```
#valgrind --log-file=x.txt ./sample

ile rapor x.txt dosyasına loglanır


#valgrind --leak-check=full

ile memory leak için daha detaylı bilgi alınabilir. Default durum summary şeklindedir

#valgrind --leak-check=full --xml=yes --xml-file=x.xml

seçeneği ile xml dosya elde edilebilir

#valgrind --tool=callgrind ile kullanılacak bileşen belirlenebilir

#valgrind --tool=memcheck --vgdb=yes --vgdb-error=0 ./sample

#Başka bir terminalde gdb ile çalıştırılır. Bunun için target remote | vgdb yapılır

#Bu noktadan sonra remote olarak erişilen vgdb içerisinde program çalıştırılır. Fakat programın ilerletilmesi gdb den yapılır. Dolayısıyla
#interaktif bir program vgdb tarafında girişler ve çıkışlar yapılabilir


#valgrind (memcheck) aşağıdaki gibi bir koddaki hatayı yakalayamaz. Çünkü alan dinamik tahsis edilmemiştir.

#include <iostream>
#include <cstdlib>

using namespace std;

int main()
{
  int k[10];

  k[13] = 20;

  return 0;
}

# Aşağıdaki örnekte tahsis edilen alan free edilmediği için valgrind yakalar

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

int main(int argc, char **argv)
{
    int *p;

    p = (int *)malloc(10 * sizeof(int));

    if (!p) {
        fprintf(stderr, "Can not allocate memory\n");
```

```
53          exit(EXIT_FAILURE);
54      }
55
56
57      return 0;
58  }
59
60  # Aşağıdaki örnekte free edilmiş bir alan tekrar free edilmiştir. Bu C/C++ da
      undefined behaviour dır. valgrind yakalar
61
62  #include <stdio.h>
63  #include <stdlib.h>
64
65  int main(int argc, char **argv)
66  {
67      int *p;
68
69      p = (int *)malloc(10 * sizeof(int));
70
71      if (!p) {
72          fprintf(stderr, "Can not allocate memory\n");
73          exit(EXIT_FAILURE);
74      }
75
76      free(p);
77      free(p);
78
79
80      return 0;
81  }
82
83
84  #include <iostream>
85
86  using namespace std;
87
88  int main()
89  {
90      int *p = new int;
91
92      delete p;
93      delete p;
94
95
96      return 0;
97  }
98
99  # Aşağıdaki örnekte heap de tahsis edilmemiş bir alan free edilmeye
      çalışılmıştır. valgrind yakalar
100
101 #include <stdio.h>
102 #include <stdlib.h>
103
104 int main(int argc, char **argv)
105 {
106     int a[10];
```

```
107     int *p = a;
108
109     free(p);
110
111
112     return 0;
113 }
114
115 #include <iostream>
116
117 using namespace std;
118
119 int main()
120 {
121     int a[10];
122
123     int *p = a;
124
125     delete p;
126
127
128     return 0;
129 }
130
131
132
133 Aşağıdaki kodda valgrind sadece ilk değer vermeyi değil. Fonksiyona geçirilen ↵
      argümana da ilk değer verilmediğinin raporunu verir
134
135 #include <iostream>
136 #include <cstdlib>
137 #include <cstdio>
138 #include <unistd.h>
139
140 using namespace std;
141
142 int main()
143 {
144   char *pc = new char[10];
145   int *p = new int;
146
147   write(1, pc, 10);
148   cout << *p << endl;
149   exit(*p);
150
151   return 0;
152 }
153
154 #Aşağıdaki kodda dinamik tahsis edilmemiş alan delete edilmeye çalışılıyor.  ↵
      valgrind yakalar
155
156 #include <iostream>
157 #include <cstdlib>
158 #include <cstdio>
159 #include <unistd.h>
160
```

```
161 using namespace std;
162
163 int main()
164 {
165   int a = 10;
166
167   int *p = &a;
168
169   cout << *p << endl;
170
171   delete p;
172
173   return 0;
174 }
175
176 # Aşağıdaki kodda iki defa delete yapılmıştır. valgrind yakalar
177 #include <iostream>
178 #include <cstdlib>
179 #include <cstdio>
180 #include <unistd.h>
181
182 using namespace std;
183
184 int main()
185 {
186   int *p = new int;
187
188   *p = 10;
189
190   cout << *p << endl;
191
192   delete p;
193   delete p;
194
195   return 0;
196 }
197
198 #Aşağıdaki kodda n elemanlı blok tek elemanlı gibi siliniyor. Silmede problem ↵
      olmaz.
199 valgrind yakalar
200
201 #include <iostream>
202 #include <cstdlib>
203 #include <cstdio>
204 #include <unistd.h>
205
206 using namespace std;
207
208 int main()
209 {
210   int *p = new int[10];
211
212   *p = 10;
213
214   cout << *p << endl;
215
```

```
216   delete p;
217
218   return 0;
219 }
220
221 #Aşağıdaki kodda delete yine tanımsız davranıştır. Bir sorun oluşturmasa da
      yapılmamalıdır. valgrind yakalar
222
223 #include <iostream>
224 #include <cstdlib>
225 #include <cstdio>
226 #include <unistd.h>
227
228 using namespace std;
229
230 class Sample {
231   int x;
232   char y;
233   short z;
234   double t;
235 };
236
237 int main()
238 {
239   Sample *s = new Sample[40000];
240   cout << sizeof(Sample) << endl;
241
242   delete s;
243
244   return 0;
245 }
246
247 #Aşağıdaki kodda kopyalanan bloklar çakışık olduğundan memcpy nin  davranışı
      tanımsızdır. valgrind yakalar. memmove için birşey olmaz
248
249 #include <iostream>
250 #include <cstring>
251
252 using namespace std;
253
254 int main()
255 {
256   int *p = new int[10];
257
258   p[0] = 10;
259   p[1] = 20;
260
261   memmove(p + 1, p, 2 * sizeof(int));
262   memcpy(p + 1, p, 2 * sizeof(int));
263
264   delete [] p;
265
266   return 0;
267 }
268
269
```

270
271
272
273