## 作业2.1

题目：重写POW和f，使得当有一个溢出时产生一个错误信号，而不是默默地工作在模运算中。你可以使用声明 error("Overflow");来标志一个溢出。

**核心函数代码：2.1\_solution.c0**

#use <util>

int POW(int x, int y)

//@requires y >= 0;

{

if (y == 0)

return 1;

else

{

int a = POW(x, y-1);

int b = x\*a;

if(b/x!= a)

error("Overflow");

else

return b;

}

}

int f(int x, int y)

//@requires y >= 0;

//@ensures \result == POW(x, y);

{

int b = x;

int e = y;

int ans = 1;

while (e > 0)

//@loop\_invariant e >= 0;

//@loop\_invariant POW(x, y) == ans \* POW(b, e);

{

if (e % 2 == 1)

{

int temp = b \* ans;

if(temp/b!= ans)

error("Overflow");

ans = temp;

}

b = b \* b;

e = e / 2;

}

//@assert e == 0;

return ans;

}

**测试overflow代码：2.1\_test\_overflow.c0**

#use <conio>

int f(int x, int y);

int main()

{

assert(f(2,32) == 32);

assert(f(-2,32) == 32);

return 0;

}

**测试success代码：2.1\_test\_success.c0**

#use <conio>

int f(int x, int y);

int main()

{

assert(f(2,5) == 32);

assert(f(-2,5) == -32);

assert(f(2,0) == 1);

print("All tests for FUNTION f passed!\n");

return 0;

}

## 作业2.2

题目：为f找到一个输入，使得循环不变量不满足第一个猜测：

//@loop\_invariant r \* POW(b,e) == POW(x,y);

**找不到这样的输入满足条件**

## 作业3.1

**函数定义代码：3.1\_solution.c0**

int rem(int a,int b)

//@requires b!=0;

//@ensures \result==(a%b);

{

int x=a;

int y=b;

if(x==0)

return 0;

else

{

int fx=1;

int fy=1;

int z=0;

if(x<0)

{

x=-x;

fx=-1;

}

if(y<0)

{

y=-y;

fy=-1;

}

for (z = 0; x >= y; z++)

//@loop\_invariant x >= 0 && y > 0 && x+z\*y== fx\*a;

{

x = x - y;

}

x = fx\*x;

int temp=0;

if(x<0) temp=-x;

//@assert temp>=0&&temp<y;

return x;

}

}

**函数测试代码3.1\_test.c0**

#use <conio>

int rem(int x,int y);

int quot(int x,int y);

int main()

{

assert(quot(20,5)==4);

assert(quot(20,6)==3);

assert(quot(-20,5)==-4);

assert(quot(-20,6)==-3);

assert(quot(-20,-5)==4);

assert(quot(-20,-6)==3);

assert(quot(20,-5)==-4);

assert(quot(20,-6)==-3);

print("All tests for FUNTION quot passed!\n");

assert(rem(20,5)==0);

assert(rem(20,6)==2);

assert(rem(-20,5)==0);

assert(rem(-20,6)==-2);

assert(rem(-20,-5)==0);

assert(rem(-20,-6)==-2);

assert(rem(20,-5)==0);

assert(rem(20,-6)==2);

print("All tests for FUNTION rem passed!\n");

assert(quot(20,5)\*5+rem(20,5)==20);

assert(quot(20,6)\*6+rem(20,6)==20);

assert(quot(-20,5)\*5+rem(-20,5)==-20);

assert(quot(-20,6)\*6+rem(-20,6)==-20);

assert(quot(-20,-5)\*(-5)+rem(-20,-5)==-20);

assert(quot(-20,-6)\*(-6)+rem(-20,-6)==-20);

assert(quot(20,-5)\*(-5)+rem(20,-5)==20);

assert(quot(20,-6)\*(-6)+rem(20,-6)==20);

print("All tests for FUNTIONS quot and rem passed!\n");

return 0;

}

## 作业3.2

题目：写一个函数int2hex，返回一个字符串包含一个给定整数十六进制表示字符串。函数原型：string int2hex(int x);

**函数定义代码：3.2\_solution.c0**

#use <string>

#use <parse>

string int2hex(int x)

//

// @ensures (\*(parse\_int(\result, 16))) == x;

{

int i;

char ch;

int mask = 0xf;

int temp = x;

int xnum = 0;

string s= "";

for (i = 0; i < 8; i++)

//@loop\_invariant xnum >= 0 && xnum <= 15;

{

xnum = temp & mask;

if (xnum < 10)

ch = char\_chr(xnum + char\_ord('0'));

else

ch = char\_chr(xnum - 10 + char\_ord('a'));

s = string\_join(string\_fromchar(ch), s);

temp = temp >> 4;

}

//@assert temp == -1 || temp == 0;

return s;

}

测试代码：3.2\_test.c0

#use <conio>

#use <string>

#use <parse>

string int2hex(int x);

int main()

{

bool flag=true;

while(flag)

{

println("Key false to exit or true to continue");

flag = \*(parse\_bool(readline()));

if(flag)

{

println("Input a num");

int a = \*(parse\_int(readline(), 10));

println(int2hex(a));

}

else

{

println("Test is finished");

}

}

return 0;

}

## 作业3.3

题目：写一个函数lsr(逻辑右移)，如同右移(>>)一样，只不过是用0填充高位而不是复制符号位。解释lsr(x,1)在整数用2的补码表示下的含义。

其中，lsr(x,1)表示将x逻辑右移1位，最高位补零。若x>0，则为x除以2；若x=0，右移后不变；若x<0，右移后变为正数。

**函数定义代码：3.3\_solution.c0**

int lsr(int x,int n)

//@requires n > 0 && n < 32;

{

int i;

int mask = 1 << 31;

//@assert mask < 0;

mask = ~mask;

int temp = x;

for (i = 0; i < n; i++)

{

temp = temp >> 1;

temp = temp & mask;

}

//@assert temp >= 0;

return temp;

}

**测试代码：3.3\_test.c0**

#use <conio>

#use <parse>

#use <string>

int lsr(int x,int n);

int main()

{

bool flag=true;

while(flag)

{

println("Key false to exit or true to continue");

flag = \*(parse\_bool(readline()));

if(flag)

{

println("Input a num");

int x = \*(parse\_int(readline(), 10));

println("Input the right shift count");

int n = \*(parse\_int(readline(), 10));

int r=lsr(x,n);

string s= "";

string sign=">>";

s = string\_join(string\_fromint(r), s);

s = string\_join("=", s);

s = string\_join(string\_fromint(n), s);

s = string\_join(sign, s);

s = string\_join(string\_fromint(x), s);

println(s);

}

else

{

println("Test is finished");

}

}

return 0;

}

## 作业6.1

题目：重新编写二分查找函数使区间的上下界都包含在内。确保重写的循环不变量和循环体合适，并证明新的循环不变量的正确性。同样，通过给定一个轮循环都严格减少而且有下界的量来证明循环可终止。

**证明：**

**三个循环不变量如下：**

**//@loop\_invariant lower >= low && lower<=upper && upper<=high;**

**//@loop\_invariant (lower==low||A[lower-1]<x);**

**//@loop\_invariant (upper==high||A[upper]>x);**

1. 在初始情况下，有lower=low和upper=high，同时有前置条件low >= 0 && low <= high && high <= \length(A)，因此三个前置条件都满足。
2. 假设在进入某次循环时，3个条件均成立，则有：

low<=lower<=upper<=n

lower=low或A[lower-1]<x

upper=high或A[upper]>x

lower<upper

在此条件下，我们只需证明下次循环三个条件仍满足。

计算mid = lower + (upper - lower) / 2。

分三种情况讨论：

1. A[mid]=x：

也即找到了x，函数将返回到被调用的地方，无需判断3个循环不变量是否成立，而且lower和upper均未改变，因此仍成立。

1. A[mid]<x：(在前面加~表示新的值)

那么有~lower=mid+1，~upper=upper；

据此仍有low<=~lower<=~upper<=n；

对于第二个不变量，得A[~lower-1]=A[mid]<x，也成立。

对于第三个不变量，~upper==upper，也成立。

1. A[mid]>x：

那么有~lower=lower，~upper=mid；

据此仍有low<=~lower<=~upper<=n；

对于第二个不变量，~lower==lower，成立。

对于第三个不变量，A[~upper]=A[mid]>x，也成立。

**证明完毕。**

**同时，由于每次循环后lower和upper差值越来越小，那么除非我们找到了元素，那么两者之间的差值最后必定为0，从而因为不满足条件约束而终止循环。**

**函数定义代码：6.1\_solution.c0**

bool is\_in(int x, int[ ] A, int lower, int upper)

//@requires 0 <= lower && lower <= upper && upper <= \length(A);

{

for (int i = lower; i < upper; i++)

//@loop\_invariant lower <= i && i <= upper;

{

if (A[i] == x) return true;

}

return false;

}

bool is\_sorted(int[ ] A, int lower, int upper)

//@requires 0 <= lower && lower <= upper && upper <= \length(A);

{

for (int i = lower; i < upper-1; i++)

//@loop\_invariant i >= lower;

{

if (!(A[i] <= A[i+1]))

return false;

}

return true;

}

int binsearch(int x, int[] A, int low, int high)

//@requires low >= 0 && low <= high && high <= \length(A);

//@requires is\_sorted(A, low, high);

/\*@ensures (\result==-1 && !is\_in(x, A, low, high))

|| (\result>=low && \result<high && A[\result]==x);

@\*/

{

int lower = low;

int upper = high;

while (lower < upper)

//@loop\_invariant lower >= low && lower<=upper && upper<=high;

//@loop\_invariant (lower==low||A[lower-1]<x);

//@loop\_invariant (upper==high||A[upper]>x);

{

int mid = lower + (upper - lower) / 2;

//@assert lower<=mid && mid<=upper;

if (A[mid] == x) return mid;

else if (A[mid] < x) lower = mid + 1;

else /\*@assert (A[mid] > x);@\*/

upper = mid;

}

return -1;

}

**测试代码：6.1\_test.c0**

#use <conio>

#use <parse>

#use <string>

int binsearch(int x,int[]A,int low,int high);

int main()

{

int i=0;

bool flag=true;

while(flag)

{

println("Key false to exit or true to continue");

flag = \*(parse\_bool(readline()));

if(flag)

{

println("Input the number you want to search");

int x = \*(parse\_int(readline(), 10));

println("Input the count of numbers in the array");

int count = \*(parse\_int(readline(), 10));

int[]A=alloc\_array(int,count);

println("Input the numbers in the array by lines");

for(i=0;i<count;i++)

{

int temp = \*(parse\_int(readline(), 10));

A[i]=temp;

}

println("Input the low limit");

int low = \*(parse\_int(readline(), 10));

println("Input the high limit");

int high = \*(parse\_int(readline(), 10));

int ans=binsearch(x,A,low,high);

if(ans==-1)

{

println("The num isn't in the array");

}

else

{

print("The order of the num is ");

print(string\_fromint(ans+1));

println(" (The order starts from 1) ");

}

}

else

{

println("Test is finished");

}

}

return 0;

}

## 作业6.2

题目：重写二分查找函数的不变量来使用is\_in( x, A, l , u)，如果存在一个i，使得对于有时，这个函数返回正。is\_in假设，这里n为数组大小。证明新循环不变量，验证它们足够健壮来表明函数的后置条件。

**证明：**

**4个循环不变量如下：**

**//@loop\_invariant lower >=0 && lower<=upper && upper<=n;**

**//@loop\_invariant lower==0||A[lower-1]<x;**

**//@loop\_invariant upper==n||A[upper]>x;**

**//@loop\_invariant is\_in(x, A, 0, n) == is\_in(x, A, lower, upper);**

1. 在初始情况下，有lower=0和upper=n，同时有前置条件n >= 0 &&n<= \length(A)，因此4个前置条件都满足。
2. 假设在进入某次循环时，4个条件均成立，则有：

low<=lower<=upper<=n

lower=low或A[lower-1]<x

upper=high或A[upper]>x

lower<upper

is\_in(x, A, 0, n) == is\_in(x, A, lower, upper)

在此条件下，我们只需证明下次循环4个条件仍满足。

分三种情况讨论：

1. A[mid]=x：

也即找到了x，函数将返回到被调用的地方，无需判断4个循环不变量是否成立，而且lower和upper均未改变，因此仍成立。

1. A[mid]<x：(在前面加~表示新的值)

那么有~lower=mid+1，~upper=upper；

据此仍有low<=~lower<=~upper<=n；

对于第二个不变量，得A[~lower-1]=A[mid]<x，也成立。

对于第三个不变量，~upper==upper，也成立。

对于第四个不变量，由于A[mid]<x，而且数列单增有序，那么使得A[i]==x的i值必定比mid大，则i>=mid+1,

因此得is\_in(x, A, ~lower,~ upper)=is\_in(x,A,mid+1,upper)= is\_in(x, A, 0, n),也成立。

1. A[mid]>x：

那么有~lower=lower，~upper=mid；

据此仍有low<=~lower<=~upper<=n；

对于第二个不变量，~lower==lower，成立。

对于第三个不变量，A[~upper]=A[mid]>x，也成立。

对于第四个不变量，由于A[mid]>x，而且数列单增有序，那么使得A[i]==x的i值必定比mid小，则i<mid,

因此得is\_in(x, A, ~lower,~ upper)=is\_in(x,A,lower,mid)= is\_in(x, A, 0, n),也成立。

**证明完毕。**

若该元素在其中，当还没有找到待查找元素时，该元素必定仍在lower和upper所构成的元素区间中，因此is\_in(x, A, 0, n)与is\_in(x, A, lower, upper)必定相等且为真；若该元素不在数组中，则在lower和upper构成的区间中必定也找不到该元素，因此is\_in(x, A, 0, n)与 is\_in(x, A, lower, upper)必定相等且为假。而函数的后置条件也分为两种情况，一种是在数组中而且其位置必定在0和n之间；一种是不在数组中此时return -1。新的循环不变量恰好是通过判断在数组范围内是否查找到和在给定范围内是否查找到的对应情况进行约束，因此足够健壮来表明函数的后置条件。

**函数定义代码：6.2\_solution.c0**

bool is\_in(int x, int[ ] A, int lower, int upper)

//@requires 0 <= lower && lower <= upper && upper <= \length(A);

{

for (int i = lower; i < upper; i++)

//@loop\_invariant lower <= i && i <= upper;

{

if (A[i] == x) return true;

}

return false;

}

bool is\_sorted(int[ ] A, int lower, int upper)

//@requires 0 <= lower && lower <= upper && upper <= \length(A);

{

for (int i = lower; i < upper-1; i++)

//@loop\_invariant lower <= i;

if (!(A[i] <= A[i+1])) return false;

return true;

}

int binsearch(int x, int[] A, int n)

//@requires n>=0 && n<=\length(A);

//@requires is\_sorted(A, 0, n);

/\*@ensures (\result==-1 && !is\_in(x, A, 0, n))

|| (\result>=0 && \result<n && A[\result]==x);

@\*/

{

int lower = 0;

int upper = n;

while (lower < upper)

//@loop\_invariant lower >=0 && lower<=upper && upper<=n;

//@loop\_invariant lower==0||A[lower-1]<x;

//@loop\_invariant upper==n||A[upper]>x;

//@loop\_invariant is\_in(x, A, 0, n) == is\_in(x, A, lower, upper);

{

int mid = lower + (upper - lower) / 2;

//@assert lower<=mid && mid<=upper;

if (A[mid] == x) return mid;

else if (A[mid] < x) lower = mid + 1;

else //@assert (A[mid] > x);

upper = mid;

}

return -1;

}

**测试代码：6.2\_test.c0**

#use <conio>

#use <parse>

#use <string>

int binsearch(int x, int[] A, int n);

int main()

{

int i=0;

bool flag=true;

while(flag)

{

println("Key false to exit or true to continue");

flag = \*(parse\_bool(readline()));

if(flag)

{

println("Input the number you want to search");

int x = \*(parse\_int(readline(), 10));

println("Input the count of numbers in the array");

int count = \*(parse\_int(readline(), 10));

int[]A=alloc\_array(int,count);

println("Input the numbers in the array by lines");

for(i=0;i<count;i++)

{

int temp = \*(parse\_int(readline(), 10));

A[i]=temp;

}

int ans=binsearch(x,A,count);

if(ans==-1)

{

println("The num isn't in the array");

}

else

{

print("The order of the num is ");

print(string\_fromint(ans+1));

println(" (The order starts from 1) ");

}

}

else

{

println("Test is finished");

}

}

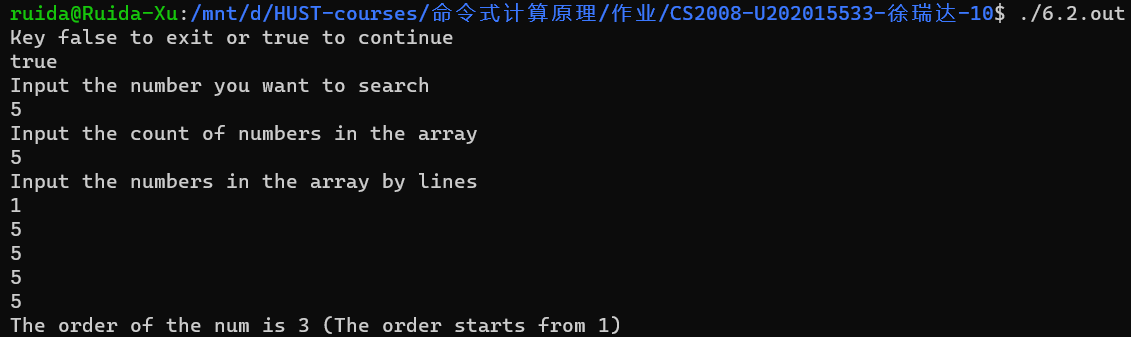
return 0;

}

## 作业6.3

题目：这里提出的二分查找可能找不到数组中最左侧出现的x，因为这种出现可能不是唯一的。举一个例子说明这个问题。现在修改二分查找函数和它的循环不变量，使得它总能找到在给定的数组中，最左侧出现的x（如果它实际在数组里，如果不在则返回-1）。证明新版本的循环不变量和后置条件并验证终止。

分析：假如输入序列1 5 5 5 5在其中查找5，则可以看到6.2的程序查找到的5是序列位置为3的5，而非第一个5。因此在查找到一个5后我们应返回查找直到找到第一个符合条件的元素（最左侧的待查找元素）。也即while(mid > 0 && A[mid-1] == A[mid]) mid--;新版本的循环不变量和后置条件不变，而且增加的步骤仅仅是在A[mid]==x时往左侧查找，upper和lower未改变，因此仍成立。



**函数定义代码：6.3\_solution.c0**

bool is\_in(int x, int[ ] A, int lower, int upper)

//@requires 0 <= lower && lower <= upper && upper <= \length(A);

{

for (int i = lower; i < upper; i++)

//@loop\_invariant lower <= i && i <= upper;

{

if (A[i] == x) return true;

}

return false;

}

bool is\_sorted(int[ ] A, int lower, int upper)

//@requires 0 <= lower && lower <= upper && upper <= \length(A);

{

for (int i = lower; i < upper-1; i++)

//@loop\_invariant lower <= i;

if (!(A[i] <= A[i+1])) return false;

return true;

}

int binsearch(int x, int[] A, int n)

//@requires n>=0 && n<=\length(A);

//@requires is\_sorted(A, 0, n);

/\*@ensures (\result==-1 && !is\_in(x, A, 0, n))

|| (\result>=0 && \result<n && A[\result]==x);

@\*/

{

int lower = 0;

int upper = n;

while (lower < upper)

//@loop\_invariant lower >=0 && lower<=upper && upper<=n;

//@loop\_invariant lower==0||A[lower-1]<x;

//@loop\_invariant upper==n||A[upper]>x;

{

int mid = lower + (upper - lower) / 2;

//@assert lower<=mid && mid<=upper;

if (A[mid] == x) {

while(mid > 0 && A[mid-1] == A[mid])

mid--;

return mid;

}

else if

(A[mid] < x) lower = mid + 1;

else

//@assert (A[mid] > x);

upper = mid;

}

return -1;

}

**测试代码：6.3\_test.c0**

#use <conio>

#use <parse>

#use <string>

int binsearch(int x, int[] A, int n);

int main()

{

int i=0;

bool flag=true;

while(flag)

{

println("Key false to exit or true to continue");

flag = \*(parse\_bool(readline()));

if(flag)

{

println("Input the number you want to search");

int x = \*(parse\_int(readline(), 10));

println("Input the count of numbers in the array");

int count = \*(parse\_int(readline(), 10));

int[]A=alloc\_array(int,count);

println("Input the numbers in the array by lines");

for(i=0;i<count;i++)

{

int temp = \*(parse\_int(readline(), 10));

A[i]=temp;

}

int ans=binsearch(x,A,count);

if(ans==-1)

{

println("The num isn't in the array");

}

else

{

print("The order of the num is ");

print(string\_fromint(ans+1));

println(" (The order starts from 1) ");

}

}

else

{

println("Test is finished");

}

}

return 0;

}

## 作业6.4

题目：如果将中间点的计算方法改为

int mid = (lower + upper)/2;

那约定中的哪一部分会提醒你思维中的瑕疵呢？为什么？给一个在这种情况下显示约定如何失败的例子。

**循环中//@assert lower <= mid && mid < upper;断言会提示错误，当lower+upper溢出时，则有mid<lower，不满足条件。**

## 作业6.5

题目：这一讲中，我们从前面已经验证过的循环不变量，用不变量-设计思想来构建循环体实现。我们用下面的代码来代替整个循环体，进而维护循环不变量

// .... loop\_invariant elided ....  
{  
 lower = lower;

upper = upper;  
 }

证明循环体的循环不变量。这个选择有什么问题呢？我们证明的哪一部分出错，从而表明为什么这个循环体不能正确实现二分查找？

**证明：**

**三个循环不变量如下：**

**//@loop\_invariant lower >= low && lower<=upper && upper<=high;**

**//@loop\_invariant (lower==low||A[lower-1]<x);**

**//@loop\_invariant (upper==high||A[upper]>x);**

1. 在初始情况下，有lower=low和upper=high，同时有前置条件low >= 0 && low <= high && high <= \length(A)，因此三个前置条件都满足。
2. 假设在进入某次循环时，3个条件均成立，则有：

low<=lower<=upper<=n

lower=low或A[lower-1]<x

upper=high或A[upper]>x

lower<upper

在此条件下，我们只需证明下次循环三个条件仍满足。

lower = lower;

upper = upper;

然而，此时循环不变量//@loop\_invariant (lower == 0 || A[lower-1] < x)中下标low-1可能数组越界，所以这个循环体错误。