**VLD** 内存泄漏检测工具

简介：VLD 是一种轻量级的内存泄漏检测工具。

特点：

（1）支持 Linux 和 Windows 平台，支持 C 语言和 C++语言工程；

（2）C/C++中通常利用 malloc/free/new/delete 来申请、释放内存，可以对函数

进行重载来记录用户申请、释放的内存；

(3)用 hash\_table 记录内存申请与释放，高效快捷；

（4）申请一块内存并记录于一个节点，释放一块内存并删除对应节点，能准确

判断程序是否造成内存泄漏，并计算出所泄漏内存空间的大小。

可以检测C/C++ 的内存泄漏问题；

2> 可以检测对象是否析构和因异常引起的内存泄漏问题；

3> 可以检测出在哪个文件的哪一行出现了多少字节的内存泄漏；

4> 以工具形式发布。

**hash\_table.h**

#pragma once

#include<malloc.h>

#include<stdio.h>

#include<assert.h>

typedef struct MemNode

{

const char \*pfile;

size\_t line;

size\_t size;

struct MemNode \*link;

}MemNode;

typedef struct Pair

{

MemNode \*prev;

MemNode \*cur;

}Pair;

#define UL unsigned long

#define HASH\_TABLE\_SIZE 389

MemNode\* hash\_table[HASH\_TABLE\_SIZE] = {0};

MemNode\* create\_node(size\_t sz)

{

MemNode \*p = (MemNode\*)malloc(sz);

assert(p != NULL);

return p;

}

void free\_node(MemNode \*p)

{

free(p);

p = NULL;

}

void fill\_node(MemNode \*n, size\_t sz, const char \*file, size\_t line)

{

n->pfile = file;

n->line = line;

n->size = sz;

n->link = NULL;

}

size\_t Hash(void \*p)

{

return ((UL)p) % HASH\_TABLE\_SIZE;

}

void link\_node(MemNode \*n)

{

size\_t index = Hash(n+1);

if(hash\_table[index] == NULL)

hash\_table[index] = n ;

else

{

n->link = hash\_table[index];

hash\_table[index] = n;

}

}

Pair find\_node(void \*ptr)

{

Pair pr;

size\_t index = Hash(ptr);

if(hash\_table[index] == 0)

{

pr.prev = NULL;

pr.cur = NULL;

}

else

{

MemNode \*p = hash\_table[index];

//处理第一个节点

if(p+1 == ptr)

{

pr.prev = p;

pr.cur = p;

return pr;

}

pr.prev = p;

p = p->link;

pr.cur = p;

while(p != NULL)

{

if(p+1 == ptr)

break;

pr.prev = p;

p = p->link;

pr.cur = p;

}

if(p == NULL)

{

pr.prev = NULL;

pr.cur = NULL;

}

}

return pr;

}

void check\_leak()

{

MemNode \*p = NULL;

int count = 0;

int flag = 0;

int i;

for(i=0; i<HASH\_TABLE\_SIZE; ++i)

{

p = hash\_table[i];

if(p == NULL)

{

flag++;

continue;

}

while(p != NULL)

{

count++;

printf("[ERROR %d] at %p memleak,size: %d,file: %s, line %d\n",

count, p+1,p->size,p->pfile, p->line);

p = p->link;

}

}

if(flag == HASH\_TABLE\_SIZE)

printf("[OK] No Memory Leak.\n");

}

Makefile

all:libvld\_cpp.a libvld\_c.a

libvld\_cpp.a:vld\_cpp.o

ar -cr libvld\_cpp.a vld\_cpp.o

vld\_cpp.o:vld\_cpp.h vld\_cpp.cpp

g++ -o vld\_cpp.o -c vld\_cpp.cpp

libvld\_c.a:vld\_c.o

ar -cr libvld\_c.a vld\_c.o

vld\_c.o:vld\_c.h vld\_c.c

gcc -o vld\_c.o -c vld\_c.c

.PHONY:clean install

clean:

rm \*.a \*.o

install:

./install-sh

vld.h

#include"vld\_c.h"

#include"vld\_cpp.h"

vld\_c.c

#include"hash\_table.h"

#include"vld\_c.h"

void\* \_\_malloc(size\_t sz, const char\* file, size\_t line)

{

void \*result;

size\_t total\_size = sizeof(MemNode) + sz;

MemNode \*p = create\_node(total\_size);

fill\_node(p, sz, file, line);

link\_node(p);

result = p+1;

return result;

}

void \_\_free(void \*ptr)

{

Pair pr = find\_node(ptr);

if(pr.prev==NULL && pr.cur==NULL)

{

printf("%p delete is invalid.\n");

return;

}

MemNode \*p = pr.cur;

size\_t index = Hash(ptr);

//删除节点位第一个节点

if(pr.prev==pr.cur && pr.prev!=NULL)

hash\_table[index] = p->link;

else

pr.prev->link = p->link;

free\_node(p);

}

vld\_c.h

#pragma once

#include<stdio.h>

void\* \_\_malloc(size\_t sz, const char\* file, size\_t line);

void \_\_free(void \*ptr);

void check\_leak();

#define malloc(sz) \_\_malloc(sz,\_\_FILE\_\_,\_\_LINE\_\_)

#define free(p) \_\_free(p)

vld\_cpp.cpp

#include<stdio.h>

#include<assert.h>

#include<malloc.h>

#include"hash\_table.h"

void\* operator new(size\_t sz, const char \*file, size\_t line)

{

void \*result;

size\_t total\_size = sizeof(MemNode) + sz;

MemNode \*p = create\_node(total\_size);

fill\_node(p, sz, file, line);

link\_node(p);

result = p+1;

return result;

}

void operator delete(void \*ptr)

{

Pair pr = find\_node(ptr);

if(pr.prev==NULL && pr.cur==NULL)

{

printf("%p delete is invalid.\n");

return;

}

MemNode \*p = pr.cur;

size\_t index = Hash(ptr);

//删除节点位第一个节点

if(pr.prev==pr.cur && pr.prev!=NULL)

hash\_table[index] = p->link;

else

pr.prev->link = p->link;

free\_node(p);

}

void\* operator new[](size\_t sz, const char \*file, size\_t line)

{

return ::operator new(sz, file, line);

}

void operator delete[](void \*ptr)

{

::operator delete(ptr);

}

///////////////////////////////////////////////////////////////////////////////

void\* \_\_malloc(size\_t sz, const char \*file, size\_t line)

{

return ::operator new(sz, file, line);

}

void \_\_free(void \*ptr)

{

::operator delete(ptr);

}

///////////////////////////////////////////////////////////////////////////////

class VLD

{

public:

VLD(){}

~VLD(){check\_leak();}

};

static VLD vld;

vld\_cpp.h

#pragma once

#include<iostream>

#include<new>

using namespace std;

void\* operator new(size\_t sz, const char \*file, size\_t line);

void operator delete(void \*ptr);

void\* operator new[](size\_t sz, const char \*file, size\_t line);

void operator delete[](void \*ptr);

//void\* malloc(size\_t sz, const char \*file, size\_t line);

//void free(void \*ptr);

void\* \_\_malloc(size\_t sz, const char \*file, size\_t line);

void \_\_free(void \*ptr);

#define new new(\_\_FILE\_\_,\_\_LINE\_\_)

#define malloc(sz) \_\_malloc(sz, \_\_FILE\_\_, \_\_LINE\_\_)

#define free(p) \_\_free(p)