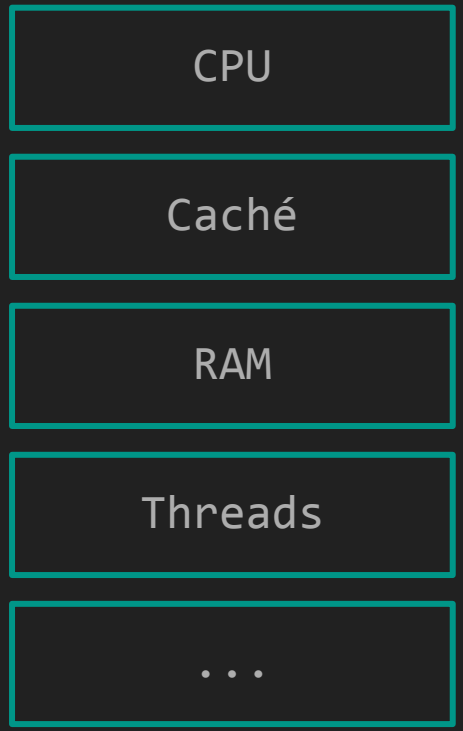
Intro a C

Valgrind

With 💜 by @vichoeq & @KnowYourselves

Valgrind

Valgrind es un set de herramientas para analizar y monitorear el uso de recursos de un programa.



Valgrind

Para esto ejecuta el programa en un espacio virtual.

Esto multiplica su tiempo de ejecución por ~40.

```
$ gcc main.c -g -o main
$ time ./main
real  0m2.630s
user  0m1.938s
sys   0m0.203s
```

```
$ gcc main.c -g -o main
$ time valgrind ./main
real  1m14.508s
user  1m10.641s
sys   0m2.500s
```

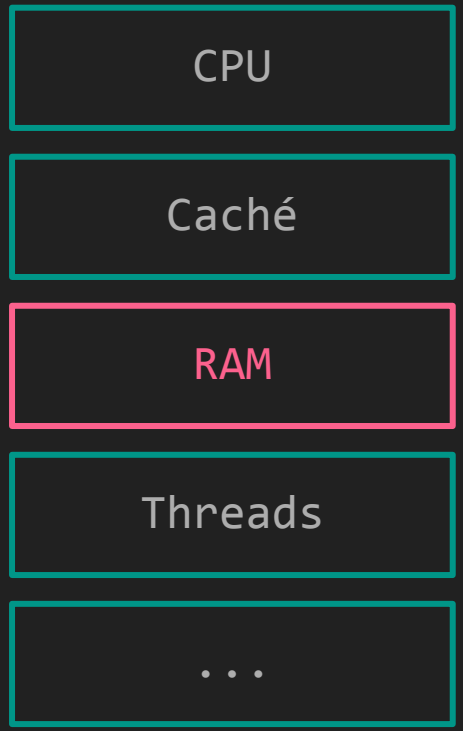
Memcheck

Memcheck

Memcheck

Memcheck es la herramienta default de Valgrind.

Sirve para detectar errores de memoria en el HEAP.



Memcheck - Output general

```
$ gcc main.c -g -o main
$ valgrind ./main
```

```c
int main()
{
  return 0;
}
```

Memcheck - Output general

```
$ gcc main.c -g -o main
$ valgrind ./main
==150== Memcheck, a memory error detector
==150== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==150== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==150== Command: ./main
==150==
==150== HEAP SUMMARY:
==150==       in use at exit: 0 bytes in 0 blocks
==150==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==150==
==150== All heap blocks were freed -- no leaks are possible
==150==
==150== For lists of detected and suppressed errors, rerun with: -s
==150== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

```c
int main()
{
  return 0;
}
```

Memcheck - Output general

```
$ gcc main.c -g -o main
$ valgrind ./main
==150== Memcheck, a memory error detector
==150== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==150== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==150== Command: ./main
==150==
==150== HEAP SUMMARY:
==150==       in use at exit: 0 bytes in 0 blocks
==150==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==150==
==150== All heap blocks were freed -- no leaks are possible
==150==
==150== For lists of detected and suppressed errors, rerun with: -s
==150== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

150 = Program ID

Memcheck - Output general

```
$ gcc main.c -g -o main
$ valgrind ./main
==150== Memcheck, a memory error detector
==150== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==150== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==150== Command: ./main
==150==
==150== HEAP SUMMARY:
==150==       in use at exit: 0 bytes in 0 blocks
==150==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==150==
==150== All heap blocks were freed -- no leaks are possible
==150==
==150== For lists of detected and suppressed errors, rerun with: -s
==150== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Información de versión

Memcheck - Output general

```
$ gcc main.c -g -o main
$ valgrind ./main
==150== Memcheck, a memory error detector
==150== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==150== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==150== Command: ./main
==150==
==150== HEAP SUMMARY:
==150==       in use at exit: 0 bytes in 0 blocks
==150==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==150==
==150== All heap blocks were freed -- no leaks are possible
==150==
==150== For lists of detected and suppressed errors, rerun with: -s
==150== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Comando ejecutado

Memcheck - Output general

```
$ gcc main.c -g -o main
$ valgrind ./main
==150== Memcheck, a memory error detector
==150== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==150== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==150== Command: ./main
==150==
==150== HEAP SUMMARY:
==150==       in use at exit: 0 bytes in 0 blocks
==150==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==150==
==150== All heap blocks were freed -- no leaks are possible
==150==
==150== For lists of detected and suppressed errors, rerun with: -s
==150== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Información de leaks

Memcheck - Output general

```
$ gcc main.c -g -o main
$ valgrind ./main
==150== Memcheck, a memory error detector
==150== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==150== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==150== Command: ./main
==150==
==150== HEAP SUMMARY:
==150==       in use at exit: 0 bytes in 0 blocks
==150==   total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==150==
==150== All heap blocks were freed -- no leaks are possible
==150==
==150== For lists of detected and suppressed errors, rerun with: -s
==150== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Otros errores de memoria

Memcheck
leaks

Memcheck - leaks

```
$ gcc leak_ind.c -g -o leak
$ valgrind ./leak
```

leak_ind.c

```c
#include <stdlib.h>

int main()
{
  int** B = malloc(10 * sizeof(int*));
  for(int i = 0; i < 10; i+=1)
  {
    B[i] = malloc(10 * sizeof(int));
  }
  return 0;
}
```

Memcheck - leaks

```
$ gcc leak_ind.c -g -o leak
$ valgrind ./leak
...
==72== HEAP SUMMARY:
==72== in use at exit: 480 bytes in 11 blocks
==72==   total heap usage: 11 allocs, 0 frees, 480 bytes allocated
==72==
==72== LEAK SUMMARY:
==72==    definitely lost: 80 bytes in 1 blocks
==72==    indirectly lost: 400 bytes in 10 blocks
==72==      possibly lost: 0 bytes in 0 blocks
==72==    still reachable: 0 bytes in 0 blocks
==72==         suppressed: 0 bytes in 0 blocks
==72== Rerun with --leak-check=full to see details of leaked memory
==72==
==72== For lists of detected and suppressed errors, rerun with: -s
==72== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

leak_ind.c

```c
#include <stdlib.h>

int main()
{
  int** B = malloc(10 * sizeof(int*));
  for(int i = 0; i < 10; i+=1)
  {
    B[i] = malloc(10 * sizeof(int));
  }
  return 0;
}
```

Memcheck - leaks

```
$ gcc leak_ind.c -g -o leak
$ valgrind ./leak
...
==72== HEAP SUMMARY:
==72== in use at exit: 480 bytes in 11 blocks
==72==   total heap usage: 11 allocs, 0 frees, 480 bytes allocated
==72==
==72== LEAK SUMMARY:
==72==    definitely lost: 80 bytes in 1 blocks
==72==    indirectly lost: 400 bytes in 10 blocks
==72==      possibly lost: 0 bytes in 0 blocks
==72==    still reachable: 0 bytes in 0 blocks
==72==         suppressed: 0 bytes in 0 blocks
==72== Rerun with --leak-check=full to see details of leaked memory
==72==
==72== For lists of detected and suppressed errors, rerun with: -s
==72== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

leak_ind.c

```c
#include <stdlib.h>

int main()
{
  int** B = malloc(10 * sizeof(int*));
  for(int i = 0; i < 10; i+=1)
  {
    B[i] = malloc(10 * sizeof(int));
  }
  return 0;
}
```

Memcheck - leaks

```
$ gcc leak_ind.c -g -o leak
$ valgrind ./leak
...
==72== HEAP SUMMARY:
==72== in use at exit: 480 bytes in 11 blocks
==72==   total heap usage: 11 allocs, 0 frees, 480 bytes allocated
==72==
==72== LEAK SUMMARY:
==72==    definitely lost: 80 bytes in 1 blocks
==72==    indirectly lost: 400 bytes in 10 blocks
==72==      possibly lost: 0 bytes in 0 blocks
==72==    still reachable: 0 bytes in 0 blocks
==72==         suppressed: 0 bytes in 0 blocks
==72== Rerun with --leak-check=full to see details of leaked memory
==72==
==72== For lists of detected and suppressed errors, rerun with: -s
==72== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

leak_ind.c

```c
#include <stdlib.h>

int main()
{
  int** B = malloc(10 * sizeof(int*));
  for(int i = 0; i < 10; i+=1)
  {
    B[i] = malloc(10 * sizeof(int));
  }
  return 0;
}
```

Memcheck - leaks

```
$ gcc leak_ind.c -g -o leak
$ valgrind --leak-check=full ./leak
...
==72== HEAP SUMMARY:
==72== in use at exit: 480 bytes in 11 blocks
==72==   total heap usage: 11 allocs, 0 frees, 480 bytes allocated
==72==
==72== 480 (80 direct, 400 indirect) bytes in 1 blocks are definitely lost
==72== in loss record 2 of 2
==72==    at 0x483B7F3: malloc (in ...)
==72==    by 0x10915F: main (leak_ind.c:5)
==72==
==72== LEAK SUMMARY:
==72==    definitely lost: 80 bytes in 1 blocks
==72==    indirectly lost: 400 bytes in 10 blocks
==72==      possibly lost: 0 bytes in 0 blocks
==72==    still reachable: 0 bytes in 0 blocks
...
```

leak_ind.c

```c
#include <stdlib.h>

int main()
{
  int** B = malloc(10 * sizeof(int*));
  for(int i = 0; i < 10; i+=1)
  {
    B[i] = malloc(10 * sizeof(int));
  }
  return 0;
}
```

Memcheck - leaks

```
$ gcc leak_ind.c -g -o leak
$ valgrind --leak-check=full ./leak
...
==72== HEAP SUMMARY:
==72== in use at exit: 480 bytes in 11 blocks
==72==   total heap usage: 11 allocs, 0 frees, 480 bytes allocated
==72==
==72== 480 (80 direct, 400 indirect) bytes in 1 blocks are definitely lost
==72== in loss record 2 of 2
==72==    at 0x483B7F3: malloc (in ...)
==72==    by 0x10915F: main (leak_ind.c:5)
==72==
==72== LEAK SUMMARY:
==72==    definitely lost: 80 bytes in 1 blocks
==72==    indirectly lost: 400 bytes in 10 blocks
==72==      possibly lost: 0 bytes in 0 blocks
==72==    still reachable: 0 bytes in 0 blocks
...
```

leak_ind.c

```c
#include <stdlib.h>

int main()
{
  int** B = malloc(10 * sizeof(int*));
  for(int i = 0; i < 10; i+=1)
  {
    B[i] = malloc(10 * sizeof(int));
  }
  return 0;
}
```

Memcheck - leaks

```
$ gcc leak_ind.c -g -o leak
$ valgrind --leak-check=full ./leak
```

?

leak_ind.c

```c
#include <stdlib.h>

int main()
{
  int** B = malloc(10 * sizeof(int*));
  for(int i = 0; i < 10; i+=1)
  {
    B[i] = malloc(10 * sizeof(int));
  }
  free(B);
  return 0;
}
```

Memcheck - leaks

```
$ gcc leak_ind.c -g -o leak
$ valgrind --leak-check=full ./leak
...
==72== HEAP SUMMARY:
==72== in use at exit: 480 bytes in 11 blocks
==72==   total heap usage: 11 allocs, 0 frees, 480 bytes allocated
==72==
==72== 400 bytes in 10 blocks are definitely lost in loss record 1 of 1
==72==    at 0x483B7F3: malloc (in ...)
==72==    by 0x1091AB: main (leak_ind.c:8)
==72==
==72== LEAK SUMMARY:
==72==    definitely lost: 400 bytes in 10 blocks
==72==    indirectly lost: 0 bytes in 0 blocks
==72==      possibly lost: 0 bytes in 0 blocks
==72==    still reachable: 0 bytes in 0 blocks
...
```

leak_ind.c

```c
#include <stdlib.h>

int main()
{
  int** B = malloc(10 * sizeof(int*));
  for(int i = 0; i < 10; i+=1)
  {
    B[i] = malloc(10 * sizeof(int));
  }
  free(B);
  return 0;
}
```

Memcheck - leaks

```
$ gcc leak_ind.c -g -o leak
$ valgrind --leak-check=full ./leak
...
==72== HEAP SUMMARY:
==72== in use at exit: 480 bytes in 11 blocks
==72==   total heap usage: 11 allocs, 0 frees, 480 bytes allocated
==72==
==72== 400 bytes in 10 blocks are definitely lost in loss record 1 of 1
==72==    at 0x483B7F3: malloc (in ...)
==72==    by 0x1091AB: main (leak_ind.c:8)
==72==
==72== LEAK SUMMARY:
==72==    definitely lost: 400 bytes in 10 blocks
==72==    indirectly lost: 0 bytes in 0 blocks
==72==      possibly lost: 0 bytes in 0 blocks
==72==    still reachable: 0 bytes in 0 blocks
...
```

leak_ind.c

```c
#include <stdlib.h>

int main()
{
  int** B = malloc(10 * sizeof(int*));
  for(int i = 0; i < 10; i+=1)
  {
    B[i] = malloc(10 * sizeof(int));
  }
  free(B);
  return 0;
}
```

Memcheck - leaks

```
$ gcc leak_ind.c -g -o leak
$ valgrind --leak-check=full --show-leak-kinds=all ./leak
```

leak_ind.c

```c
#include <stdlib.h>

int main()
{
  int** B = malloc(10 * sizeof(int*));
  for(int i = 0; i < 10; i+=1)
  {
    B[i] = malloc(10 * sizeof(int));
  }
  return 0;
}
```

Memcheck errores

error - use of uninitialized value

```
gcc bad_val.c -g -o error
valgrind ./error
```

bad_val.c

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

int main()
{
  int* E = malloc(sizeof(int));
  if (*E > 10)
  {
    printf("Hi there!\n");
  }
  free(E);
  return 0;
}
```

error - use of uninitialized value

```
gcc bad_val.c -g -o error
valgrind ./error
...
==49== Conditional jump or move depends on uninitialised value(s)
==49==    at 0x1091AC: main (bad_val.c:7)
==49==
==49==
==49== HEAP SUMMARY:
==49== in use at exit: 0 bytes in 0 blocks
==49==   total heap usage: 1 allocs, 1 frees, 4 bytes allocated
==49==
==49== All heap blocks were freed -- no leaks are possible
==49==
==49== Use --track-origins=yes to see where uninitialised values come from
==49== For lists of detected and suppressed errors, rerun with: -s
==49== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

bad_val.c

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

int main()
{
  int* E = malloc(sizeof(int));
  if (*E > 10)
  {
    printf("Hi there!\n");
  }
  free(E);
  return 0;
}
```

error - use of uninitialized value

```
gcc bad_val.c -g -o error
valgrind ./error
...
==49== Conditional jump or move depends on uninitialised value(s)
==49==    at 0x1091AC: main (bad_val.c:7)
==49==
==49==
==49== HEAP SUMMARY:
==49== in use at exit: 0 bytes in 0 blocks
==49==   total heap usage: 1 allocs, 1 frees, 4 bytes allocated
==49==
==49== All heap blocks were freed -- no leaks are possible
==49==
==49== Use --track-origins=yes to see where uninitialised values come from
==49== For lists of detected and suppressed errors, rerun with: -s
==49== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

bad_val.c

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

int main()
{
  int* E = malloc(sizeof(int));
  if (*E > 10)
  {
    printf("Hi there!\n");
  }
  free(E);
  return 0;
}
```

error - use of uninitialized value

```
gcc bad_val.c -g -o error
valgrind --track-origins=yes ./error
...
==49== Conditional jump or move depends on uninitialised value(s)
==49==    at 0x1091AC: main (bad_val.c:7)
==49== Uninitialised value was created by a heap allocation
==49==    at 0x483B7F3: malloc (in ...)
==49==    by 0x10919E: main (bad_val.c:6)
==49==
==49==
==49== HEAP SUMMARY:
==49== in use at exit: 0 bytes in 0 blocks
==49==   total heap usage: 1 allocs, 1 frees, 4 bytes allocated
==49==
==49== All heap blocks were freed -- no leaks are possible
==49==
==49== For lists of detected and suppressed errors, rerun with: -s
==49== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

bad_val.c

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

int main()
{
  int* E = malloc(sizeof(int));
  if (*E > 10)
  {
    printf("Hi there!\n");
  }
  free(E);
  return 0;
}
```

error - invalid write

```
gcc bad_write.c -g -o error
valgrind ./error
```

bad_write.c

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

int main()
{
  int* F = calloc(10, sizeof(int));
  for(int i = 0; i <= 10; i+=1)
  {
    F[i] = i * i;
  }
  free(F);
  return 0;
}
```

error - invalid write

```
gcc bad_write.c -g -o error
valgrind ./error
...
==182== Invalid write of size 4
==182==    at 0x1091AB: main (bad_write.c:9)
==182== Address 0x4a47068 is 0 bytes after a block of size 40 alloc'd
==182==    at 0x483DD99: calloc (in ...)
==182==    by 0x109183: main (bad_write.c:6)
==182==
==182==
==182== HEAP SUMMARY:
==182==       in use at exit: 0 bytes in 0 blocks
==182==   total heap usage: 1 allocs, 1 frees, 40 bytes allocated
==182==
==182== All heap blocks were freed -- no leaks are possible
==182==
==182== For lists of detected and suppressed errors, rerun with: -s
==182== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

bad_write.c

```
1  #include <stdlib.h>
2  #include <stdio.h>
3
4  int main()
5  {
6    int* F = calloc(10, sizeof(int));
7    for(int i = 0; i <= 10; i+=1)
8    {
9      F[i] = i * i;
10   }
11   free(F);
12   return 0;
13 }
```

error - invalid write

```
gcc bad_write.c -g -o error
valgrind ./error
...
==182== Invalid write of size 4
==182==    at 0x1091AB: main (bad_write.c:9)
==182== Address 0x4a47068 is 0 bytes after a block of size 40 alloc'd
==182==    at 0x483DD99: calloc (in ...)
==182==    by 0x109183: main (bad_write.c:6)
==182==
==182==
==182== HEAP SUMMARY:
==182==       in use at exit: 0 bytes in 0 blocks
==182==   total heap usage: 1 allocs, 1 frees, 40 bytes allocated
==182==
==182== All heap blocks were freed -- no leaks are possible
==182==
==182== For lists of detected and suppressed errors, rerun with: -s
==182== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

bad_write.c

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

int main()
{
  int* F = calloc(10, sizeof(int));
  for(int i = 0; i <= 10; i+=1)
  {
    F[i] = i * i;
  }
  free(F);
  return 0;
}
```

error - invalid read

```
gcc bad_read.c -g -o error
valgrind ./error
```

bad_read.c

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

int main()
{
  int* G = malloc(10 * sizeof(int));
  for(int i = 0; i < 10; i+=1)
  {
    G[i] = i * i + G[i - 1];
  }
  free(G);
  return 0;
}
```

error - invalid read

```
gcc bad_read.c -g -o error
valgrind ./error
...
==206== Invalid read of size 4
==206==    at 0x1091A8: main (bad_read.c:9)
==206==  Address 0x4a4703c is 4 bytes before a block of size 40 alloc'd
==206==    at 0x483B7F3: malloc (in ...)
==206==    by 0x10917E: main (bad_read.c:6)
==206==
==206==
==206== HEAP SUMMARY:
==206==     in use at exit: 0 bytes in 0 blocks
==206==   total heap usage: 1 allocs, 1 frees, 40 bytes allocated
==206==
==206== All heap blocks were freed -- no leaks are possible
==206==
==206== For lists of detected and suppressed errors, rerun with: -s
==206== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

bad_read.c

```
1  #include <stdlib.h>
2  #include <stdio.h>
3
4  int main()
5  {
6    int* G = malloc(10 * sizeof(int));
7    for(int i = 0; i < 10; i+=1)
8    {
9      G[i] = i * i + G[i - 1];
10   }
11   free(G);
12   return 0;
13 }
```

error - invalid read

```
gcc bad_read.c -g -o error
valgrind ./error
...
==206== Invalid read of size 4
==206==    at 0x1091A8: main (bad_read.c:9)
==206==  Address 0x4a4703c is 4 bytes before a block of size 40 alloc'd
==206==    at 0x483B7F3: malloc (in ...)
==206==    by 0x10917E: main (bad_read.c:6)
==206==
==206==
==206== HEAP SUMMARY:
==206==     in use at exit: 0 bytes in 0 blocks
==206==   total heap usage: 1 allocs, 1 frees, 40 bytes allocated
==206==
==206== All heap blocks were freed -- no leaks are possible
==206==
==206== For lists of detected and suppressed errors, rerun with: -s
==206== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

bad_read.c

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

int main()
{
  int* G = malloc(10 * sizeof(int));
  for(int i = 0; i < 10; i+=1)
  {
    G[i] = i * i + G[i - 1];
  }
  free(G);
  return 0;
}
```

Memcheck
STACK trace

STACK trace

```
gcc stack_trace.c -g -o stack_trace
valgrind --leak-check=full ./stack_trace
```

stack_trace.c

```c
#include <stdlib.h>
int* p()
{
  return malloc(sizeof(int));
}
int* q()
{
  return p();
}
int main()
{
  int* H = q();
  return 0;
}
```

STACK trace

```
gcc stack_trace.c -g -o stack_trace
valgrind --leak-check=full ./stack_trace
...
==216== HEAP SUMMARY:
==216==       in use at exit: 4 bytes in 1 blocks
==216==   total heap usage: 1 allocs, 0 frees, 4 bytes allocated
==216==
==216== 4 bytes in 1 blocks are definitely lost in loss record 1 of 1
==216==    at 0x483B7F3: malloc (in ...)
==216==    by 0x10915A: p (stack_trace.c:4)
==216==    by 0x10916E: q (stack_trace.c:8)
==216==    by 0x109186: main (stack_trace.c:12)
==216==
==216== LEAK SUMMARY:
==216==    definitely lost: 4 bytes in 1 blocks
==216==    indirectly lost: 0 bytes in 0 blocks
...
```

stack_trace.c

```c
#include <stdlib.h>
int* p()
{
  return malloc(sizeof(int));
}
int* q()
{
  return p();
}
int main()
{
  int* H = q();
  return 0;
}
```

¡Muchas Gracias!

With 💜 by @vichoeq & @KnowYourselves