國立屏東大學 資訊工程學系 物件導向程式設計

3. 變數與常數

3.1 變數

- C++語言變數命名具備以下規定：
  - 只能使用英文大小寫字母、數字與底線(_)
  - 不能使用數字開頭
  - 大寫與小寫字元將視為不同字元
  - 不能與C++語言的保留字相同
  - 在C++的實作中，以一個底線開頭後接一個大寫字母，或是以兩個底線開頭的變數，被保留做為C++的實作用途(供編譯器及其相關資源使用)。因此，使用這樣的命名並不會造成compiler的錯誤，但有可能造成執行上的錯誤。

C++語言共有以下84個保留字:

alignas	alignof	and	and_eq	asm	auto	bitand
bitor	bool	break	case	catch	char	char16_t
char32_t	class	compl	const	constexpr	const_cast	continue
decltype	default	delete	do	double	dynamic_cast	else
enum	explicit	export	extern	false	float	for
friend	goto	if	inline	int	long	mutable
namespace	new	noexcept	not	not_eq	nullptr	operator
or	or_eq	private	protected	public	register	reinterpret_cast
return	short	signed	sizeof	static	static_assert	static_cast
struct	switch	template	this	thread_local	throw	true
try	typedef	typeid	typename	union	unsigned	
using	virtual	void	volatile	wchar_t	while	xor
xor_eq						

變數可以在任何地方加以宣告，只要在第一次使用前完成宣告即可。

```
using namespace std;
#include <iostream>

int main ()
{
   double a;

   cout << "Hello, this is a test program." << endl;

   cout << "Type parameter a: ";
   cin >> a;
```

```
   a = (a + 1) / 2;

   double c;

   c = a * 5 + 1;

   cout << "c contains      : " << c << endl;

   int i, j;

   i = 0;
   j = i + 1;

   cout << "j contains      : " << j << endl;

   return 0;
}
```

我們可以利用這個C++的特性，將程式寫的更有彈性。請參考下面這個程式：

```
using namespace std;
#include <iostream>

int main ()
{
   double a;

   cout << "Type a number: ";
   cin >> a;

   {
      int a = 1;
      a = a * 10 + 4;
      cout << "Local number: " << a << endl;
   }

   cout << "You typed: " << a << endl;

   return 0;
}
```

此外C++還允許我們在宣告變數時，使用別的變數做為初始值設定的一部份，請參考下面這個程式：

```
#include <iostream>
using namespace std;

int main ()
```

```
{
   double a = 12 * 3.25;
   double b = a + 1.112;

   cout << "a contains: " << a << endl;
   cout << "b contains: " << b << endl;

   a = a * 2 + b;

   double c = a + b * a;

   cout << "c contains: " << c << endl;

   return 0;
}
```

我們也可以在迴圈中宣告變數，其生命週期就僅限於迴圈內，請參考下面的程式：

```
using namespace std;
#include <iostream>

int main ()
{
   int i;                       // Simple declaration of i
   i = 487;

   for (int i = 0; i < 4; i++)  // Local declaration of i
   {
      cout << i << endl;        // This outputs 0, 1, 2 and 3
   }

   cout << i << endl;           // This outputs 487

   return 0;
}
```

若在區域內的變數與某個全域變數名稱相同，只要加上 “ :: “ 就可以在區域內使用，請參考下面的程式：

```
#include <iostream>
using namespace std;

double a = 128;

int main ()
{
   double a = 256;
```

```
    cout << "Local a:  " << a   << endl;
    cout << "Global a: " << ::a << endl;

    return 0;
}
```

3.2 常數

- 以const宣告之變數，其值不能被變更，稱為常數。例如：

```
const double radius = 5.5;
```

- 也可以用#define這一個preprocessor directive來定義常數。例如:

```
#define PI 3.1415926
```

3.3 新的初始值給定方式

C++提供了新的變數初始值給定的方法,

```
type valName = value;
type valName(value);
type valName = {value};
type valName{value};
```

其中最後一種宣告的方法，是在C++11的標準才開始提供，所以在編譯時必須要使用「-std=gnu++11」的選項才能順利的編譯。請參考下面的例子：

```
using namespace std;
#include <iostream>

int main()
{
  int i = 3;
  int j (4);
  int k = {5};
  int l {6};

  cout << "i=" << i << endl;
  cout << "j=" << j << endl;
  cout << "k=" << k << endl;
  cout << "l=" << l << endl;
```

```
  return 0;
}
```

上述這個程式的編譯與執行畫面如下：

```
[03:39 user@ws ch3]$ g++ -std=gnu++11 newVarInitial.cpp
[03:39 user@ws ch3]$ ./a.out
i=3
j=4
k=5
l=6
[03:39 user@ws ch3]$
```

新的宣告方式，還提供了型態安全的檢查，我們將上面的程式修改如下：

```
using namespace std;
#include <iostream>

int main()
{
  int i = 3.5;
  int j (4.5);
  int k = {5.5};
  int l {6.5};
  char x {332};

  cout << "i=" << i << endl;
  cout << "j=" << j << endl;
  cout << "k=" << k << endl;
  cout << "l=" << l << endl;

  return 0;
}
```

其編譯結果如下：

```
[03:52 user@ws ch3]$ g++ -std=gnu++11 newVarInitial2.cpp
newVarInitial2.cpp: In function ‘int main()’:
newVarInitial2.cpp:8:15: warning: narrowing conversion of ‘5.5e+0’ from
‘double’ to ‘int’ inside { } [-Wnarrowing]
newVarInitial2.cpp:9:13: warning: narrowing conversion of ‘6.5e+0’ from
‘double’ to ‘int’ inside { } [-Wnarrowing]
newVarInitial2.cpp:10:14: warning: narrowing conversion of ‘332’ from ‘int’
to ‘char’ inside { } [-Wnarrowing]
```

```
newVarInitial2.cpp:10:14: warning: overflow in implicit constant conversion
[-Woverflow]
[03:53 user@ws ch3]$ ./a.out
i=3
j=4
k=5
l=6
[03:53 user@ws ch3]$
```

注意到了嗎？新的「{}」方式還會進行型態安全的檢查。我們將「{}」這種宣告初始值的方法稱為List Initialization。當「{}」內沒有提供初始值時，編譯器會以0做為初始值。

最後C++還提供了一種新的宣告方式：

```
auto valName = value;
auto valName (value);
auto valName = {value};
auto valName {value};
```

使用auto是讓編譯器視變數的初始值，自動決定適切的資料型態。