Systems Programming Laboratory, Spring 2022

Introduction to valgrind

Abhijit Das
Arobinda Gupta

Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur

February 1, 2022

What is valgrind?

- Valgrind is not value grinder. It is the gate to Valhalla (Hall of the Slain).
- Our valgrind is a memory debugging tool.
- Capable of detecting many common memory-related errors and problems.
- Also used for various memory-related profiling.
- Here, we use only the memcheck feature of valgrind.
  - Memory leaks
  - Invalid memory access

How to use valgrind

- Run as: **`valgrind executable <command line options>`**
- Example: **`valgrind ./a.out 2022 -name "Sad Tijihba"`**
- Sample valgrind output

```
==4825== Memcheck, a memory error detector
==4825== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==4825== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==4825== Command: ./a.out
==4825==

Input/Output of your program

==4825==
==4825== HEAP SUMMARY:
==4825==     in use at exit: 0 bytes in 0 blocks
==4825==   total heap usage: 23 allocs, 23 frees, 1,376 bytes allocated
==4825==
==4825== All heap blocks were freed -- no leaks are possible
==4825==
==4825== For lists of detected and suppressed errors, rerun with: -s
==4825== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

- valgrind can also issue error/warning messages inside your program's transcript.

Common memory issues

Memory leaks

- Memory is allocated in blocks by malloc, calloc, realloc.
- Allocated memory should be free'd when no longer in use.
- Types of memory allocated but not free'd.

Reachable	Blocks not free'd but can be accessed until the end of the program.
Definitely lost	Blocks not free'd and not accessible by any pointer.
Indirectly lost	Blocks not free'd and accessible only by pointers in other lost blocks.
Possibly lost	Blocks not free'd and accessible only by pointers in their interiors.

Invalid memory access

- Read/write a memory location outside the allocated area of an array.

An implementation of sorted linked lists

The node data type

```
typedef struct _node {
   int data;
   struct _node *next;
} node;
```

- In my machine, sizeof(node) is 16.
- We maintain a dummy node at the beginning of the list.
- An initialization function only creates the dummy node.
- An insert function makes a sorted insertion in the list. If the element to be inserted is already present, no new node is created.
- A delete function deletes a data from the list. If that data is not present in the list, the list is kept unchanged.

Delete without freeing: Example

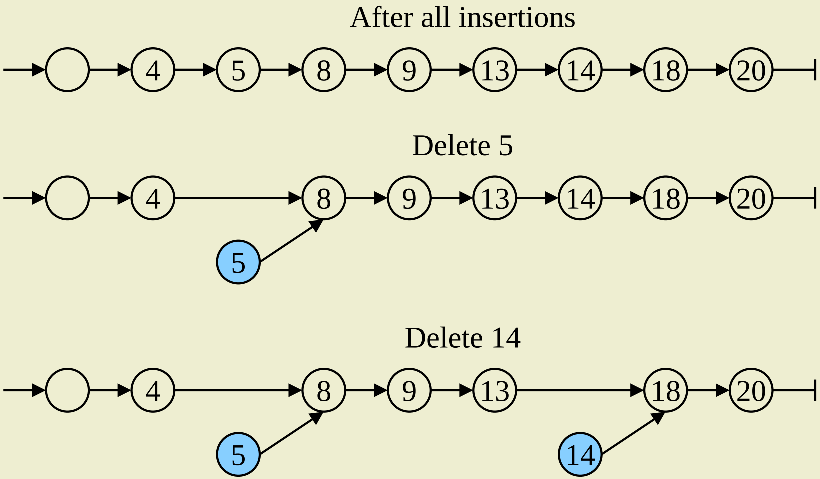
```
node *ldel ( node *L, int x )
{
   node *p;

   p = L;
   while (p -> next) {
      if (p -> next -> data == x) {
         p -> next = p -> next -> next;
         return L;
      }
      if (p -> next -> data > x) break;
      p = p -> next;
   }
   return L;
}
```

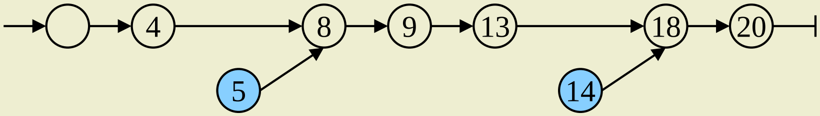
Delete without freeing: Example

```
sizeof(node) = 16
insert  5 : 5
insert 14 : 5 14
insert 18 : 5 14 18
insert  4 : 4 5 14 18
insert 13 : 4 5 13 14 18
insert 20 : 4 5 13 14 18 20
insert  8 : 4 5 8 13 14 18 20
insert  9 : 4 5 8 9 13 14 18 20
delete  5 : 4 8 9 13 14 18 20
delete 14 : 4 8 9 13 18 20
delete 18 : 4 8 9 13 20
delete 13 : 4 8 9 20
delete 20 : 4 8 9
==9837==
==9837== HEAP SUMMARY:
==9837==     in use at exit: 144 bytes in 9 blocks
==9837==   total heap usage: 10 allocs, 1 frees, 1,168 bytes allocated
==9837==
==9837== LEAK SUMMARY:
==9837==    definitely lost: 48 bytes in 3 blocks
==9837==    indirectly lost: 32 bytes in 2 blocks
==9837==      possibly lost: 0 bytes in 0 blocks
==9837==    still reachable: 64 bytes in 4 blocks
==9837==         suppressed: 0 bytes in 0 blocks
==9837== Rerun with --leak-check=full to see details of leaked memory
==9837==
==9837== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

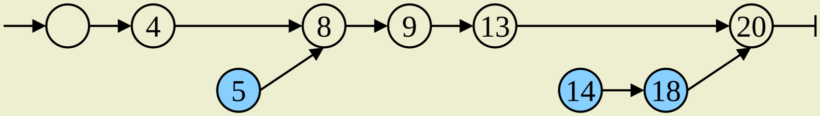
Memory leak example



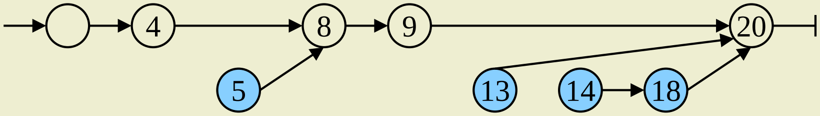
Memory leak example: Continued



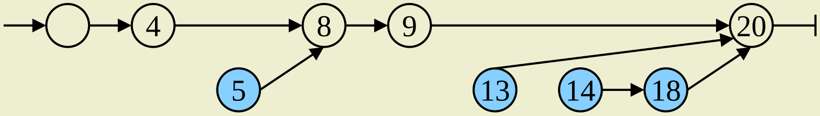
Delete 18



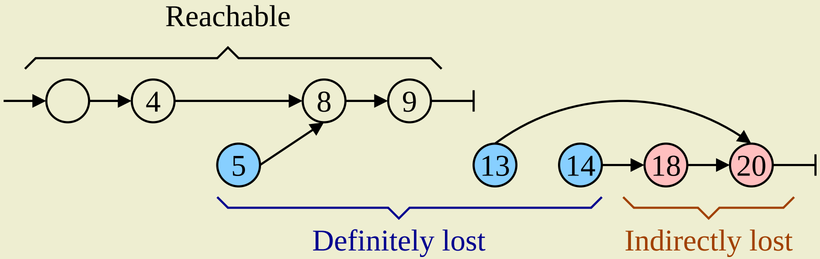
Delete 13



Memory leak example: Continued



Delete 20



Delete with freeing

```
node *ldel ( node *L, int x )
{
   node *p, *q;
   p = L;
   while (p -> next) {
      if (p -> next -> data == x) {
         q = p -> next; p -> next = q -> next; free(q);
         return L;
      }
      if (p -> next -> data > x) break;
      p = p -> next;
   }
   return L;
}
```

```
==10012==
==10012== HEAP SUMMARY:
==10012==     in use at exit: 64 bytes in 4 blocks
==10012==   total heap usage: 10 allocs, 6 frees, 1,168 bytes allocated
==10012==
==10012== LEAK SUMMARY:
==10012==    definitely lost: 0 bytes in 0 blocks
==10012==    indirectly lost: 0 bytes in 0 blocks
==10012==      possibly lost: 0 bytes in 0 blocks
==10012==    still reachable: 64 bytes in 4 blocks
==10012==         suppressed: 0 bytes in 0 blocks
==10012== Rerun with --leak-check=full to see details of leaked memory
==10012==
==10012== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Free the reachable nodes at the end

```
void ldestroy ( node *L )
{
   node *p;

   while (L) {
      p = L -> next;
      free(L);
      L = p;
   }
}
```

```
==10160==
==10160== HEAP SUMMARY:
==10160==     in use at exit: 0 bytes in 0 blocks
==10160==   total heap usage: 10 allocs, 10 frees, 1,168 bytes allocated
==10160==
==10160== All heap blocks were freed -- no leaks are possible
==10160==
==10160== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

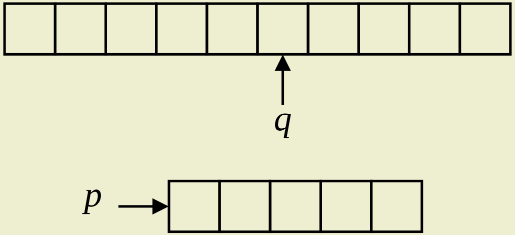
Demonstrating possibly lost memory

```
int main ()
{
   int *p, *q;

   p = (int *)malloc(10 * sizeof(int));
   q = p + 5;

   p = (int *)malloc(5 * sizeof(int));

   exit(0);
}
```



Possibly lost memory: valgrind output

```
==4155== Memcheck, a memory error detector
==4155== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==4155== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==4155== Command: ./a.out
==4155==
==4155==
==4155== HEAP SUMMARY:
==4155==     in use at exit: 60 bytes in 2 blocks
==4155==   total heap usage: 2 allocs, 0 frees, 60 bytes allocated
==4155==
==4155== LEAK SUMMARY:
==4155==    definitely lost: 0 bytes in 0 blocks
==4155==    indirectly lost: 0 bytes in 0 blocks
==4155==      possibly lost: 40 bytes in 1 blocks
==4155==    still reachable: 20 bytes in 1 blocks
==4155==         suppressed: 0 bytes in 0 blocks
==4155== Rerun with --leak-check=full to see details of leaked memory
==4155==
==4155== For lists of detected and suppressed errors, rerun with: -s
==4155== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Overflow in arrays

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

int main ()
{
   int n = 16, i, *A;

   A = (int *)malloc(n * sizeof(int));
   printf("A starts at %p, and ends at %p\n", A, A+n-1);
   for (i=1; i<=n; ++i) A[i] = i * i;
   for (i=1; i<=n; ++i) printf("%d ", A[i]);
   printf("\n");

   free(A);

   exit(0);
}
```

Overflow in arrays: valgrind output

```
==13180== Memcheck, a memory error detector
==13180== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==13180== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==13180== Command: ./a.out
==13180==
A starts at 0x4a5a040, and ends at 0x4a5a07c
==13180== Invalid write of size 4
==13180==    at 0x109240: main (in /home/abhij/IITKGP/course/lab/SPL/Spring22/prog/valgrind/a.out)
==13180== Address 0x4a5a080 is 0 bytes after a block of size 64 alloc'd
==13180==    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==13180==    by 0x1091EC: main (in /home/abhij/IITKGP/course/lab/SPL/Spring22/prog/valgrind/a.out)
==13180==
==13180== Invalid read of size 4
==13180==    at 0x10926B: main (in /home/abhij/IITKGP/course/lab/SPL/Spring22/prog/valgrind/a.out)
==13180== Address 0x4a5a080 is 0 bytes after a block of size 64 alloc'd
==13180==    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==13180==    by 0x1091EC: main (in /home/abhij/IITKGP/course/lab/SPL/Spring22/prog/valgrind/a.out)
==13180==
1 4 9 16 25 36 49 64 81 100 121 144 169 196 225 256
==13180==
==13180== HEAP SUMMARY:
==13180==     in use at exit: 0 bytes in 0 blocks
==13180==   total heap usage: 2 allocs, 2 frees, 1,088 bytes allocated
==13180==
==13180== All heap blocks were freed -- no leaks are possible
==13180==
==13180== For lists of detected and suppressed errors, rerun with: -s
==13180== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

Buffer overflow: Example

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main ( int argc, char *argv[] )
{
   char *wnote = malloc(32);

   if (argc == 1) exit(1);

   printf("The input has size %ld\n", strlen(argv[1]));
   printf("wnote starts at %p and ends at %p\n", wnote, wnote + 31);
   sprintf(wnote, "Welcome to %s", argv[1]);
   printf("%s\n", wnote);

   free(wnote);

   exit(0);
}
```

- Compile and run as follows. There may be no errors.

  ```
  ./a.out "Systems Programming Laboratory"
  ```

- Run as follows to see the problems.

  ```
  valgrind ./a.out "Systems Programming Laboratory"
  ```

Buffer overflow: valgrind output

```
==12432== Command: ./a.out Systems\ Programming\ Laboratory
==12432==
The input has size 30
wnote starts at 0x4a5a040 and ends at 0x4a5a05f
==12432== Invalid write of size 1
==12432== Address 0x4a5a060 is 0 bytes after a block of size 32 alloc'd
==12432== Invalid write of size 1
==12432== Address 0x4a5a069 is 9 bytes after a block of size 32 alloc'd
==12432== Invalid read of size 1
==12432== Address 0x4a5a060 is 0 bytes after a block of size 32 alloc'd
==12432== Invalid read of size 1
==12432== Address 0x4a5a068 is 8 bytes after a block of size 32 alloc'd
==12432== Invalid read of size 1
==12432== Address 0x4a5a066 is 6 bytes after a block of size 32 alloc'd
==12432==
Welcome to Systems Programming Laboratory
==12432==
==12432== HEAP SUMMARY:
==12432==     in use at exit: 0 bytes in 0 blocks
==12432==   total heap usage: 2 allocs, 2 frees, 1,056 bytes allocated
==12432==
==12432== All heap blocks were freed -- no leaks are possible
==12432==
==12432== For lists of detected and suppressed errors, rerun with: -s
==12432== ERROR SUMMARY: 38 errors from 6 contexts (suppressed: 0 from 0)
```