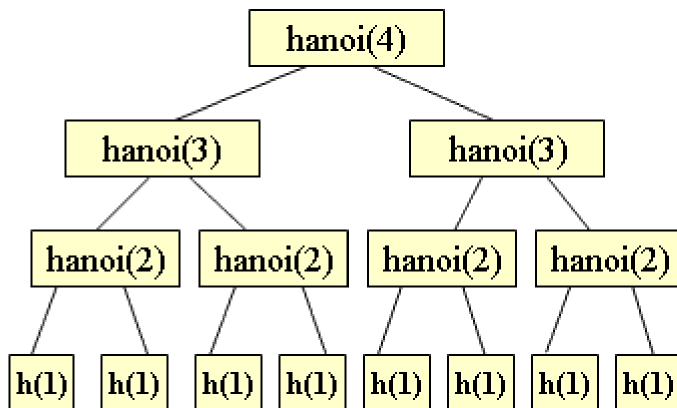
```
    }
    return 0;
}
```

خروجی برنامه با فرض اینکه می خواهیم مراحل انتقال چهار دیسک را ببینیم به صورت زیر می باشد :

```
Moving disks form tower A to C.
How many disks do you want to move?4
Disk 1 from tower A to tower B
Disk 2 from tower A to tower C
Disk 1 from tower B to tower C
Disk 3 from tower A to tower B
Disk 1 from tower C to tower A
Disk 2 from tower C to tower B
Disk 1 from tower A to tower B
Disk 4 from tower A to tower C
Disk 1 from tower B to tower C
Disk 2 from tower B to tower A
Disk 1 from tower C to tower A
Disk 3 from tower B to tower C
Disk 1 from tower A to tower B
Disk 2 from tower A to tower C
Disk 1 from tower B to tower C
0
```

روال فراخوانی تابع هانوی به صورت شکل زیر می باشد:



تولید اعداد تصادفی

یکی از کاربردهای اساسی کامپیوتر، استفاده از آن در کارهای شبیه سازی می باشد. در اینجا به بررسی نحوه تولید اعداد تصادفی می پردازیم. اعداد تصادفی در مسائل شبیه سازی کاربرد فراوانی دارند، به عنوان مثال شبیه سازی پرتاب سکه، پرتاب تاس و مسائلی از این قبیل.

برای تولید اعداد تصادفی زبان ++C تابعی با نام **()rand** را که در فایل کتابخانه ای **stdlib.h** قرار دارد، در اختیار ما گذاشته است. به عنوان مثال دستور زیر :

```
i = rand();
```

یک عدد تصادفی بین **۱** تا **۳۲۷۶۷** را در متغیر **i** قرار می دهد . تابع **()rand** اعداد را با احتمال مساوی در این بازه انتخاب می کند پس شانس انتخاب هر عددی در این بازه با اعداد دیگر برابر است.

معمولاً بازه اعدادی که توسط تابع **rand** تولید می شود، با آنچه که مورد نیاز ماست متفاوت می باشد. به عنوان مثال برای شبیه سازی پرتاب سکه به دو عدد تصادفی و برای تاس به شش عدد تصادفی نیاز داریم. فرض کنید که می خواهید عدد **۳۱۲۵۰** را به عددی بین **۱** تا **۶** تبدیل کنید. چه راه کاری را در نظر می گیرید؟ راهی که برای این تبدیل وجود دارد استفاده از باقیمانده صحیح می باشد، همانطور که می دانید باقیمانده صحیح تقسیم هر عددی بر **۶** یکی از اعداد **۰** تا **۵** می باشد. پس با اضافه کردن **۱** واحد به باقیمانده ، عددی بین **۱** تا **۶** خواهیم داشت. به عنوان مثال در کادر زیر عددی بین **۱** تا **۳۲۷۶۷** وارد کنید و با کلیک بر روی دکمه محاسبه خروجی ، نتیجه محاسبه را ببینید :

a= 31250
a%6+1= [ ] محاسبه خروجي

حال اگر به جای متغیر a عبارت **()rand** را قرار دهیم عبارت **rand()%6+1** ، تابع عددی تصادفی بین **۱** تا **۶** به ما می دهد به طور کلی برای بدست آوردن عددی تصادفی در بازه **[a,b]** می توانیم از فرمول زیر استفاده کنیم.

```
rand()%(b-a+1)+a
```

به عنوان مثال خروجی قطعه برنامه زیر عدد صحیحی در بازه [۲۰،۱۰۰] می باشد.

```
int a = 20 , b = 100;
cout<< rand()%(b-a+1)+a;
```

برنامه زیر **۲۰** عدد تصادفی بین **۱** تا **۶** را ایجاد می کند. که این برنامه را می توان **۲۰** بار پرتاب یک تاس در نظر گرفت :

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

```
#include <stdlib.h>

int main()
{
   for (int i = 1; i<= 20; i++ )
    {
     cout << rand() % 6 + 1<<"\t";

     if ( i % 5 == 0 )
        cout << endl;
    }
   return 0;
}
```

خروجی برنامه فوق به صورت زیر می باشد :

```
5       5       3       5       5
2       4       2       5       5
5       3       2       2       1
5       1       4       6       4
```

یک بار دیگر برنامه فوق را اجرا کنید و خروجی را مجدداً بررسی کنید. خواهید دید خروجی دقیقاً همان اعداد قبلی می باشد. خروجی تابع **rand()** اعداد تصادفی می باشد ولی با اجرای دوباره برنامه همان اعداد مجدداً به همان ترتیب قبلی تکرار می شوند. این تکرار یکی از قابلیتهای تابع می باشد ودر اشکال زدایی برنامه کاربرد دارد.

اگر بخواهیم که تابع **rand()** اعداد کاملاً تصادفی ایجاد کند باید از تابع **srand()** استفاده کنیم. این تابع ورودی از نوع اعداد صحیح بدون علامت می گیرد و باعث تصادفی شدن تابع **rand()** بر اساس مقدار ورودی تابع **srand()** می شود. تابع**srand()** نیز در فایل کتابخانه ای **stdlib.h** قرار دارد. در برنامه زیر به نحوه استفاده از تابع **srand()** پی خواهید برد.

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

int main()
{
  unsigned int num;

  cout<<"Enter a number: ";
  cin>>num;

  srand(num);
```

```
  for (int i = 1; i<= 20; i++ )
   {
    cout << rand() % 6 + 1<<"\t";

    if ( i % 5 == 0 )
      cout << endl;
   }
  return 0;
}
```

خروجی برنامه به صورت زیر می باشد.

```
Enter a number: 251
3       4       1       4       6
6       4       6       2       5
5       3       1       4       5
1       6       6       6       1
Enter a number: 350
1       4       3       4       1
2       6       2       6       2
4       2       5       3       5
4       4       5       2       3
Enter a number: 251
3       4       1       4       6
6       4       6       2       5
5       3       1       4       5
1       6       6       6       1
```

همانطور که می بینید بر اساس ورودی های مختلف خروجی نیز تغییر می کند. توجه داشته باشید که ورودی های یکسان خروجی های یکسانی دارند.

اگر بخواهیم بدون نیاز به وارد کردن عددی توسط کاربر، اعداد تصادفی داشته باشیم می توانیم از تابع **time** که در فایل کتابخانه ای **time.h** قرار دارد استفاده کنیم . تابع **time** ساعت کامپیوتر را می خواند و زمان را بر حسب ثانیه بر می گرداند ، به این ترتیب دستور زیر:

```
srand(time(0));
```

باعث می شود که تابع **rand()** در هر بار اجرای برنامه اعداد متفاوتی را ایجاد کند. اگر برنامه فوق را به صورت زیر باز نویسی کنیم با هر بار اجرای برنامه اعداد تصادفی متفاوتی خواهیم داشت.

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

```
#include <stdlib.h>
#include <time.h>

int main()
{
  srand(time(0));

  for (int i = 1; i<= 20; i++ )
   {
    cout << rand() % 6 + 1<<"\t";

    if ( i % 5 == 0 )
      cout << endl;
   }
  return 0;
}
```

مثال : برنامه ای بنویسید که پرتاب سکه ای را شبیه سازی کند ، بدین صورت که سکه را **۲۰۰۰** بار پرتاب کند و دفعات رو یا پشت آمدن سکه را چاپ کند.

```
#include <iostream.h>
#include <stdlib.h>
#include <time.h>

int main()
{
  int back=0,front=0,face;

  srand(time(0));

  for (int i = 1; i<= 2000; i++ )
    {
      face=rand()%2+1;
      switch(face)
        {
          case 1:
            ++back;
            break;
          case 2:
            ++front;
            break;
          default:
            cout<<"Error!";
        }
    }
```

```
  cout<<"Front: "<< front<<endl;
  cout<<"Back : "<< back<<endl;

  return 0;
}
```

نوع داده ای enum

enum یک نوع داده ای تعریف شده توسط برنامه نویس را که به آن نوع داده شمارش می گویند، ایجاد می کند. نحوه ایجاد یک نوع داده شمارش به صورت زیر می باشد.

```
enum   نام نوع داده{ثابت ۱ و ثابت ۲ و ... و ثابتn}
```

این دستور به ترتیب در ثابت ۱ ، ثابت ۲ و ... و ثابتn اعداد صحیح متوالی ۰ تا **n** را قرار می دهد . به صورت پیش فرض مقداردهی متغیرها در این دستور از صفر شروع می شود.

```
enum TrueFalse {FALSE , TRUE}
```

دستور فوق به ثابت **FALSE** ، عدد **صفر** و به ثابت **TRUE** عدد **۱** را تخصیص می دهد. حال اگر بخواهیم مقداردهی از عددی غیر از صفر شروع شود باید عدد مورد نظر را مشخص کنیم :

```
enum Days {SAT = 1, SUN, MON, TUE, WED, THU, FRI}
```

دستور فوق به روزهای هفته به ترتیب اعداد **۱** تا **۷** را نسبت می دهد. توصیه می شود که نام ثابت های شمارشی را با حروف بزرگ بنویسید ، بدین صورت این ثابتها با متغیرهای برنامه ، اشتباه نمی شوند. ضمناً **enum** را در ابتدای برنامه به کار ببرید.

در حقیقت این نوع داده به هر یک از موارد لیستی از اعداد نامی را نسبت می دهد. به عنوان نمونه در مثال روزهای هفته هر یک از اعداد **۱** تا **۷** را با یکی از روزهای هفته نام گذاری کردیم.

مقدار دهی موارد لیست به صورت های مختلف امکان پذیر می باشد.

```
enum Days { MON, TUE, WED, THU, FRI, SAT , SUN = 0}
```

دستور فوق **SUN** را با عدد **صفر** و **SAT** را با عدد ۱- و ... و **MON** را با عدد **-6** مقدار دهی می کند.

```
enum Colors {BLACK = 2, GREEN = 4, RED = 3,
             BLUE = 5, GRAY,WHITE = 0}
```

در دستور فوق هر یک از موارد با عدد نسبت داده شده مقدار دهی می شوند و **GRAY** با عدد ۶ مقدار دهی می شود چون بعد از **BLUE = 5** آمده است.

به محض ساخته شدن لیست ، نوع داده نوشته شده توسط برنامه نویس قابل استفاده می گردد و می توان متغیرهایی را از نوع داده نوشته شده توسط برنامه نویس به همان شیوه ای که سایر متغیرها را تعریف می کردیم، تعریف کرد. به عنوان مثال :

```
TrueFalse  tf1,tf2;
Days       day1, day2 = SUN;
Colors     color1 = BLACK , color2 = GRAY;
```

همچنین متغیرها را می توان هنگام ایجاد نوع داده، تعریف کرد. به عنوان مثال :

```
TrueFalse {FALSE, TRUE} tf1 ,tf2;
```

نکته : تبدیل داده ای از نوع **enum** به عدد صحیح مجاز می باشد ولی بر عکس این عمل غیر مجاز است. به عنوان مثال :

```
enum MyEnum {ALPHA, BETA, GAMMA};
int   i = BETA;
int   j = 3+GAMMA;
```

دستورات فوق مجاز می باشند، و این دستورات عدد ۱ را در **i** و ۵ را در **j** قرار می دهند.

```
enum MyEnum {ALPHA, BETA, GAMMA};
MyEnum  x = 2;
MyEnum  y = 123;
```

ولی دستورات فوق غیر مجاز می باشند. البته بعضی از کامپایلرها این موضوع را نادیده می گیرند و تنها یک پیغام اخطار می دهند ولی توصیه می شود که برای پیشگیری از وقوع خطاهای منطقی در برنامه از به کار بردن دستوراتی مانند کدهای فوق خودداری کنید.

برنامه زیر نحوه کاربرد نوع داده **enum** را نشان می دهد.

```
#include <iostream.h>
int main()
{
  enum PizzaSize{SMALL,MEDIUM,LARGE,EXTRALARGE};
  PizzaSize size;
  size=LARGE;

  cout<<"The small pizza has a value of "<<SMALL;
  cout<<"\nThe medium pizza has a value of "<<MEDIUM;
  cout<<"\nThe large pizza has a value of "<<size;

  return 0;
}
```

خروجی برنامه به صورت زیر می باشد:

```
The small pizza has a value of 0
The medium pizza has a value of 1
The large pizza has a value of 2
```

توابع بدون خروجی و آرگومان

در برنامه نویسی به توابعی نیاز پیدا می کنیم که نیاز ندارند چیزی را به عنوان خروجی تابع برگردانند و یا توابعی که نیاز به آرگومان و ورودی ندارند ویا هر دو. زبان ++C برای امکان استفاده از چنین توابعی، کلمه **void** را در اختیار ما قرار داده است. اگر بخواهیم تابعی بدون خروجی ایجاد کنیم کافی است به جای نوع داده خروجی تابع کلمه **void** را قرار دهیم. به تابع زیر توجه کنید.

```
void function (int num)
{
   cout << "My input is" << num << endl;
}
```

همانطور که می بینید این تابع نیازی به استفاده از دستور **return** ندارد چون قرار نیست چیزی را به عنوان خروجی تابع برگرداند. تابع فوق بر اساس مقدار داده ورودی، پیغامی را بر روی صفحه نمایش چاپ می کند.

حال که با **void** آشنا شدید می توانیم از این به بعد تابع **main** را از نوع **void** تعریف کنیم. در این صورت دیگر نیازی به استفاده از دستور **;return 0**در انتهای برنامه نداریم :

```
void main()
{
    دستورات برنامه
}
```

به عنوان مثال برنامه برج هانوی را که در مبحث توابع بازگشتی نوشتیم با استفاده از نوع **void** بازنویسی می کنیم.

```
#include <iostream.h>

void hanoi(int, char, char, char);

void main( )
{
   cout<<"Moving 4 disks form tower A to C."<<endl;
   hanoi(4,'A','B','C');
}

void hanoi(int n, char first, char help, char second) {
   if (n == 1) {
       cout << "Disk " << n << " from tower " << first
            << " to tower " << second << endl;
   } else {
       hanoi(n-1, first, second, help);
       cout << "Disk " << n << " from tower " << first
            << " to tower " << second << endl;
       hanoi(n-1, help, first, second);
   }
}
```

همانطور که در برنامه فوق می بینید تابع **hanoi** بدون دستور **return** نوشته شده است، زیرا نوع تابع **void** می باشد، یعنی تابع بدون خروجی است و توابع بدون خروجی را می توانیم مستقیماً همانند برنامه فوق فراخوانی کنید. یعنی کافی است نام تابع را همراه آرگومانهای مورد نظر بنویسیم.

برای ایجاد توابع بدون آرگومان می توانید در پرانتز تابع کلمه **void** را بنویسید یا اینکه این پرانتز را خالی گذاشته و در آن چیزی ننویسید.

```
void function1();
int  function2(void);
```

در دو دستور فوق تابع **function1** از نوع توابع بدون خروجی و بدون آرگومان ایجاد می شود. و تابع **function2** از نوع توابع بدون آرگومان و با خروجی از نوع عدد صحیح می باشد، برای آشنایی با نحوه کاربرد توابع بدون آرگومان به برنامه زیر توجه کنید :

```
#include <iostream.h>

int function(void);

void main( )
{
   int num, counter = 0;
   float average, sum = 0;

   num=function();

   while (num != -1){
      sum += num ;
      ++counter;
      num=function();
   }

   if (counter != 0){
      average = sum / counter;
      cout << "The average is " << average << endl;
   }
   else
      cout << "No numbers were entered." << endl;
}

int function(void){
  int x;
  cout << "Enter a number (-1 to end):";
  cin >>x;
  return x;
}
```

همانطور که در برنامه فوق مشاهده می کنید تابع **function** از نوع توابع بدون آرگومان می باشد و دارای خروجی صحیح می باشد. این تابع در برنامه دو بار فراخوانی شده است و وظیفه این تابع دریافت متغیری از صفحه کلید و برگرداندن آن متغیر به عنوان خروجی تابع می باشدو دستور **num=function()** عدد دریافت شده از صفحه کلید را در متغیر **num** قرار می دهد اگر به یاد داشته باشید این برنامه قبلاً بدون استفاده از تابع در مبحث ساختار تکرار while نوشته بودیم. برای درک بهتر این برنامه توصیه می شود آن را با برنامه موجود در مبحث ساختار تکرار while مقایسه کنید، و متوجه خواهید شد که تابع **function** ما را از دوباره نویسی بعضی از دستورات بی نیاز کرده است و نیز برنامه خلاصه تر و مفهوم تر شده است.

قوانین حوزه

قسمتی از برنامه که در آن متغیری تعریف شده و قابل استفاده می باشد، حوزه آن متغیر گفته می شود. در زبان C++ به قسمتی از برنامه که با یک علامت ({) شروع شده و با علامت (}) به پایان می رسد یک بلوک می گویند. به عنوان مثال هنگامی که متغیری را در یک بلوک تعریف می کنیم، متغیر فقط در آن بلوک قابل دسترسی می باشد ولذا حوزه آن متغیر بلوکی که در آن تعریف شده است ، می باشد. به مثال زیر توجه کنید :

```
#include <iostream.h>

void main( )
{
   {
     int x= 1;
     cout << x;
   }
   cout << x;
}
```

اگر برنامه فوق را بنویسیم و بخواهیم اجرا کنیم پیغام خطای **Undefined symbol 'x'** را دریافت خواهیم کرد ودلیل این امر این است که متغیر **x** فقط در بلوک درونی تابع **main** تعریف شده است، لذا در خود تابع قابل دسترسی نمی باشد. در این مبحث به بررسی حوزه تابع، حوزه فایل و حوزه بلوک می پردازیم.

متغیری که خارج از همه توابع تعریف می شود، دارای حوزه فایل می باشد و چنین متغیری برای تمام توابع، شناخته شده وقابل استفاده می باشد. به مثال زیر توجه کنید:

```
#include <iostream.h>

int x=1;
int f();

void main( )
{
   cout << x;
   cout << f();
   cout << x;
}
int f(){
  return 2*x;
}
```

متغیر **x** دارای حوزه فایل می باشد. لذا در تابع **main** و تابع **f** قابل استفاده می باشد. خروجی برنامه فوق به صورت زیر می باشد.

```
121
```

متغیری که درون توابع و یا به عنوان آرگومان تابع تعریف می گردد، دارای حوزه تابع می باشد و از نوع متغیرهای محلی است و خارج ازتابع قابل استفاده و دسترسی نمی باشد. توابعی که تا کنون نوشتیم ومتغیرهایی که در آن ها تعریف کردیم، همگی دارای حوزه تابع بودند. ضمنا این متغیرها، هنگامی که برنامه از آن تابع خارج می شود، مقادیر خود را از دست می دهند. حال اگر بخواهیم یک متغیر محلی تابع، مقدار خود را حفظ کرده و برای دفعات بعدی فراخوانی تابع نیز نگه دارد، زبان C++ کلمه **static** را در اختیار ما قرار داده است. کلمه **static** را باید قبل از نوع متغیر قرار دهیم. مانند:

```
static int x=1;
```

دستور فوق متغیر **x** را از نوع عدد صحیح تعریف می کند و این متغیر با اولین فراخوانی تابع مقدار دهی می شود و در دفعات بعدی فراخوانی تابع مقدار قبلی خود را حفظ می کند. به مثال زیر توجه کنید:

```
#include <iostream.h>

int f();

void main( )
{
   cout << f();
   cout << f();
   cout << f();
}
int f(){
  static int x=0;
  x++;
  return x;
}
```

خروجی برنامه فوق به صورت زیر می باشد:

```
123
```

برنامه با اولین فراخوانی تابع **f** به متغیر محلی **x** مقدار ۰ را می دهد، سپس به **x** یک واحد اضافه می شود و به عنوان خروجی برگردانده می شود. پس ابتدا عدد**۱** چاپ می گردد. در بار دوم فراخوانی تابع **f** ، متغیر **x** دوباره مقداردهی نمی شود، بلکه به مقدار قبلی آن که عدد **۱** است، یک واحد اضافه گشته و به عنوان خروجی برگردانده می شود. پس این بار عدد **۲**

چاپ می گردد و در نهایت با فراخوانی تابع **f** برای بار سوم عدد **۳** چاپ خواهد شد. اگر برنامه فوق را بدون کلمه **static** بنویسیم، خروجی **۱۱۱** خواهد بود.

یکی از نکاتی که می توان در قوانین حوزه بررسی کرد، متغیرهای همنام در بلوک های تو در تو می باشد. به مثال زیر توجه کنید:

```
#include <iostream.h>

void main( )
{
   int x=1;
   cout << x;
   {
     int x= 2;
     cout << x;
   }
   cout << x;
}
```

در برنامه فوق متغیر **x** یک بار در تابع **main** تعریف شده است و با دیگر در بلوک درونی. خروجی برنامه به صورت زیر می باشد:

```
121
```

هنگامی که در بلوک درونی متغیری با نام **x** را مجددا تعریف می کنیم، متغیر خارج از بلوک از دید برنامه پنهان می گردد و تنها متغیر داخل بلوک قابل استفاده می شود. همچنین هنگامی که برنامه از بلوک خارج می گردد، متغیر **x** بیرونی دوباره قابل استفاده می گردد. ضمنا توجه داشته باشید که مقدار متغیر x تابع **main** تغییری نکرده است، یعنی با وجود استفاده از متغیری همنام و نیز مقداردهی آن، تاثیری روی متغیر **x** تابع ایجاد نشده است. چون حوزه متغیر **x** بلوک درونی تنها داخل آن بلوک می باشد.

برنامه زیر تمام موارد ذکر شده در این مبحث را شامل می شود. بررسی آن و خروجی برنامه شما را در فهم بهتر این مبحث یاری می نماید.

```
#include <iostream.h>

void useLocal( void ); // function prototype
void useStaticLocal( void ); // function prototype
void useGlobal( void ); // function prototype

int x = 1; // global variable
```

```
void main()
{
   int x = 5; // local variable to main

   cout <<"local x in main's outer scope is "<<x<<endl;

   { // start new scope

     int x = 7;

     cout <<"local x in main's inner scope is "<<x<<endl;

   } // end new scope

   cout <<"local x in main's outer scope is "<<x<< endl;

 useLocal(); //useLocal has local x
 useStaticLocal(); //useStaticLocal has static local x
 useGlobal(); //useGlobal uses global x
 useLocal(); //useLocal reinitializes its local x
 useStaticLocal();//static local x retains its prior value
 useGlobal(); //global x also retains its value

   cout << "\nlocal x in main is " << x << endl;

} // end main

//useLocal reinitializes local variable x during each call
void useLocal( void )
{
   int x = 25; //initialized each time useLocal is called

   cout << endl << "local x is " << x
        << " on entering useLocal" << endl;
   ++x;
   cout << "local x is " << x
        << " on exiting useLocal" << endl;

} // end function useLocal

// useStaticLocal initializes static local variable x
// only the first time the function is called; value
// of x is saved between calls to this function
void useStaticLocal( void )
{
// initialized first time useStaticLocal is called.
```

```
   static int x = 50;

   cout << endl << "local static x is " << x
        << " on entering useStaticLocal" << endl;
   ++x;
   cout << "local static x is " << x
        << " on exiting useStaticLocal" << endl;

} // end function useStaticLocal

// useGlobal modifies global variable x during each call
void useGlobal( void )
{
   cout << endl << "global x is " << x
        << " on entering useGlobal" << endl;
   x *= 10;
   cout << "global x is " << x
        << " on exiting useGlobal" << endl;

} // end function useGlobal
```

خروجی برنامه به صورت زیر می باشد:

```
local x in main's outer scope is 5
local x in main's inner scope is 7
local x in main's outer scope is 5

local x is 25 on entering useLocal
local x is 26 on exiting useLocal

local static x is 50 on entering useStaticLocal
local static x is 51 on exiting useStaticLocal

global x is 1 on entering useGlobal
global x is 10 on exiting useGlobal

local x is 25 on entering useLocal
local x is 26 on exiting useLocal

local static x is 51 on entering useStaticLocal
local static x is 52 on exiting useStaticLocal

global x is 10 on entering useGlobal
global x is 100 on exiting useGlobal
```