top < 0)                     //若上限小於0
                        throw Range();          //投擲例外
                return st[top--];               //否則top=top-1
        }
};
```

類別錯誤處理 (續)

- 範例一 (續)

```
int main(int argc, char *argv[]) {
        Stack s;
        Try {
                s.push(10);                     //top=0; st[0]=10;
                s.push(40);                     //top=1; st[1]=40;
                cout << s.pop() << endl;    //傳回st[1]=40; top=0
                cout << s.pop() << endl;    //傳回st[0]=10; top=-
                cout << s.pop() << endl;     //top<0; throw Range()
        } catch(Stack::Range) {                 //捕捉例外
                cout << "堆疊滿了或空了！";
        }
        return 0;
}
```