CSE 333 25sp Midterm Exam 5/9/25

Name _____ UW Netid: _____@uw.edu
(please *print* ***legibly***)**

There are 7 questions worth a total of 100 points. Please budget your time so you get to all of the questions. Keep your answers brief and to the point.

The exam is closed book, closed notes, closed electronics, closed telepathy, open mind. However, you may have a single 5x8 notecard with any hand-written notes you wish on both sides.

There is a blank page at the end with extra space for your answers if you need more room. It is after all the questions but before the detachable pages with reference information.

After the extra blank page for answers, there is a sheet of paper containing assorted reference information (most of which you probably won't need). You should remove this reference sheet from the exam and use it during the exam. It will not be scanned for grading, so do not write answers on it.

Do not remove any pages from the middle of the exam.

If you do not remember the exact syntax for something, make the best attempt you can. We will make allowances when grading.

Don't be alarmed if there seems to be more space than is needed for some answers – we tried to include enough blank space.

Relax, you are here to learn.

Please wait to turn the page until everyone is told to begin.

Score _____ / 100

1. _____ / 16
2. _____ / 20
3. _____ / 18
4. _____ / 12
5. _____ / 20
6. _____ / 12
7. _____ / 2

Question 1. (16 points) Making things. Once again, we're working on some software for our band. This time we're using C++ and we've got two main programs, `Song` and `BandInfo`, which use libraries named `Graphic` and `Links`. Here's how the files are related.

Links.h	Graphic.h	Graphic.cc

`...`	`Class Graphic {...}`	`#include "Links.h"`
		`#include "Graphic.h"`
		`// Graphics class`
		`// implementations`

Song.cc	BandInfo.cc

`#include "Links.h"`
 	
`int main(...) {...}`
	`#include "Graphic.h"`
	
	`int main(...) {...}`

(a) (7 points) We want to construct a `Makefile` that will build either or both of these programs, and incrementally re-compile and rebuild programs after files are changed. The `Makefile` should have a default target named `all` that builds (or rebuilds) both programs, as well as separate targets to build the `Song` and `BandInfo` programs.

Draw the dependency diagram showing the dependencies between all files used or created during the build process. You should draw an arrow pointing from each file or target that is built by the `Makefile` to the files or targets that it depends on.

(continued on next page)

Question 1. (cont.) (b) (7 points) One of the summer interns has written the following `Makefile` which, unfortunately, doesn't work properly. The `Makefile` should build both programs by default if `make` is executed without any arguments, and if we run `make clean`, it should remove all compiled files and any editor backup files with filenames ending with the ~ character. If some files are changed, only necessary files should be re-compiled or re-linked.

Fix the `Makefile` so it works as specified. You should write your changes on the code below. If you need to add or move lines, write them in the correct place, or show clearly with arrows where they should be located.

Hint: recall that if we compile `Foo.cc` with the `-c` option and do not specify an output file name (no `-o` option), the output file created will be named `Foo.o`.

```
Song: Song.o BandInfo.o

      g++ -Wall -g -std=c++17 -o Song Song.o

BandInfo: BandInfo.o Graphic.o

      g++ -Wall -g -std=c++17 -o BandInfo BandInfo.o Graphic.o

Song.o: Song.cc Links.h

      g++ -Wall -g -std=c++17 -c Song.cc

BandInfo.o: BandInfo.cc Graphic.h Links.h

      g++ -Wall -g -std=c++17 -c BandInfo.cc

all: Song BandInfo

clean:
```

(c) (2 points) Suppose we fix all the bugs in the `Makefile` and run `make` to build everything. Then, suppose we edit and change `Graphic.cc` and run `make` again. List the `Makefile` targets that are rebuilt after this change in the order in which they are rebuilt. If there is more than one possible ordering, give one of the possible orderings.

Question 2. (20 points) Valgrind and memory. This question concerns the following C program `membugs.c` that manipulates `SimpleString` structs, which are a representation for string data used in a section exercise.

Warning!! Watch your time and do not get bogged down on this question! Only a small number of fixes are needed once you've found the problem(s).

Note: error checks for `NULL` pointers following `malloc` functions are omitted to save space. We will assume `malloc` works without failing for this problem.

```
 1  #include <stdlib.h>
 2  #include <stdio.h>
 3  #include <string.h>
 4  #define ARRAY_SIZE 3
 5
 6  typedef struct simplestring_st {
 7    char* word;
 8    int num_letters;
 9  } SimpleString;
10
11  // Copy length number of characters from source into dest
12  static void CopyMe(char* source, int length, char* dest);
13
14  int main(int argc, char **argv) {
15    char word[] = "apples";
16    // create array of ARRAY_SIZE pointers to new
17    //    SimpleString structs on the heap
18    SimpleString** arr =
19       (SimpleString**) malloc(ARRAY_SIZE*sizeof(SimpleString*));
20    for (int i = 0; i < ARRAY_SIZE; i++) {
21      SimpleString simple_str;
22      arr[i] = &simple_str;
23      int length = strlen(word);
24      arr[i]->word = (char*) malloc(length);
25      printf("%p\n", (void*) arr[i]->word);
26      CopyMe(word, length, arr[i]->word);
27    }
28
```

(continued on next page)

Question 2 (cont.) Remainder of program code:

```
29    // print only the first array element
30    printf("%s has %i letters\n", arr[0]->word,
31                                  arr[0]->num_letters);
32    // clean up
33    for (int i = 0; i < ARRAY_SIZE; i++) {
34      printf("%p\n", (void*) arr[i]->word);
35      free(arr[i]->word);
36    }
37    free(arr);
38    return EXIT_SUCCESS;
39  }
40
41  static void CopyMe(char* source, int length, char* dest) {
42    for (int i = 0; i < length; i++) {
43      dest[i] = source[i];
44    }
45  }
```

Question continues on next page. Remainder of this page left blank to be used as needed while working the problem.

(continued on next page)

Question 2. (cont.) The program compiles without errors. But when we run it, it prints a strange value for the length of "apples", then crashes at the end with the message

```
free(): double free detected in tcache 2
```

We re-ran the program using valgrind to see if we could get more information, and discovered that there were more problems with the code.

Your job for this question is to examine the code and the valgrind report and then show where the bugs are in the code and how to fix them. You should show corrections by crossing out, changing, or adding code in the listing on the previous two pages. There is extra blank space at the bottom of the previous page for you to use if you need additional space, or if you need room to draw diagrams or do other work to figure out the corrections.

A second copy of this valgrind output is included on the very last sheet of paper at the end of the exam. You can remove that page and use it for reference while solving this problem to minimize page flipping.

```
==2913707== Memcheck, a memory error detector
==2913707== Copyright (C) 2002-2024, and GNU GPL'd, by Julian Seward et al.
==2913707== Using Valgrind-3.23.0 and LibVEX; rerun with -h for copyright info
==2913707== Command: ./membugs
==2913707==
0x4a770a0
0x4a77530
0x4a77580
==2913707== Invalid read of size 1
==2913707==    at 0x484DE84: strlen (vg_replace_strmem.c:505)
==2913707==    by 0x48CD0F7: __vfprintf_internal (in /usr/lib64/libc.so.6)
==2913707==    by 0x48C14FE: printf (in /usr/lib64/libc.so.6)
==2913707==    by 0x40126D: main (membugs.c:30)
==2913707==  Address 0x4a77586 is 0 bytes after a block of size 6 alloc'd
==2913707==    at 0x484482F: malloc (vg_replace_malloc.c:446)
==2913707==    by 0x4011DF: main (membugs.c:24)
==2913707==
==2913707== Conditional jump or move depends on uninitialised value(s)
==2913707==    at 0x48CC3BB: __vfprintf_internal (in /usr/lib64/libc.so.6)
==2913707==    by 0x48C14FE: printf (in /usr/lib64/libc.so.6)
==2913707==    by 0x40126D: main (membugs.c:30)
==2913707==
==2913707== Use of uninitialised value of size 8
==2913707==    at 0x48C091B: _itoa_word (in /usr/lib64/libc.so.6)
==2913707==    by 0x48CBFAB: __vfprintf_internal (in /usr/lib64/libc.so.6)
==2913707==    by 0x48C14FE: printf (in /usr/lib64/libc.so.6)
==2913707==    by 0x40126D: main (membugs.c:30)
==2913707==
==2913707== Conditional jump or move depends on uninitialised value(s)
==2913707==    at 0x48C092C: _itoa_word (in /usr/lib64/libc.so.6)
==2913707==    by 0x48CBFAB: __vfprintf_internal (in /usr/lib64/libc.so.6)
==2913707==    by 0x48C14FE: printf (in /usr/lib64/libc.so.6)
==2913707==    by 0x40126D: main (membugs.c:30)
```

(continued on next page)

Question 2. (cont.) Remainder of valgrind output:

```
==2913707==
==2913707== Conditional jump or move depends on uninitialised value(s)
==2913707==    at 0x48CC8A3: __vfprintf_internal (in /usr/lib64/libc.so.6)
==2913707==    by 0x48C14FE: printf (in /usr/lib64/libc.so.6)
==2913707==    by 0x40126D: main (membugs.c:30)
==2913707==
==2913707== Conditional jump or move depends on uninitialised value(s)
==2913707==    at 0x48CC0C7: __vfprintf_internal (in /usr/lib64/libc.so.6)
==2913707==    by 0x48C14FE: printf (in /usr/lib64/libc.so.6)
==2913707==    by 0x40126D: main (membugs.c:30)
==2913707==
apples has 32768 letters
0x4a77580
0x4a77580
==2913707== Invalid free() / delete / delete[] / realloc()
==2913707==    at 0x4847B4C: free (vg_replace_malloc.c:989)
==2913707==    by 0x4012C4: main (membugs.c:35)
==2913707==  Address 0x4a77580 is 0 bytes inside a block of size 6 free'd
==2913707==    at 0x4847B4C: free (vg_replace_malloc.c:989)
==2913707==    by 0x4012C4: main (membugs.c:35)
==2913707==  Block was alloc'd at
==2913707==    at 0x484482F: malloc (vg_replace_malloc.c:446)
==2913707==    by 0x4011DF: main (membugs.c:24)
==2913707==
0x4a77580
==2913707==
==2913707== HEAP SUMMARY:
==2913707==     in use at exit: 12 bytes in 2 blocks
==2913707==   total heap usage: 5 allocs, 5 frees, 1,066 bytes allocated
==2913707==
==2913707== 12 bytes in 2 blocks are definitely lost in loss record 1 of 1
==2913707==    at 0x484482F: malloc (vg_replace_malloc.c:446)
==2913707==    by 0x4011DF: main (membugs.c:24)
==2913707==
==2913707== LEAK SUMMARY:
==2913707==    definitely lost: 12 bytes in 2 blocks
==2913707==    indirectly lost: 0 bytes in 0 blocks
==2913707==      possibly lost: 0 bytes in 0 blocks
==2913707==    still reachable: 0 bytes in 0 blocks
==2913707==         suppressed: 0 bytes in 0 blocks
==2913707==
==2913707== Use --track-origins=yes to see where uninitialised values come from
==2913707== For lists of detected and suppressed errors, rerun with: -s
==2913707== ERROR SUMMARY: 17 errors from 8 contexts (suppressed: 0 from 0)
```

Question 3. (18 points) POSIX I/O. You have been assigned to a critical mission. The world's stealthiest operatives - the Kitten Spy Network - have slipped a top-secret message into your machine under the filename `input.txt`. Your job is to safely copy this file to a new file `output.txt` and preserve the data so that it can be picked up by another undercover agent.

All fancy tech has been compromised. You must use only POSIX I/O functions**.**

Here is a summary of some key POSIX I/O functions for your reference.

int open(const char **name*, int *mode*);
    mode is one of O_RDONLY, O_WRONLY, O_RDWR
int creat(const char **name*, int *mode*);
    create a new file
int close(int *fd*);
ssize_t read(int *fd*, void **buffer*, size_t *count*);
    returns # bytes read or 0 (eof) or -1 (error)
ssize_t write(int *fd*, void **buffer*, size_t *count*);
    returns # bytes written or -1 (error)

You can also use perror (for error reporting) and exit (to quit if something fails).

Below is the code you are to complete. Some blanks are unnecessary, so cross them out if they don't apply to your solution. You should assume that all necessary header files have been `#included` and you do not need to add any `#includes`. You also only need to fill in blanks that are already included in the code. You should assume that there will be no recoverable errors and do not need to add any error handling code beyond what is already included in the question.

```
#define BUF_SIZE 100
int main() {
    int input_fd = open(_____________, _____________);
    if (_____________ < 0) {
        perror("Failed when opening input file");
        close(_____________);
        return EXIT_FAILURE;
    }
    int output_fd = open( _____________, _____________);
    if (________ < 0) {
        perror("Failed when opening output file");
        close(_____________);
        close(_____________);
        return EXIT_FAILURE;
    }
```

(continued on next page)

Question 3 (cont.)

```
    char buffer[BUF_SIZE];
    ssize_t bytes_read;
    while ((bytes_read = read(input_fd, buffer, BUF_SIZE)) > 0) {
        ssize_t bytes_written = 0;
        while (bytes_written < bytes_read) {
            ssize_t result = write(output_fd,
                                   _________________________________,
                                   _________________________________);
            if (result < 0) {
                perror("Failed to write to output file");
                close(input_fd);
                close(output_fd);
                return EXIT_FAILURE;
            }
            bytes_written += _______________;
    }

    if (_______________ < 0) {
        perror("Failed while reading from input file");
        close(input_fd);
        close(output_fd);
        return EXIT_FAILURE;
    }

    close(input_fd);
    close(output_fd);
    return EXIT_SUCCESS;
}
```

Question 4. (12 points) Preprocessor. Suppose we have the following four C source files:

foo.h

```
#ifndef FOO_H_
#define FOO_H_

#include "bar.h"
#define IT 42
#define MORE IT + 1
struct empty { };

#endif  // FOO_H_
```

main.c

```
#include "bar.h"
#include "foo.h"

int main() {
  int k = MORE;
  bigint n = bar(k+IT);
  return 0;
}
```

bar.h

```
#ifndef BAR_H_
#define BAR_H_

typedef long int bigint;
bigint bar(int n);

#endif  // BAR_H_
```

bar.c

```
#include "bar.h"

bigint bar(int n) {
  return MORE*IT*n;
}
```

Show the output produced by the C preprocessor when it processes file `main.c` (i.e., if we were compiling this file, what output would the preprocessor send to the C compiler that actually translates the contents of `main.c` to machine code?) Hint: remember that the preprocessor only does string substitution and does not analyze the C code it produces for correctness.

Question 5. (20 points) Here is another one of those slightly maddening C++ programs with a fairly simple class, which represents chocolate chip cookies this time, and a small program that uses it. The code compiles and executes with no errors. In the box on the right, write the output produced when it runs.

You should assume that all copy constructors, constructors, and destructors are called as specified by the C++ language and not eliminated by possible compiler optimizations

```
#include <iostream>
using namespace std;
class Cookie {
 public:
  Cookie() : chips_(0) { cout << "Bake plain cookie" << endl; }
  Cookie(int n) : chips_(n) {
    cout << "Bake " << chips_ << " chip cookie" << endl;
  }
  Cookie(const Cookie& other) : chips_(other.chips_) {
    cout << "Clone " << chips_ << " chips" << endl;
  }
  Cookie& operator=(const Cookie& other) {
    cout << "Replace " << chips_
         << " chips with " << other.chips_ << " chips" << endl;
    if (this != &other) chips_ = other.chips_;
    return *this;
  }
  ~Cookie() { cout << "Eat " << chips_ << " chips" << endl; }
  int chips() const { return chips_; }
private:
  int chips_;
};

Cookie taste(Cookie c) {
  cout << "Taste cookie..." << endl;
  return c;
}

int main() {
  Cookie a(3);
  Cookie b = 2;
  cout << "--1--" << endl;
  Cookie c;
  c = a
  cout << "--2--" << endl;
  Cookie d = taste(b);
  cout << "-- 3--" << endl;
  return EXIT_SUCCESS;
}
```

Write the program output here

Question 6. (12 points) One of your colleagues has some questions about C++ classes. Their example is the following code, which doesn't do very much, but does include the required constructors, destructor, and assignment for a typical class:

```
#include <cstdlib>
class Ogre {
 public:
  Ogre(){ name_ = new char[10]; }
  Ogre(const Ogre &other) = default;
  Ogre& operator=(const Ogre &other) = default;
  ~Ogre(){ delete [] name_; }
 private:
  char* name_;
};

int main(int argc, char **argv) {
  Ogre shrek;                          // Line 1
  Ogre* fiona = new Ogre();            // Line 2
  shrek = *fiona;                      // Line 3
  delete fiona;                        // Line 4
  return EXIT_SUCCESS;                 // Line 5
}
```

(a) (8 points) Below, draw a diagram showing the contents of memory immediately after `Line 3` is executed. Be sure to carefully distinguish between local (stack) variables and heap-allocated memory. Variables that are local to `main` should be drawn in a box labeled main on the stack.

Stack | Heap

Question not used during exam because of potential time pressure to finish everything in 50 min.

(b) (4 points) Does this program execute successfully when it runs? If not, what goes wrong and why? (A couple of sentences should be more than enough.)

Question 7. (2 free points) (All reasonable answers receive the points. All answers are reasonable as long as there is an answer. ☺)

(a) (1 point) What question were you expecting to appear on this exam that wasn't included?

(b) (1 point) Should we include that question on the final exam? (circle or fill in)

Yes

No

Heck No!!

$!@$^*% No !!!!!

Yes, yes, it *must* be included!!!

No opinion / don't care

None of the above. My answer is _____.

Extra space for answers, if needed. Please be sure to label which question(s) are answered here, and be sure to put a note on the question page so the grader will know to look here.

Reference information. Here is a collection of information that might, or might not, be useful while taking the test. You should **remove this page** from the exam. **Do not write answers on this page. It will not be scanned for grading.**

Memory management (<stdlib.h>)

- void * malloc(size_t size)
- void free(void *ptr)
- void * calloc(size_t number, size_t size)
- void * realloc(void *ptr, size_t size)

Strings and characters (<string.h>, <ctype.h>)

Some of the string library functions:

- char* strncpy(*dest*, *src*, *n*), copies exactly *n* characters from *src* to *dst*, adding '\0's at end if the '\0' at the end of the string *src* is found before *n* chars copied.
- char* strcpy(*dest*, *src*), same as strncpy but with no length check
- char* strncat(*dest*, *src, n*), Appends the first *n* characters of *src* to *dst*, plus a terminating null-character. If the length of the C string in *src* is less than *n*, only the content up to the terminating null-character is copied.
- char* strcat(*dest*, *src*), same as strncat but with no length check
- int strncmp(*string1*, *string2*, *n*), <0, =0, >0 if compare <, =, >
- int strcmp(*string1*, *string2*)
- char* strstr(*string, search_string*)
- int strnlen(*s, max_length*), # characters in *s* not including terminating '\0'
- int strlen(*s*)
- Character tests: isupper(*c*), islower(*c*), isalpha(*c*), isdigit(*c*), isspace(*c*)
- Character conversions: toupper(*c*), tolower(*c*)

Files (<stdio.h>)

Some file functions and information:

- Default streams: stdin, stdout, and stderr.
- FILE* fopen(*filename*, *mode*), modes include "r" and "w"
- char* fgets(*line, max_length, file*), returns NULL if eof or error, otherwise reads up to max-1 characters into buffer, including any \n, and adds a \0 at the end
- size_t fread(buf, 1, count, FILE* f)
- size_t fwrite(buf, 1, count, FILE* f)
- int fprintf(format_string, data…, FILE *f)
- int feof(*file*), returns non-zero if end of *file* has been reached
- int ferror(FILE* f), returns non-zero if the error indicator associated with f is set
- int fputs(*line, file*)
- int fclose(*file*)

A few printf format codes: %d (integer), %c (char), %s (char*)

More reference information, C++ this time. **Do not write answers on this page. It will not be scanned for grading.**

C++ strings

If `s` is a string, `s.length()` and `s.size()` return the number of characters in it. Subscripts (`s[i]`) can be used to access individual characters. The usual comparison operators can be used to compare strings, and the operator + can be used to concatenate strings.

C++ STL

- If `lst` is a STL `vector`, then `lst.begin()` and `lst.end()` return iterator values of type `vector<...>::iterator`. STL `lists` and `sets` are similar.
- A STL `map` is a collection of `Pair` objects. If p is a `Pair`, then `p.first` and `p.second` denote its two components. If the `Pair` is stored in a `map`, then `p.first` is the key and `p.second` is the associated value.
- If `m` is a `map`, `m.begin()` and `m.end()` return iterator values. For a `map`, these iterators refer to the `Pair` objects in the `map`.
- If `it` is an iterator, then `*it` can be used to reference the item it currently points to, and `++it` will advance `it` to the next item, if any.
- Some useful operations on STL containers (lists, maps, sets, etc.):
  - `c.clear()` – remove all elements from c
  - `c.size()` – return number of elements in c
  - `c.empty()` – true if number of elements in c is 0, otherwise false
- Additional operations on `vectors`:
  - `c.push_back(x)` – copy x to end of c
- Some additional operations on `maps`:
  - `m.insert(x)` – add copy of `x` to `m` (a key-value pair for a `map`)
  - `m.count(x)` – number of elements with key `x` in `m` (0 or 1)
  - `m[k]` can be used to access the value associated with key `k`. If `m[k]` is read and has never been accessed before, then a <key,value> `Pair` is added to the map with `k` as the key and with a value created by the default constructor for the value type (0 or `nullptr` for primitive types).
- Some additional operations on `sets`
  - `s.insert(x)` – add `x` to `s` if not already present
  - `s.count(x)` – number of copies of `x` in `s` (0 or 1)
- You may use the C++11 `auto` keyword, C++11-style `for`-loops for iterating through containers, and any other features of standard C++11, but you are not required to do so.

Extra copy of the valgrind output from Question 2. You should **remove this page** and use it for reference while solving that problem to minimize page flipping.

Do not write answers on this page. It will not be scanned for grading.

```
==2913707== Memcheck, a memory error detector
==2913707== Copyright (C) 2002-2024, and GNU GPL'd, by Julian Seward et al.
==2913707== Using Valgrind-3.23.0 and LibVEX; rerun with -h for copyright info
==2913707== Command: ./membugs
==2913707==
0x4a770a0
0x4a77530
0x4a77580
==2913707== Invalid read of size 1
==2913707==    at 0x484DE84: strlen (vg_replace_strmem.c:505)
==2913707==    by 0x48CD0F7: __vfprintf_internal (in /usr/lib64/libc.so.6)
==2913707==    by 0x48C14FE: printf (in /usr/lib64/libc.so.6)
==2913707==    by 0x40126D: main (membugs.c:30)
==2913707==  Address 0x4a77586 is 0 bytes after a block of size 6 alloc'd
==2913707==    at 0x484482F: malloc (vg_replace_malloc.c:446)
==2913707==    by 0x4011DF: main (membugs.c:24)
==2913707==
==2913707== Conditional jump or move depends on uninitialised value(s)
==2913707==    at 0x48CC3BB: __vfprintf_internal (in /usr/lib64/libc.so.6)
==2913707==    by 0x48C14FE: printf (in /usr/lib64/libc.so.6)
==2913707==    by 0x40126D: main (membugs.c:30)
==2913707==
==2913707== Use of uninitialised value of size 8
==2913707==    at 0x48C091B: _itoa_word (in /usr/lib64/libc.so.6)
==2913707==    by 0x48CBFAB: __vfprintf_internal (in /usr/lib64/libc.so.6)
==2913707==    by 0x48C14FE: printf (in /usr/lib64/libc.so.6)
==2913707==    by 0x40126D: main (membugs.c:30)
==2913707==
==2913707== Conditional jump or move depends on uninitialised value(s)
==2913707==    at 0x48C092C: _itoa_word (in /usr/lib64/libc.so.6)
==2913707==    by 0x48CBFAB: __vfprintf_internal (in /usr/lib64/libc.so.6)
==2913707==    by 0x48C14FE: printf (in /usr/lib64/libc.so.6)
==2913707==    by 0x40126D: main (membugs.c:30)
```

(continued on next page)

Remainder of the valgrind output from question 2. **Do not write answers on this page. It will not be scanned for grading.**

```
==2913707==
==2913707== Conditional jump or move depends on uninitialised value(s)
==2913707==    at 0x48CC8A3: __vfprintf_internal (in /usr/lib64/libc.so.6)
==2913707==    by 0x48C14FE: printf (in /usr/lib64/libc.so.6)
==2913707==    by 0x40126D: main (membugs.c:30)
==2913707==
==2913707== Conditional jump or move depends on uninitialised value(s)
==2913707==    at 0x48CC0C7: __vfprintf_internal (in /usr/lib64/libc.so.6)
==2913707==    by 0x48C14FE: printf (in /usr/lib64/libc.so.6)
==2913707==    by 0x40126D: main (membugs.c:30)
==2913707==
apples has 32768 letters
0x4a77580
0x4a77580
==2913707== Invalid free() / delete / delete[] / realloc()
==2913707==    at 0x4847B4C: free (vg_replace_malloc.c:989)
==2913707==    by 0x4012C4: main (membugs.c:35)
==2913707==  Address 0x4a77580 is 0 bytes inside a block of size 6 free'd
==2913707==    at 0x4847B4C: free (vg_replace_malloc.c:989)
==2913707==    by 0x4012C4: main (membugs.c:35)
==2913707==  Block was alloc'd at
==2913707==    at 0x484482F: malloc (vg_replace_malloc.c:446)
==2913707==    by 0x4011DF: main (membugs.c:24)
==2913707==
0x4a77580
==2913707==
==2913707== HEAP SUMMARY:
==2913707==     in use at exit: 12 bytes in 2 blocks
==2913707==   total heap usage: 5 allocs, 5 frees, 1,066 bytes allocated
==2913707==
==2913707== 12 bytes in 2 blocks are definitely lost in loss record 1 of 1
==2913707==    at 0x484482F: malloc (vg_replace_malloc.c:446)
==2913707==    by 0x4011DF: main (membugs.c:24)
==2913707==
==2913707== LEAK SUMMARY:
==2913707==    definitely lost: 12 bytes in 2 blocks
==2913707==    indirectly lost: 0 bytes in 0 blocks
==2913707==      possibly lost: 0 bytes in 0 blocks
==2913707==    still reachable: 0 bytes in 0 blocks
==2913707==         suppressed: 0 bytes in 0 blocks
==2913707==
==2913707== Use --track-origins=yes to see where uninitialised values come from
==2913707== For lists of detected and suppressed errors, rerun with: -s
==2913707== ERROR SUMMARY: 17 errors from 8 contexts (suppressed: 0 from 0)
```