CMPUT201W20B2 Week 8

Abram Hindle

March 3, 2020

Contents

1 Week8

1.1 Copyright Statement

1.1.1 License

1.1.2 Hazel Code is licensed under AGPL3.0+

Hazel's code is also found here `https://github.com/hazelybell/examples/tree/C-2020-01`

Hazel code is licensed: The example code is licensed under the AGPL3+ license, unless otherwise noted.

1.2 Init ORG-MODE

```
;; I need this for org-mode to work well

;(require 'ob-sh)
(require 'ob-shell)
; (org-babel-do-load-languages 'org-babel-load-languages '((sh . t)))
(org-babel-do-load-languages 'org-babel-load-languages '((shell . t)))
(org-babel-do-load-languages 'org-babel-load-languages '((C . t)))
(org-babel-do-load-languages 'org-babel-load-languages '((python . t)))
(setq org-src-fontify-natively t)
(setq org-confirm-babel-evaluate nil) ;; danger!
(custom-set-faces
 ;; custom-set-faces was added by Custom.
 ;; If you edit it by hand, you could mess it up, so be careful.
 ;; Your init file should contain only one such instance.
 ;; If there is more than one, they won't work right.
 '(org-block ((t (:inherit shadow :foreground "black")))))
```

1.2.1 Org export

```
(org-html-export-to-html)
(org-latex-export-to-pdf)
(org-ascii-export-to-ascii)
(org-odt-export-odf)
```

1.3 Org Template

Copy and paste this to demo C

```
#include <stdio.h>

int main(int argc, char**argv) {
    return 0;
}
```

1.4 Remember how to compile?

gcc -std=c99 -Wall -pedantic -Werror -o programname programname.c

1.5 Malloc continued!

Continued.

1.5.1 free

What happens if we don't free?
Our program can get bigger!

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

int * testAllocArray(int arrLen) {
  int* array = calloc( sizeof(int), arrLen );
  assert(array!=NULL);
  for(int idx=0; idx<arrLen; idx++) {
    array[idx] = idx;
  }
  return array;
```

```
}
int main() {
    for (int i = 1; i < 10000000; i+=1*1024*1024) {
        int * bigArray = testAllocArray( i );
        printf("%u ints allocated!\n",1+bigArray[i-1]);
        printf("%lu bytes!\n", sizeof(int)*i);
        // free(bigArray); // remember to free it when done!
    }
}
```

```
1 ints allocated!
4 bytes!
1048577 ints allocated!
4194308 bytes!
2097153 ints allocated!
8388612 bytes!
3145729 ints allocated!
12582916 bytes!
4194305 ints allocated!
16777220 bytes!
5242881 ints allocated!
20971524 bytes!
6291457 ints allocated!
25165828 bytes!
7340033 ints allocated!
29360132 bytes!
8388609 ints allocated!
33554436 bytes!
9437185 ints allocated!
37748740 bytes!
```

Valgrind is a memory leak detector. It analyzes memory allocations and warns us about mistakes.

Valgrind will show us that we're leaking memory (losing track of it and not freeing it).

```
gcc -std=c99 -Wall -pedantic -Werror -o nofree ./nofree.c
valgrind ./nofree 2>&1
echo now let\'s leak check
valgrind --leak-check=full ./nofree 2>&1
```

```
==28799== Memcheck, a memory error detector
==28799== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==28799== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==28799== Command: ./nofree
==28799==
1 ints allocated!
4 bytes!
1048577 ints allocated!
4194308 bytes!
2097153 ints allocated!
8388612 bytes!
3145729 ints allocated!
12582916 bytes!
4194305 ints allocated!
16777220 bytes!
5242881 ints allocated!
20971524 bytes!
6291457 ints allocated!
25165828 bytes!
7340033 ints allocated!
29360132 bytes!
8388609 ints allocated!
33554436 bytes!
9437185 ints allocated!
37748740 bytes!
==28799==
==28799== HEAP SUMMARY:
==28799==     in use at exit: 188,743,720 bytes in 10 blocks
==28799==   total heap usage: 11 allocs, 1 frees, 188,747,816 bytes allocated
==28799==
==28799== LEAK SUMMARY:
==28799==    definitely lost: 100,663,320 bytes in 6 blocks
==28799==    indirectly lost: 0 bytes in 0 blocks
==28799==      possibly lost: 88,080,400 bytes in 4 blocks
==28799==    still reachable: 0 bytes in 0 blocks
==28799==         suppressed: 0 bytes in 0 blocks
==28799== Rerun with --leak-check=full to see details of leaked memory
==28799==
==28799== For counts of detected and suppressed errors, rerun with: -v
==28799== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

```
now let's leak check
==28801== Memcheck, a memory error detector
==28801== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==28801== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==28801== Command: ./nofree
==28801==
1 ints allocated!
4 bytes!
1048577 ints allocated!
4194308 bytes!
2097153 ints allocated!
8388612 bytes!
3145729 ints allocated!
12582916 bytes!
4194305 ints allocated!
16777220 bytes!
5242881 ints allocated!
20971524 bytes!
6291457 ints allocated!
25165828 bytes!
7340033 ints allocated!
29360132 bytes!
8388609 ints allocated!
33554436 bytes!
9437185 ints allocated!
37748740 bytes!
==28801==
==28801== HEAP SUMMARY:
==28801==     in use at exit: 188,743,720 bytes in 10 blocks
==28801==   total heap usage: 11 allocs, 1 frees, 188,747,816 bytes allocated
==28801==
==28801== 88,080,400 bytes in 4 blocks are possibly lost in loss record 1 of 2
==28801==    at 0x4C31B25: calloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux
==28801==    by 0x1086F6: testAllocArray (in /home/hindle1/projects/CMPUT201W20/2020-0
==28801==    by 0x10876F: main (in /home/hindle1/projects/CMPUT201W20/2020-01/CMPUT201
==28801==
==28801== 100,663,320 bytes in 6 blocks are definitely lost in loss record 2 of 2
==28801==    at 0x4C31B25: calloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux
==28801==    by 0x1086F6: testAllocArray (in /home/hindle1/projects/CMPUT201W20/2020-0
==28801==    by 0x10876F: main (in /home/hindle1/projects/CMPUT201W20/2020-01/CMPUT201
```

```
==28801==
==28801== LEAK SUMMARY:
==28801==    definitely lost: 100,663,320 bytes in 6 blocks
==28801==    indirectly lost: 0 bytes in 0 blocks
==28801==      possibly lost: 88,080,400 bytes in 4 blocks
==28801==    still reachable: 0 bytes in 0 blocks
==28801==         suppressed: 0 bytes in 0 blocks
==28801==
==28801== For counts of detected and suppressed errors, rerun with: -v
==28801== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

1.5.2 Malloc and structs

Mallocs are often used with arrays of structs. You need to get the sizeof the struct.

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

enum card_face {
    ACE = 1,
    FACE2,
    FACE3,
    FACE4,
    FACE5,
    FACE6,
    FACE7,
    FACE8,
    FACE9,
    FACE10,
    JACK,
    QUEEN,
    KING,
};

typedef enum card_face CardFace;

#define NFACES 13
```

```
#define NFACEOFF 1

enum card_suit {
    CLUBS,
    HEARTS,
    DIAMONDS,
    SPADES
};

typedef enum card_suit CardSuit;

#define NSUIT 4

struct playing_card {
    CardFace face;
    CardSuit suit;
};

typedef struct playing_card PlayingCard;

#define HANDSIZE 5

bool isFlush(PlayingCard hand[HANDSIZE]) {
    CardSuit suit = hand[0].suit;
    for (int i = 1;  i < HANDSIZE; i++ ) {
        if (suit != hand[i].suit) {
            return false;
        }
    }
    return true;
}

PlayingCard randomCard() {
    PlayingCard card = {ACE, CLUBS};
    card.face = NFACEOFF + ( rand() % NFACES );
    card.suit =  rand() % NSUIT;
    return card;
}
int main() {
    srand(time(NULL));
```

```
    const int N = 1000000;
    PlayingCard * bigHand = malloc(sizeof(PlayingCard)*N);
    for (int i = 0; i < N; i++) {
        bigHand[i] = randomCard();
    }
    int flushes = 0;
    for (int i = 0; i < N - HANDSIZE; i+=HANDSIZE) {
        if (isFlush(bigHand + i)) {
            if (flushes < 10) { // reduce printing
                printf("Flush found at card %d\n", i);
                printf("Suit %d\n", bigHand[i].suit);
            }
            flushes++;
        }
    }
    printf("We found %d flushes out of %d hands: %f\n", flushes, N/HANDSIZE, flushes/(i
}
```

```
Flush found at card 225
Suit 2
Flush found at card 1370
Suit 1
Flush found at card 4095
Suit 1
Flush found at card 8160
Suit 1
Flush found at card 8665
Suit 0
Flush found at card 10025
Suit 1
Flush found at card 12900
Suit 0
Flush found at card 13085
Suit 0
Flush found at card 14855
Suit 3
Flush found at card 15145
Suit 2
We found 799 flushes out of 200000 hands: 0.003995
```

1.5.3 Malloc Array of Array versus 2D

So instead of allocating a big block and carving a 2D array out of it we could just allocate each row and make an array of arrays.

Try playing with the order of allocation of rows. Does it affect the result?

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

// This example compares using malloc to get space for a 2-D array vs using malloc to m

int * alloc2d(size_t n) {
    // we can just do 1 malloc()
    return (int *) malloc(n * n * sizeof(int));
}

int ** alloc_aoa(size_t n) {
    // we have to do 1 + n malloc()s
    int ** p = malloc(n * sizeof(int *));
    // we don't need to do them in order...
    for (size_t i = 0; i < n; i++) {
        p[i] = malloc(n * sizeof(int));
    }
    return p;
}

void free2d(int * p) {
    // we can just do 1 free()
    free(p);
}

void free_aoa(size_t n, int ** p) {
    // we have to do n + 1 free()s
    for (size_t i = 0; i < n; i++) {
        free(p[i]);
    }
    free(p);
}

int get2d(size_t n, int * p, size_t i, size_t j) {
    return p[i * n + j];
```

```
}

int get_aoa(int **p, size_t i, size_t j) {
    return p[i][j];
}

int set2d(size_t n, int * p, size_t i, size_t j, int v) {
    return p[i * n + j] = v;
}

int set_aoa(int **p, size_t i, size_t j, int v) {
    return p[i][j] = v;
}

int main(int argc, char **argv) {
    srand(1);
    // printf("I'm going to make space for a big, square table in memory.\n");
    // printf("How many rows and columns would you like to make space for? ");
    size_t n;
    // int r = scanf("%zu", &n);
    n = 30;
    if (n != 1) {
        printf("Sorry, I couldn't understand that :(\n");
        exit(1);
    }
    // allocate them
    int *p2d = alloc2d(n);
    int **aoa = alloc_aoa(n);
    // initialize them
    for (size_t i = 0; i < n; i++) {
        for (size_t j = 0; j < n; j++) {
            set2d(n, p2d, i, j, rand() % 10);
            set_aoa(aoa, i, j, rand() % 10);
        }
    }
    // print them out
    printf("2d:\n");
    for (size_t i = 0; i < n; i++) {
        for (size_t j = 0; j < n; j++) {
            int x = get2d(n, p2d, i, j);
```

```
            printf("%d ", x);
        }
        printf("\n");
    }
    printf("aoa:\n");
    for (size_t i = 0; i < n; i++) {
        for (size_t j = 0; j < n; j++) {
            int x = get_aoa(aoa, i, j);
            printf("%d ", x);
        }
        printf("\n");
    }
    // free them
    free2d(p2d);
    free_aoa(n, aoa);
}
```

```
I'm going to make space for a big, square table in memory.
How many rows and columns would you like to make space for? Sorry, I couldn't understar
2d:
3 7 3 6 9 2 0 3 0 2 1 7 2 2 7 9 2 9 3 1 9 1 4 8 5 3 1 6 2 6
5 4 6 6 3 4 2 4 4 3 7 6 8 3 4 2 6 9 6 4 5 4 7 7 7 2 1 6 5 4
0 1 7 1 9 7 7 6 6 9 8 2 3 0 8 0 6 8 6 1 9 4 1 3 4 4 7 3 7 9
2 7 5 4 8 9 5 8 3 8 6 3 3 6 4 8 9 7 4 0 0 2 4 5 4 9 2 7 5 8
2 9 6 0 1 5 1 8 0 4 2 8 2 4 2 0 2 9 8 3 1 3 0 9 9 9 3 0 6 4
0 6 6 5 9 7 8 9 6 2 6 3 1 9 1 9 0 5 7 4 0 2 6 0 2 2 5 2 0 8
8 4 9 9 2 4 9 3 0 0 9 3 1 4 1 6 4 2 4 2 8 2 8 6 3 3 3 0 7 8
0 8 9 3 3 3 6 2 5 7 6 4 0 8 0 6 4 9 9 8 0 7 9 5 9 5 4 9 5 3
7 8 9 7 2 3 9 2 1 6 1 0 3 1 0 6 7 0 4 4 5 2 0 6 6 8 6 7 1 1
7 2 4 2 2 0 9 5 0 7 8 0 6 6 9 5 7 5 3 3 9 7 7 1 0 8 5 4 7 3
0 7 9 2 3 1 2 2 7 1 4 7 1 7 4 8 1 6 1 6 8 8 0 2 7 6 6 7 7 9
7 6 8 3 4 5 1 5 9 3 5 2 7 3 6 6 3 4 9 2 8 0 4 6 7 3 3 5 0 7
3 0 0 1 3 9 4 5 8 5 5 9 7 3 6 5 6 0 1 2 9 0 2 4 3 8 3 0 3 9
7 2 2 4 8 0 9 2 1 3 2 4 1 5 1 9 1 3 7 8 7 4 4 1 8 2 9 6 6 9
0 9 1 8 6 7 7 2 1 0 0 0 3 4 1 0 2 7 6 4 2 7 4 6 7 5 2 3 4 9
2 1 3 2 5 5 0 4 6 2 8 5 6 8 7 2 0 8 5 7 8 3 7 7 9 1 0 9 8 3
0 9 1 7 7 2 1 8 4 6 6 4 8 8 5 4 0 7 2 2 3 9 1 5 4 2 1 2 2 9
4 5 1 0 1 7 9 1 7 0 0 5 9 1 1 0 8 4 2 4 9 2 9 0 4 9 5 6 3 9
2 3 9 1 4 8 7 3 9 5 8 0 3 1 7 5 1 3 0 5 2 9 9 9 1 3 3 4 1 6
7 2 2 1 4 8 3 7 3 2 3 6 1 6 0 5 5 9 8 2 9 1 0 6 9 8 8 3 0 5
```

3 8 1 9 0 5 4 4 9 9 3 3 7 4 9 9 2 6 9 6 1 3 2 3 9 4 4 9 8 2
5 3 4 5 7 9 7 7 9 5 4 7 3 2 2 3 1 8 0 2 9 9 3 8 6 7 7 1 0 4
3 3 7 1 9 6 9 5 1 9 1 2 0 3 1 7 8 0 4 3 9 4 5 2 7 8 9 3 8 4
6 8 5 1 6 8 6 5 6 1 3 5 6 4 6 7 3 9 0 2 9 3 5 7 7 6 4 3 2 6
9 5 3 4 1 1 9 5 2 9 7 4 1 1 8 4 3 3 7 3 8 0 8 8 3 5 5 2 8 2
3 7 7 6 2 7 3 2 5 7 9 1 4 5 8 3 5 1 5 0 8 9 9 6 5 5 0 2 9 2
6 5 8 7 6 2 9 0 7 5 4 0 8 4 4 8 2 6 2 7 4 6 4 4 5 6 3 7 2 0
9 1 4 5 2 0 3 1 5 4 0 3 9 4 3 2 5 8 1 1 8 3 9 5 4 6 2 0 3 7
3 1 4 1 6 3 7 0 4 3 7 9 3 2 9 5 0 3 9 5 3 2 7 7 0 6 5 8 9 7
0 1 3 7 2 1 3 8 8 8 8 9 3 4 7 3 6 2 2 5 4 4 1 3 8 3 9 4 1 0
aoa:
6 5 5 2 1 7 9 6 6 6 8 9 0 3 5 2 8 7 6 2 3 9 7 4 0 6 0 3 0 1
5 7 5 9 7 5 5 7 4 0 8 8 4 1 9 0 8 2 6 9 0 8 1 2 2 6 0 1 9 9
9 7 1 5 7 6 3 5 3 4 1 9 9 8 5 9 3 5 1 5 8 8 0 0 4 4 6 1 5 6
1 8 7 1 5 7 3 8 1 9 4 3 8 0 8 8 7 6 3 3 9 5 0 9 6 2 4 7 4 1
8 3 8 2 0 1 0 5 6 6 5 6 8 7 4 6 9 0 1 1 0 4 3 1 6 3 8 5 6 0
4 2 7 6 8 2 2 9 0 7 1 2 5 9 4 1 7 8 0 8 4 9 1 4 2 0 5 9 2 3
0 0 1 6 5 4 9 6 5 2 4 5 7 3 4 9 2 6 1 8 9 8 8 8 8 3 8 4 6 9
6 7 0 3 7 2 5 6 8 9 0 1 4 7 8 2 7 3 2 3 1 8 1 4 2 7 9 4 9 5
0 1 9 8 5 4 0 0 9 2 2 7 1 9 5 7 4 6 7 8 8 6 6 4 2 9 0 0 0 3
7 6 5 0 9 9 4 1 3 8 6 4 7 0 7 9 8 3 8 7 3 8 4 9 9 8 8 3 1 8
9 9 3 4 7 2 0 1 5 7 1 1 1 0 0 5 6 2 9 4 0 1 2 9 5 4 3 9 4 1
0 0 5 9 1 4 5 4 8 8 2 2 0 4 3 3 4 3 7 5 9 2 7 5 1 3 8 1 8 6
5 8 4 1 5 3 1 0 3 6 9 0 6 7 1 0 5 8 2 6 1 4 7 0 2 0 7 0 4 2
4 5 4 3 6 8 2 3 8 4 2 5 7 7 6 8 3 3 9 6 0 8 8 6 5 1 9 0 4 9
8 3 4 9 7 3 1 2 5 9 4 1 7 1 3 3 1 5 5 2 1 2 1 5 8 9 7 6 7 7
2 6 0 1 6 0 3 6 0 5 9 0 0 3 8 1 5 5 0 3 2 0 7 6 1 9 8 8 0 7
6 2 7 9 6 7 5 8 5 5 8 8 3 7 2 5 5 3 7 1 4 4 9 7 1 2 6 0 2 7
3 6 4 3 2 7 8 0 6 1 2 1 7 3 2 6 7 9 4 5 1 8 6 6 0 4 4 6 9 5
1 0 9 3 5 5 3 8 5 3 6 3 6 8 0 1 0 0 4 4 4 9 4 8 6 9 3 6 5 1
2 9 8 2 7 6 7 2 7 5 7 8 3 4 3 8 0 9 0 4 0 2 0 3 0 3 7 1 0 0
1 0 7 1 3 9 8 6 2 0 0 3 9 9 1 4 0 5 5 1 4 7 7 3 2 4 9 3 3 9
4 9 9 5 3 0 2 2 0 0 1 9 6 1 5 9 8 7 5 7 1 6 6 4 6 2 4 0 6 4
7 4 2 7 5 8 5 2 5 9 6 1 5 2 9 6 2 6 3 6 0 8 1 9 3 0 2 1 7 1
3 5 0 2 4 5 2 2 9 3 1 2 9 4 0 4 7 0 2 6 0 5 8 1 0 0 1 0 9 0
3 4 6 3 9 0 4 6 5 1 7 1 9 3 7 9 1 8 9 8 4 0 6 2 8 0 9 6 5 8
6 8 2 6 9 0 7 3 1 8 4 6 3 4 7 3 0 4 7 7 9 3 4 4 5 6 6 6 9 9
5 3 6 3 0 6 3 8 6 2 0 6 5 9 6 3 3 2 4 0 9 5 6 2 1 1 7 1 1 8
0 3 8 8 2 6 6 0 7 2 0 3 0 3 4 4 3 1 3 5 1 3 7 4 9 7 1 1 7 6
9 0 1 8 4 4 7 7 5 0 2 9 0 7 9 2 8 5 6 6 0 0 4 3 1 7 7 8 0 8

3 0 6 3 2 5 3 2 5 0 6 3 7 3 1 9 4 0 9 7 6 9 2 1 1 8 2 5 0 1

1.5.4 Malloc array of arrays structs?

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

enum card_face {
    ACE = 1,
    FACE2,
    FACE3,
    FACE4,
    FACE5,
    FACE6,
    FACE7,
    FACE8,
    FACE9,
    FACE10,
    JACK,
    QUEEN,
    KING,
};

typedef enum card_face CardFace;

#define NFACES 13
#define NFACEOFF 1

enum card_suit {
    CLUBS,
    HEARTS,
    DIAMONDS,
    SPADES
};

typedef enum card_suit CardSuit;

#define NSUIT 4
```

```
struct playing_card {
    CardFace face;
    CardSuit suit;
};

typedef struct playing_card PlayingCard;

#define HANDSIZE 5

bool isFlush(PlayingCard hand[HANDSIZE]) {
    CardSuit suit = hand[0].suit;
    for (int i = 1;  i < HANDSIZE; i++ ) {
        if (suit != hand[i].suit) {
            return false;
        }
    }
    return true;
}

PlayingCard randomCard() {
    PlayingCard card = {ACE, CLUBS};
    card.face = NFACEOFF + ( rand() % NFACES );
    card.suit =  rand() % NSUIT;
    return card;
}
int main() {
    srand(time(NULL));
    const int HANDS = 1000000;
    PlayingCard * hands = malloc(sizeof(PlayingCard)*HANDS*HANDSIZE);
    for (int i = 0; i < HANDS*HANDSIZE; i++) {
        hands[i] = randomCard();
    }
    int flushes = 0;
    for (int i = 0; i < HANDS; i++) {
        if (isFlush(hands + i*HANDSIZE)) {
            if (flushes < 10) { // reduce printing
                printf("Flush found at card %d\n", i);
                printf("Suit %d\n", hands[i].suit);
            }
```

```
            flushes++;
        }
    }
    printf("We found %d flushes out of %d hands: %f\n", flushes, HANDS, flushes/(float)
}
```

```
Flush found at card 19
Suit 3
Flush found at card 340
Suit 1
Flush found at card 450
Suit 0
Flush found at card 870
Suit 0
Flush found at card 918
Suit 1
Flush found at card 932
Suit 2
Flush found at card 970
Suit 2
Flush found at card 1375
Suit 0
Flush found at card 1438
Suit 3
Flush found at card 1631
Suit 2
We found 3902 flushes out of 1000000 hands: 0.003902
```

That's kind of gross, let's model our hands as arrays of 5 cards instead.

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

enum card_face {
    ACE = 1,
    FACE2,
    FACE3,
    FACE4,
```

```
    FACE5,
    FACE6,
    FACE7,
    FACE8,
    FACE9,
    FACE10,
    JACK,
    QUEEN,
    KING,
};

typedef enum card_face CardFace;

#define NFACES 13
#define NFACEOFF 1

enum card_suit {
    CLUBS,
    HEARTS,
    DIAMONDS,
    SPADES
};

typedef enum card_suit CardSuit;

#define NSUIT 4

struct playing_card {
    CardFace face;
    CardSuit suit;
};

typedef struct playing_card PlayingCard;

#define HANDSIZE 5

bool isFlush(PlayingCard hand[HANDSIZE]) {
    CardSuit suit = hand[0].suit;
    for (int i = 1;  i < HANDSIZE; i++ ) {
        if (suit != hand[i].suit) {
```

```
            return false;
        }
    }
    return true;
}

PlayingCard randomCard() {
    PlayingCard card = {ACE, CLUBS};
    card.face = NFACEOFF + ( rand() % NFACES );
    card.suit =  rand() % NSUIT;
    return card;
}
int main() {
    srand(time(NULL));
    const int HANDS = 1000000;
    // Pointer to arrays
    PlayingCard (*hands)[5] = malloc(sizeof(PlayingCard[5])*HANDS);
    for (int i = 0; i < HANDS; i++) {
        for (int j = 0; j < HANDSIZE; j++) {
            hands[i][j] = randomCard();
        }
    }
    int flushes = 0;
    for (int i = 0; i < HANDS; i++) {
        if (isFlush(hands[i])) {
            if (flushes < 10) { // reduce printing
                printf("Flush found at card %d\n", i);
                printf("Suit %d\n", hands[i][0].suit);
            }
            flushes++;
        }
    }
    printf("We found %d flushes out of %d hands: %f\n", flushes, HANDS, flushes/(float)
}
```

```
Flush found at card 223
Suit 0
Flush found at card 323
Suit 1
Flush found at card 335
```

```
Suit 3
Flush found at card 407
Suit 1
Flush found at card 896
Suit 3
Flush found at card 1027
Suit 3
Flush found at card 1124
Suit 0
Flush found at card 1279
Suit 0
Flush found at card 1301
Suit 0
Flush found at card 1734
Suit 3
We found 3855 flushes out of 1000000 hands: 0.003855
```

- Remember to tangle this to write to disk

```
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <assert.h>

enum card_face {
    ACE = 1,
    FACE2,
    FACE3,
    FACE4,
    FACE5,
    FACE6,
    FACE7,
    FACE8,
    FACE9,
    FACE10,
    JACK,
    QUEEN,
    KING,
};
```

```
typedef enum card_face CardFace;

#define NFACES 13
#define NFACEOFF 1

enum card_suit {
    CLUBS,
    HEARTS,
    DIAMONDS,
    SPADES
};

typedef enum card_suit CardSuit;

#define NSUIT 4

struct playing_card {
    CardFace face;
    CardSuit suit;
};

typedef struct playing_card PlayingCard;

#define HANDSIZE 5

bool isFlush(PlayingCard hand[HANDSIZE]) {
    CardSuit suit = hand[0].suit;
    for (int i = 1;  i < HANDSIZE; i++ ) {
        if (suit != hand[i].suit) {
            return false;
        }
    }
    return true;
}

PlayingCard randomCard() {
    PlayingCard card = {ACE, CLUBS};
    card.face = NFACEOFF + ( rand() % NFACES );
    card.suit =  rand() % NSUIT;
```

```
    return card;
}

PlayingCard * allocateHand() {
    PlayingCard * hand = malloc(sizeof(PlayingCard[HANDSIZE]));
    assert(hand!=NULL);
    return hand;
}

void randomizeHand( PlayingCard hand[HANDSIZE]) {
    for (int i = 0; i < HANDSIZE; i++) {
        hand[i] = randomCard();
    }
}

int main() {
    srand(time(NULL));
    const int HANDS = 1000000;
    // Pointer to arrays of arrays
    PlayingCard **hands = malloc(sizeof(PlayingCard(*)[5]) * HANDS);
    for (int i = 0; i < HANDS; i++) {
        hands[i] = allocateHand();
        randomizeHand( hands[i] );
    }
    int flushes = 0;
    for (int i = 0; i < HANDS; i++) {
        if (isFlush(hands[i])) {
            if (flushes < 10) { // reduce printing
                printf("Flush found at card %d\n", i);
                printf("Suit %d\n", hands[i][0].suit);
            }
            flushes++;
        }
    }
    printf("We found %d flushes out of %d hands: %f\n", flushes, HANDS, flushes/(float)
    for (int i = 0; i < HANDS; i++) {
        // comment these out to try valgrind
        //free(hands[i]);
    }
    // comment these out to try valgrind
```

```
    // free(hands);
}

Flush found at card 16
Suit 2
Flush found at card 307
Suit 2
Flush found at card 1199
Suit 2
Flush found at card 1485
Suit 0
Flush found at card 1516
Suit 1
Flush found at card 1520
Suit 2
Flush found at card 1542
Suit 2
Flush found at card 1606
Suit 3
Flush found at card 1780
Suit 1
Flush found at card 1829
Suit 3
We found 3910 flushes out of 1000000 hands: 0.003910

gcc -std=c99 -pedantic -Wall -Wextra -ftrapv -ggdb3 -o cards-aoa ./cards-aoa.c
valgrind --leak-check=full ./cards-aoa 2>&1

==12965== Memcheck, a memory error detector
==12965== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==12965== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==12965== Command: ./cards-aoa
==12965==
Flush found at card 194
Suit 1
Flush found at card 348
Suit 2
Flush found at card 460
Suit 3
Flush found at card 1697
```

```
Suit 1
Flush found at card 1817
Suit 0
Flush found at card 2480
Suit 0
Flush found at card 2504
Suit 1
Flush found at card 2576
Suit 1
Flush found at card 2904
Suit 2
Flush found at card 3383
Suit 0
We found 3987 flushes out of 1000000 hands: 0.003987
==12965==
==12965== HEAP SUMMARY:
==12965==     in use at exit: 48,000,000 bytes in 1,000,001 blocks
==12965==   total heap usage: 1,000,002 allocs, 1 frees, 48,004,096 bytes allocated
==12965==
==12965== 80 bytes in 2 blocks are possibly lost in loss record 1 of 3
==12965==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux
==12965==    by 0x1088E2: allocateHand (cards-aoa.c:67)
==12965==    by 0x1089C6: main (cards-aoa.c:84)
==12965==
==12965== 47,999,920 (8,000,000 direct, 39,999,920 indirect) bytes in 1 blocks are defi
==12965==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux
==12965==    by 0x10899A: main (cards-aoa.c:82)
==12965==
==12965== LEAK SUMMARY:
==12965==    definitely lost: 8,000,000 bytes in 1 blocks
==12965==    indirectly lost: 39,999,920 bytes in 999,998 blocks
==12965==      possibly lost: 80 bytes in 2 blocks
==12965==    still reachable: 0 bytes in 0 blocks
==12965==         suppressed: 0 bytes in 0 blocks
==12965==
==12965== For counts of detected and suppressed errors, rerun with: -v
==12965== ERROR SUMMARY: 2 errors from 2 contexts (suppressed: 0 from 0)
```

1.5.5 Using pointers for protection

./stack.c

```
#define _POSIX_C_SOURCE 200809L // <-- needed for getline
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* Let's define Stack as a pointer to a struct,
 * which itself contains the pointer to the actual
 * data on the stack, which are pointers to chars (strings).
 *
 * This is so that when we realloc() and update elts,
 * we don't have to worry about some other piece of code
 * having the old value of elts.
 *
 * If we didn't hide our pointer that gets realloc'd behind
 * another pointer, it is easy to have an old copy of the
 * realloc'd pointer (which is now invalid) floating around.
 *
 * But by putting it behind a pointer, new_stack() can
 * create the single copy of the struct, which contains
 * the elts pointer that changes. Since the sizeof the
 * actual struct never changes, we never have to realloc
 * that pointer, so we can ensure we only have one version
 * of elts at all times. This is similar to how
 * Java/Python/JS handle arrays internally.
 *
 */

// OK so Stack is pointer of struct stack NOT struct stack.
typedef struct stack {
    size_t size;
    char ** elts;
} * Stack;

void show_stack(Stack stack) {
    printf("Stack %p: %zu items starting at %p\n",
           (void *) stack,
```

```
            stack->size,
            (void *) stack->elts
    );
}

// This is a good style, new_object, or object_create
Stack new_stack() {
    /* Constructor */
    Stack new = malloc(sizeof(*new));
    new->size = 0;
    new->elts = NULL;
    show_stack(new);
    return new;
}

/* this function deduplicates code from push and pop */
void resize(Stack stack, size_t new_size) {
    stack->elts = realloc(
        stack->elts,
        sizeof(char *) * new_size
    );

    /* make sure any new elements are initialized */
    size_t first_new_elt = stack->size;
    for (size_t idx = first_new_elt;
         idx < new_size;
         idx++) {
        stack->elts[idx] = NULL;
    }

    stack->size = new_size;
}

void push(Stack stack, char * string) {
    resize(stack, stack->size + 1);
    stack->elts[stack->size-1] = string;
    show_stack(stack);
}

char * pop(Stack stack) {
```

```
    if (stack->size == 0) {
        abort();
    }
    char * string = stack->elts[stack->size-1];
    resize(stack, stack->size - 1);
    show_stack(stack);
    return string;
}

/* Destructor */
void free_stack(Stack stack) {
    resize(stack, 0);
    free(stack);
}

char * checked_getline() {
    char * line = NULL;
    size_t alloc_len = 0;
    ssize_t got = getline(&line, &alloc_len, stdin);
    if (got < 0) {
        if (line != NULL) {
            free(line);
        }
        return NULL;
    } else {
        return line;
    }
}

void push_input_lines(Stack stack) {
    printf("Enter some lines. Press ctrl-d (EOF) to end.\n");
    char * line = NULL;
    while ((line = checked_getline()) != NULL) {
        push(stack, line);
    }
}

void pop_lines(Stack stack) {
    while (stack->size > 0) {
        char * line = pop(stack);
```

```
        printf(line);
        free(line);
    }
}

int main() {
    Stack stack1 = new_stack();
    Stack stack2 = stack1;
/* Because stack is a pointer, stack1 and stack2 are
 * actually the same stack!
 * Because the actual struct doesn't need to change size,
 * these pointers will be valid until we call free_stack()
 */
    push_input_lines(stack1);
    pop_lines(stack2);
    free_stack(stack1);
    return 0;
}

Stack 0x56362ee8c260: 0 items starting at (nil)
Enter some lines. Press ctrl-d (EOF) to end.

gcc -std=c99 -pedantic -Wall -Wextra -ftrapv -ggdb3 -o stack ./stack.c
seq 9990 9999 | ./stack

Stack 0x55f488145260: 0 items starting at (nil)
Enter some lines. Press ctrl-d (EOF) to end.
Stack 0x55f488145260: 1 items starting at 0x55f488147320
Stack 0x55f488145260: 2 items starting at 0x55f488147320
Stack 0x55f488145260: 3 items starting at 0x55f488147320
Stack 0x55f488145260: 4 items starting at 0x55f4881474c0
Stack 0x55f488145260: 5 items starting at 0x55f4881474c0
Stack 0x55f488145260: 6 items starting at 0x55f4881475f0
Stack 0x55f488145260: 7 items starting at 0x55f4881475f0
Stack 0x55f488145260: 8 items starting at 0x55f488147730
Stack 0x55f488145260: 9 items starting at 0x55f488147730
Stack 0x55f488145260: 10 items starting at 0x55f488147880
Stack 0x55f488145260: 9 items starting at 0x55f488147880
9999
Stack 0x55f488145260: 8 items starting at 0x55f488147880
```

```
9998
Stack 0x55f488145260: 7 items starting at 0x55f488147880
9997
Stack 0x55f488145260: 6 items starting at 0x55f488147880
9996
Stack 0x55f488145260: 5 items starting at 0x55f488147880
9995
Stack 0x55f488145260: 4 items starting at 0x55f488147880
9994
Stack 0x55f488145260: 3 items starting at 0x55f488147880
9993
Stack 0x55f488145260: 2 items starting at 0x55f488147880
9992
Stack 0x55f488145260: 1 items starting at 0x55f488147880
9991
Stack 0x55f488145260: 0 items starting at (nil)
9990
```

1.6 Debugging

1. GDB
    - debuggers let us step through programs and observe variables.
    - Compile a program with -g or -ggdb3 with gcc or clang
        - this adds debugging symbols (so you can read it!)
    - tell gdb to use your program
        - gdb ./a.out
    - tell gdb to run your program
        - run
    - tell gbd to print a backtrace when something crashes
        - bt
    - tell gdb to print a variable name
        - p string
    - tell gdb to break at some point
        - b filename:function
        - b filename:line
        - b line

    - b function
- tell gdb to step into code (including into functions)
    - s
- tell gdb to eval the next line (run functions)
    - n
- keep running (continue)
    - c
- print source code (list)
    - l
- remove breakpoint
    - clear
    - clear function
    - clear line
- quit
    - q
- man gdb to get more help

```
hindle1@frail:~/projects/CMPUT201/CMPUT201W20B2-public/week08$ gdb ./bad_realloc
GNU gdb (Ubuntu 8.1-0ubuntu3.2) 8.1.0.20180409-git
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./bad_realloc...done.
(gdb) run
Starting program: /home/hindle1/projects/CMPUT201/CMPUT201W20B2-public/week08/bad_
Stack: 0 items starting at (nil)
```

```
Enter some lines. Press ctrl-d (EOF) to end.
100

Program received signal SIGSEGV, Segmentation fault.
0x00005555555549a1 in push (stack=..., string=0x555555757670 "100\n")
    at ./bad_realloc.c:54
54      stack.elts[stack.size-1] = string;
(gdb) p
The history is empty.
(gdb) bt
#0  0x00005555555549a1 in push (stack=..., string=0x555555757670 "100\n")
    at ./bad_realloc.c:54
#1  0x0000555555554b30 in push_input_lines (stack=...) at ./bad_realloc.c:91
#2  0x0000555555554be0 in main () at ./bad_realloc.c:111
(gdb) p stack
$1 = {size = 0, elts = 0x0}
(gdb) p stack.size
$2 = 0
(gdb) p stack.elts
$3 = (char **) 0x0
(gdb) p string
$4 = 0x555555757670 "100\n"
(gdb) l
49      stack.size = new_size;
50 }
51
52 void push(Stack stack, char * string) {
53      resize(stack, stack.size + 1);
54      stack.elts[stack.size-1] = string;
55      show_stack(stack);
56 }
57
58 char * pop(Stack stack) {
(gdb)
```

Here's a longer example of GDB

```
hindle1@frail:~/projects/CMPUT201/CMPUT201W20B2-public/week08$ gdb ./cards-aoa
GNU gdb (Ubuntu 8.1-0ubuntu3.2) 8.1.0.20180409-git
Copyright (C) 2018 Free Software Foundation, Inc.
```

```
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./cards-aoa...done.
(gdb) b isFlush
Breakpoint 1 at 0x806: file ./cards-aoa.c, line 50.
(gdb) run
Starting program: /home/hindle1/projects/CMPUT201/CMPUT201W20B2-public/week08/carc

Breakpoint 1, isFlush (hand=0x555555757260) at ./cards-aoa.c:50
warning: Source file is more recent than executable.
50     CardSuit suit = hand[0].suit;
(gdb) c
Continuing.

Breakpoint 1, isFlush (hand=0x555555757290) at ./cards-aoa.c:50
50     CardSuit suit = hand[0].suit;
(gdb) c
Continuing.

Breakpoint 1, isFlush (hand=0x5555557572c0) at ./cards-aoa.c:50
50     CardSuit suit = hand[0].suit;
(gdb) c
Continuing.

Breakpoint 1, isFlush (hand=0x5555557572f0) at ./cards-aoa.c:50
50     CardSuit suit = hand[0].suit;
(gdb) p hand
$1 = (PlayingCard *) 0x5555557572f0
(gdb) p hand[0]
$2 = {face = QUEEN, suit = DIAMONDS}
```

```
(gdb) p hand[0].suit
$3 = DIAMONDS
(gdb) s
51     for (int i = 1;  i < HANDSIZE; i++ ) {
(gdb) s
52         if (suit != hand[i].suit) {
(gdb) s
53             return false;
(gdb) s
57 }
(gdb) s
main () at ./cards-aoa.c:88
88     for (int i = 0; i < HANDS; i++) {
(gdb) s
89         if (isFlush(hands[i])) {
(gdb) s

Breakpoint 1, isFlush (hand=0x555555757320) at ./cards-aoa.c:50
50     CardSuit suit = hand[0].suit;
(gdb) s
51     for (int i = 1;  i < HANDSIZE; i++ ) {
(gdb) s
52         if (suit != hand[i].suit) {
(gdb) s
53             return false;
(gdb) s
57 }
(gdb) s
main () at ./cards-aoa.c:88
88     for (int i = 0; i < HANDS; i++) {
(gdb) s
89         if (isFlush(hands[i])) {
(gdb) s

Breakpoint 1, isFlush (hand=0x555555757350) at ./cards-aoa.c:50
50     CardSuit suit = hand[0].suit;
(gdb) n
51     for (int i = 1;  i < HANDSIZE; i++ ) {
(gdb) n
52         if (suit != hand[i].suit) {
```

```
(gdb) n
51     for (int i = 1;  i < HANDSIZE; i++ ) {
(gdb) n
52         if (suit != hand[i].suit) {
(gdb) n
53             return false;
(gdb) n
57 }
(gdb) n
main () at ./cards-aoa.c:88
88     for (int i = 0; i < HANDS; i++) {
(gdb) n
89         if (isFlush(hands[i])) {
(gdb) n

Breakpoint 1, isFlush (hand=0x555555757380) at ./cards-aoa.c:50
50     CardSuit suit = hand[0].suit;
(gdb) clear isFlush
Deleted breakpoint 1
(gdb) c
Continuing.
Flush found at card 228
Suit 3
Flush found at card 291
Suit 2
Flush found at card 846
Suit 1
Flush found at card 886
Suit 2
Flush found at card 892
Suit 0
Flush found at card 1102
Suit 2
Flush found at card 1104
Suit 0
Flush found at card 1437
Suit 0
Flush found at card 1872
Suit 1
Flush found at card 2156
```

```
Suit 2
We found 3857 flushes out of 1000000 hands: 0.003857
[Inferior 1 (process 18051) exited normally]
(gdb) q
```

1.6.1 valgrind

- Valgrind can debug memory issues like
  - unitialized values
  - memory leaks
  - reading/writing free'd memory
  - bad use of the stack (not great)
- valgrind ./yourprogram
- valgrind –tool=memcheck ./yourprogram
- valgrind –tool=exp-sgcheck ./yourprogram
  - for stack checks (not great)
- There's always the manual `https://valgrind.org/docs/manual/manual.html`
- do you want a lot of output?
  - valgrind –leak-check=full –show-leak-kinds=all –track-origins=yes –verbose ./yourprgram

(a) Array Out of Bounds

```
#define _POSIX_C_SOURCE 200809L
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/*
 * This is an example of BAD CODE!
 * Can you use valgrind and gdb
 * to figure out what's wrong with it?
 */
```

```
int main() {
    size_t size;
    printf("How big?\n");
    if (scanf("%zu", &size) != 1) {
        abort();
    }

    int array[size];
    for (size_t idx = 0; idx < size; idx++) {
        array[idx] = 0;
    }

    printf("%d\n", array[100]);
    array[100] += 1;
    printf("%d\n", array[100]);
    return 0;
}
```

```
gcc -std=c99 -O0 -pedantic -Wall -Wextra -ftrapv -ggdb3 -o array_oob ./array_
echo 32 | ./array_oob
echo $?
```

```
How big?
0
1
0
```

```
gcc -std=c99 -O0 -pedantic -Wall -Wextra -ftrapv -ggdb3 -o array_oob ./array_
echo 32 | valgrind --leak-check=full  ./array_oob 2>&1
echo $?
```

```
==20124== Memcheck, a memory error detector
==20124== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==20124== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==20124== Command: ./array_oob
==20124==
==20124== Conditional jump or move depends on uninitialised value(s)
==20124==    at 0x4E988DA: vfprintf (vfprintf.c:1642)
==20124==    by 0x4EA0F25: printf (printf.c:33)
==20124==    by 0x1088AE: main (array_oob.c:27)
==20124==
```

```
==20124== Use of uninitialised value of size 8
==20124==    at 0x4E9486B: _itoa_word (_itoa.c:179)
==20124==    by 0x4E97F0D: vfprintf (vfprintf.c:1642)
==20124==    by 0x4EA0F25: printf (printf.c:33)
==20124==    by 0x1088AE: main (array_oob.c:27)
==20124==
==20124== Conditional jump or move depends on uninitialised value(s)
==20124==    at 0x4E94875: _itoa_word (_itoa.c:179)
==20124==    by 0x4E97F0D: vfprintf (vfprintf.c:1642)
==20124==    by 0x4EA0F25: printf (printf.c:33)
==20124==    by 0x1088AE: main (array_oob.c:27)
==20124==
==20124== Conditional jump or move depends on uninitialised value(s)
==20124==    at 0x4E98014: vfprintf (vfprintf.c:1642)
==20124==    by 0x4EA0F25: printf (printf.c:33)
==20124==    by 0x1088AE: main (array_oob.c:27)
==20124==
==20124== Conditional jump or move depends on uninitialised value(s)
==20124==    at 0x4E98B4C: vfprintf (vfprintf.c:1642)
==20124==    by 0x4EA0F25: printf (printf.c:33)
==20124==    by 0x1088AE: main (array_oob.c:27)
==20124==
==20124== Conditional jump or move depends on uninitialised value(s)
==20124==    at 0x10895E: __addvsi3 (in /home/hindle1/projects/CMPUT201/CMPUT
==20124==    by 0x1088C4: main (array_oob.c:28)
==20124==
==20124== Conditional jump or move depends on uninitialised value(s)
==20124==    at 0x4E988DA: vfprintf (vfprintf.c:1642)
==20124==    by 0x4EA0F25: printf (printf.c:33)
==20124==    by 0x1088ED: main (array_oob.c:29)
==20124==
==20124== Use of uninitialised value of size 8
==20124==    at 0x4E9486B: _itoa_word (_itoa.c:179)
==20124==    by 0x4E97F0D: vfprintf (vfprintf.c:1642)
==20124==    by 0x4EA0F25: printf (printf.c:33)
==20124==    by 0x1088ED: main (array_oob.c:29)
==20124==
==20124== Conditional jump or move depends on uninitialised value(s)
==20124==    at 0x4E94875: _itoa_word (_itoa.c:179)
==20124==    by 0x4E97F0D: vfprintf (vfprintf.c:1642)
```

```
==20124==    by 0x4EA0F25: printf (printf.c:33)
==20124==    by 0x1088ED: main (array_oob.c:29)
==20124==
==20124== Conditional jump or move depends on uninitialised value(s)
==20124==    at 0x4E98014: vfprintf (vfprintf.c:1642)
==20124==    by 0x4EA0F25: printf (printf.c:33)
==20124==    by 0x1088ED: main (array_oob.c:29)
==20124==
==20124== Conditional jump or move depends on uninitialised value(s)
==20124==    at 0x4E98B4C: vfprintf (vfprintf.c:1642)
==20124==    by 0x4EA0F25: printf (printf.c:33)
==20124==    by 0x1088ED: main (array_oob.c:29)
==20124==
How big?
0
1
==20124==
==20124== HEAP SUMMARY:
==20124==     in use at exit: 0 bytes in 0 blocks
==20124==   total heap usage: 2 allocs, 2 frees, 8,192 bytes allocated
==20124==
==20124== All heap blocks were freed -- no leaks are possible
==20124==
==20124== For counts of detected and suppressed errors, rerun with: -v
==20124== Use --track-origins=yes to see where uninitialised values come from
==20124== ERROR SUMMARY: 11 errors from 11 contexts (suppressed: 0 from 0)
0
```

The output is dependent on your input

```
gcc -std=c99 --stack-check -pedantic -Wall -Wextra -ftrapv -g3 -o array_oob ./
echo 32 | valgrind --tool=exp-sgcheck ./array_oob 2>&1
echo $?
```

```
==20317== exp-sgcheck, a stack and global array overrun detector
==20317== NOTE: This is an Experimental-Class Valgrind Tool
==20317== Copyright (C) 2003-2017, and GNU GPL'd, by OpenWorks Ltd et al.
==20317== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==20317== Command: ./array_oob
==20317==
--20317-- warning: evaluate_Dwarf3_Expr: unhandled DW_OP_ 0x93
```

```
--20317-- warning: evaluate_Dwarf3_Expr: unhandled DW_OP_ 0x93
--20317-- warning: evaluate_Dwarf3_Expr: unhandled DW_OP_ 0x93
--20317-- warning: evaluate_Dwarf3_Expr: unhandled DW_OP_ 0x93
--20317-- warning: evaluate_Dwarf3_Expr: unhandled DW_OP_ 0x93
--20317-- warning: evaluate_Dwarf3_Expr: unhandled DW_OP_ 0x93
--20317-- warning: evaluate_Dwarf3_Expr: unhandled DW_OP_ 0x93
--20317-- warning: evaluate_Dwarf3_Expr: unhandled DW_OP_ 0x93
--20317-- warning: evaluate_Dwarf3_Expr: unhandled DW_OP_ 0x93
--20317-- warning: evaluate_Dwarf3_Expr: unhandled DW_OP_ 0x93
--20317-- warning: evaluate_Dwarf3_Expr: unhandled DW_OP_ 0x93
--20317-- warning: evaluate_Dwarf3_Expr: unhandled DW_OP_ 0x93
--20317-- warning: evaluate_Dwarf3_Expr: unhandled DW_OP_ 0x93
--20317-- warning: evaluate_Dwarf3_Expr: unhandled DW_OP_ 0x93
--20317-- warning: evaluate_Dwarf3_Expr: unhandled DW_OP_ 0x93
--20317-- warning: evaluate_Dwarf3_Expr: unhandled DW_OP_ 0x93
How big?
0
1
==20317==
==20317== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 2 from 2)
0
```

```
gcc -std=c99 -pedantic -Wall -Wextra -ftrapv -ggdb3 -o array_oob ./array_oob.c
gnome-terminal -- gdb ./array_oob
```

(b) Array unitialized

```
#define _POSIX_C_SOURCE 200809L
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/*
 * This is an example of BAD CODE!
 * Can you use valgrind and gdb
 * to figure out what's wrong with it?
 */


int main() {
```

```
    size_t size;
    printf("How big?\n");
    if (scanf("%zu", &size) != 1) {
        abort();
    }

    int array[size];
    for (size_t idx = 0; idx < size; idx++) {
        printf("%d\n", array[idx]);
    }
    return 0;
}
```

```
gcc -std=c99 -O0 -pedantic -Wall -Wextra -ftrapv -ggdb3 -o array_uninit ./arra
echo 10 | ./array_uninit
echo $?
```

```
How big?
-782409112
32764
0
0
-782673888
32764
-782673984
32764
0
0
0
```

```
gcc -std=c99 -O0 -pedantic -Wall -Wextra -ftrapv -ggdb3 -o array_uninit ./arra
echo 5 | valgrind --leak-check=full  ./array_uninit 2>&1
echo $?
```

```
==16458== Memcheck, a memory error detector
==16458== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==16458== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==16458== Command: ./array_uninit
==16458==
==16458== Conditional jump or move depends on uninitialised value(s)
==16458==    at 0x4E988DA: vfprintf (vfprintf.c:1642)
```

```
==16458==    by 0x4EA0F25: printf (printf.c:33)
==16458==    by 0x108891: main (array_uninit.c:24)
==16458==
==16458== Use of uninitialised value of size 8
==16458==    at 0x4E9486B: _itoa_word (_itoa.c:179)
==16458==    by 0x4E97F0D: vfprintf (vfprintf.c:1642)
==16458==    by 0x4EA0F25: printf (printf.c:33)
==16458==    by 0x108891: main (array_uninit.c:24)
==16458==
==16458== Conditional jump or move depends on uninitialised value(s)
==16458==    at 0x4E94875: _itoa_word (_itoa.c:179)
==16458==    by 0x4E97F0D: vfprintf (vfprintf.c:1642)
==16458==    by 0x4EA0F25: printf (printf.c:33)
==16458==    by 0x108891: main (array_uninit.c:24)
==16458==
==16458== Conditional jump or move depends on uninitialised value(s)
==16458==    at 0x4E98014: vfprintf (vfprintf.c:1642)
==16458==    by 0x4EA0F25: printf (printf.c:33)
==16458==    by 0x108891: main (array_uninit.c:24)
==16458==
==16458== Conditional jump or move depends on uninitialised value(s)
==16458==    at 0x4E98B4C: vfprintf (vfprintf.c:1642)
==16458==    by 0x4EA0F25: printf (printf.c:33)
==16458==    by 0x108891: main (array_uninit.c:24)
==16458==
How big?
-16776224
31
-16776320
31
0
==16458==
==16458== HEAP SUMMARY:
==16458==     in use at exit: 0 bytes in 0 blocks
==16458==   total heap usage: 2 allocs, 2 frees, 8,192 bytes allocated
==16458==
==16458== All heap blocks were freed -- no leaks are possible
==16458==
==16458== For counts of detected and suppressed errors, rerun with: -v
==16458== Use --track-origins=yes to see where uninitialised values come from
```

```
==16458== ERROR SUMMARY: 57 errors from 5 contexts (suppressed: 0 from 0)
0
```

Yeah valgrind did not like that. It complained about uninitiliazed values.

1.6.2 More bad code

These files are debugging examples where you should practice valgrind and gcc.

```
# look a bash for loop!
echo Compiling!
for file in ./array_oob.c ./array_uninit.c ./bad_realloc.c ./bad_str.c ./double_fr
do
gcc -std=c99 -pedantic -Wall -Wextra -ftrapv -ggdb3 -o ‘basename -s .c $file‘ $fil
done

Compiling!
```

(a) Files `./array_oob.c ./array_uninit.c ./bad_realloc.c ./bad_str.c ./double_free.c ./huge_array.c ./infinite_recursion.c ./malloc.c ./malloc_oob.c ./malloc_uninit.c ./segv.c ./simple_uninit.c ./stack.c ./stack_limit.c ./use_after_free.c`

(b) $./\text{array}_{\text{oob}}.\text{c}$ `./array_oob.c`

```
gcc -std=c99 -pedantic -Wall -Wextra -ftrapv -ggdb3 -o array_oob ./array_oob.c
echo 33 | valgrind ./array_oob 2>&1
```

```
==27387== Memcheck, a memory error detector
==27387== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==27387== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==27387== Command: ./array_oob
==27387==
==27387== Conditional jump or move depends on uninitialised value(s)
==27387==    at 0x4E988DA: vfprintf (vfprintf.c:1642)
==27387==    by 0x4EA0F25: printf (printf.c:33)
==27387==    by 0x1088AE: main (array_oob.c:27)
==27387==
```

```
==27387== Use of uninitialised value of size 8
==27387==    at 0x4E9486B: _itoa_word (_itoa.c:179)
==27387==    by 0x4E97F0D: vfprintf (vfprintf.c:1642)
==27387==    by 0x4EA0F25: printf (printf.c:33)
==27387==    by 0x1088AE: main (array_oob.c:27)
==27387==
==27387== Conditional jump or move depends on uninitialised value(s)
==27387==    at 0x4E94875: _itoa_word (_itoa.c:179)
==27387==    by 0x4E97F0D: vfprintf (vfprintf.c:1642)
==27387==    by 0x4EA0F25: printf (printf.c:33)
==27387==    by 0x1088AE: main (array_oob.c:27)
==27387==
==27387== Conditional jump or move depends on uninitialised value(s)
==27387==    at 0x4E98014: vfprintf (vfprintf.c:1642)
==27387==    by 0x4EA0F25: printf (printf.c:33)
==27387==    by 0x1088AE: main (array_oob.c:27)
==27387==
==27387== Conditional jump or move depends on uninitialised value(s)
==27387==    at 0x4E98B4C: vfprintf (vfprintf.c:1642)
==27387==    by 0x4EA0F25: printf (printf.c:33)
==27387==    by 0x1088AE: main (array_oob.c:27)
==27387==
==27387== Conditional jump or move depends on uninitialised value(s)
==27387==    at 0x10895E: __addvsi3 (in /home/hindle1/projects/CMPUT201/CMPUT
==27387==    by 0x1088C4: main (array_oob.c:28)
==27387==
==27387== Conditional jump or move depends on uninitialised value(s)
==27387==    at 0x4E988DA: vfprintf (vfprintf.c:1642)
==27387==    by 0x4EA0F25: printf (printf.c:33)
==27387==    by 0x1088ED: main (array_oob.c:29)
==27387==
==27387== Use of uninitialised value of size 8
==27387==    at 0x4E9486B: _itoa_word (_itoa.c:179)
==27387==    by 0x4E97F0D: vfprintf (vfprintf.c:1642)
==27387==    by 0x4EA0F25: printf (printf.c:33)
==27387==    by 0x1088ED: main (array_oob.c:29)
==27387==
==27387== Conditional jump or move depends on uninitialised value(s)
==27387==    at 0x4E94875: _itoa_word (_itoa.c:179)
==27387==    by 0x4E97F0D: vfprintf (vfprintf.c:1642)
```

```
==27387==    by 0x4EA0F25: printf (printf.c:33)
==27387==    by 0x1088ED: main (array_oob.c:29)
==27387==
==27387== Conditional jump or move depends on uninitialised value(s)
==27387==    at 0x4E98014: vfprintf (vfprintf.c:1642)
==27387==    by 0x4EA0F25: printf (printf.c:33)
==27387==    by 0x1088ED: main (array_oob.c:29)
==27387==
==27387== Conditional jump or move depends on uninitialised value(s)
==27387==    at 0x4E98B4C: vfprintf (vfprintf.c:1642)
==27387==    by 0x4EA0F25: printf (printf.c:33)
==27387==    by 0x1088ED: main (array_oob.c:29)
==27387==
How big?
0
1
==27387==
==27387== HEAP SUMMARY:
==27387==     in use at exit: 0 bytes in 0 blocks
==27387==   total heap usage: 2 allocs, 2 frees, 8,192 bytes allocated
==27387==
==27387== All heap blocks were freed -- no leaks are possible
==27387==
==27387== For counts of detected and suppressed errors, rerun with: -v
==27387== Use --track-origins=yes to see where uninitialised values come from
==27387== ERROR SUMMARY: 11 errors from 11 contexts (suppressed: 0 from 0)
```

Run GDB

```
gcc -std=c99 -pedantic -Wall -Wextra -ftrapv -ggdb3 -o array_oob ./array_oob.c
gnome-terminal -- gdb ./array_oob
```

(c) ./$\text{array}_{\text{uninit}}$.c `./array_uninit.c`

```
gcc -std=c99 -pedantic -Wall -Wextra -ftrapv -ggdb3 -o array_uninit ./array_uninit.c
echo 33 | valgrind ./array_uninit 2>&1
```

```
==27379== Memcheck, a memory error detector
==27379== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==27379== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==27379== Command: ./array_uninit
==27379==
```

```
==27379== Conditional jump or move depends on uninitialised value(s)
==27379==    at 0x4E988DA: vfprintf (vfprintf.c:1642)
==27379==    by 0x4EA0F25: printf (printf.c:33)
==27379==    by 0x108891: main (array_uninit.c:24)
==27379==
==27379== Use of uninitialised value of size 8
==27379==    at 0x4E9486B: _itoa_word (_itoa.c:179)
==27379==    by 0x4E97F0D: vfprintf (vfprintf.c:1642)
==27379==    by 0x4EA0F25: printf (printf.c:33)
==27379==    by 0x108891: main (array_uninit.c:24)
==27379==
==27379== Conditional jump or move depends on uninitialised value(s)
==27379==    at 0x4E94875: _itoa_word (_itoa.c:179)
==27379==    by 0x4E97F0D: vfprintf (vfprintf.c:1642)
==27379==    by 0x4EA0F25: printf (printf.c:33)
==27379==    by 0x108891: main (array_uninit.c:24)
==27379==
==27379== Conditional jump or move depends on uninitialised value(s)
==27379==    at 0x4E98014: vfprintf (vfprintf.c:1642)
==27379==    by 0x4EA0F25: printf (printf.c:33)
==27379==    by 0x108891: main (array_uninit.c:24)
==27379==
==27379== Conditional jump or move depends on uninitialised value(s)
==27379==    at 0x4E98B4C: vfprintf (vfprintf.c:1642)
==27379==    by 0x4EA0F25: printf (printf.c:33)
==27379==    by 0x108891: main (array_uninit.c:24)
==27379==
How big?
8
0
82606643
0
0
0
0
0
0
0
8
0
```

```
86148960
0
1083828
0
86131360
0
0
0
0
0
82561679
0
0
0
0
0
-16776224
31
-16776320
31
0
==27379==
==27379== HEAP SUMMARY:
==27379==     in use at exit: 0 bytes in 0 blocks
==27379==   total heap usage: 2 allocs, 2 frees, 8,192 bytes allocated
==27379==
==27379== All heap blocks were freed -- no leaks are possible
==27379==
==27379== For counts of detected and suppressed errors, rerun with: -v
==27379== Use --track-origins=yes to see where uninitialised values come from
==27379== ERROR SUMMARY: 265 errors from 5 contexts (suppressed: 0 from 0)
```

Run GDB

```
gcc -std=c99 -pedantic -Wall -Wextra -ftrapv -ggdb3 -o array_uninit ./array_u
gnome-terminal -- gdb ./array_uninit
```

(d) ./$\text{bad}_{\text{realloc}}$.c `./bad_realloc.c`

```
gcc -std=c99 -pedantic -Wall -Wextra -ftrapv -ggdb3 -o bad_realloc ./bad_reall
echo 33 | valgrind ./bad_realloc 2>&1
```

Run GDB

```
gcc -std=c99 -pedantic -Wall -Wextra -ftrapv -ggdb3 -o bad_realloc ./bad_reall
gnome-terminal -- gdb ./bad_realloc
```

(e) ./bad_{str}.c `./bad_str.c`

```
gcc -std=c99 -pedantic -Wall -Wextra -ftrapv -ggdb3 -o bad_str ./bad_str.c
echo 33 | valgrind ./bad_str 2>&1
```

```
==27350== Memcheck, a memory error detector
==27350== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==27350== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==27350== Command: ./bad_str
==27350==
Enter a message:
You entered: 33
==27350==
==27350== HEAP SUMMARY:
==27350==     in use at exit: 5 bytes in 1 blocks
==27350==   total heap usage: 3 allocs, 2 frees, 8,197 bytes allocated
==27350==
==27350== LEAK SUMMARY:
==27350==    definitely lost: 5 bytes in 1 blocks
==27350==    indirectly lost: 0 bytes in 0 blocks
==27350==      possibly lost: 0 bytes in 0 blocks
==27350==    still reachable: 0 bytes in 0 blocks
==27350==         suppressed: 0 bytes in 0 blocks
==27350== Rerun with --leak-check=full to see details of leaked memory
==27350==
==27350== For counts of detected and suppressed errors, rerun with: -v
==27350== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Run GDB

```
gcc -std=c99 -pedantic -Wall -Wextra -ftrapv -ggdb3 -o bad_str ./bad_str.c
gnome-terminal -- gdb ./bad_str
```

(f) ./$\text{double}_{\text{free}}$.c `./double_free.c`

```
gcc -std=c99 -pedantic -Wall -Wextra -ftrapv -ggdb3 -o double_free ./double_fi
echo 33 | valgrind ./double_free 2>&1
```

```
==27337== Memcheck, a memory error detector
==27337== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==27337== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==27337== Command: ./double_free
==27337==
==27337== Invalid free() / delete / delete[] / realloc()
==27337==    at 0x4C30D3B: free (in /usr/lib/valgrind/vgpreload_memcheck-amd64
==27337==    by 0x108904: main (double_free.c:27)
==27337==  Address 0x522f0c0 is 0 bytes inside a block of size 132 free'd
==27337==    at 0x4C30D3B: free (in /usr/lib/valgrind/vgpreload_memcheck-amd64
==27337==    by 0x1088F8: main (double_free.c:26)
==27337==  Block was alloc'd at
==27337==    at 0x4C2FB0F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd
==27337==    by 0x10888E: main (double_free.c:21)
==27337==
How big?
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
```

```
24
25
26
27
28
29
30
31
32
==27337==
==27337== HEAP SUMMARY:
==27337==     in use at exit: 0 bytes in 0 blocks
==27337==   total heap usage: 3 allocs, 4 frees, 8,324 bytes allocated
==27337==
==27337== All heap blocks were freed -- no leaks are possible
==27337==
==27337== For counts of detected and suppressed errors, rerun with: -v
==27337== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Run GDB

```
gcc -std=c99 -pedantic -Wall -Wextra -ftrapv -ggdb3 -o double_free ./double_fi
gnome-terminal -- gdb ./double_free
```

(g) ./huge$_{\text{array}}$.c ./huge_array.c

```
gcc -std=c99 -pedantic -Wall -Wextra -ftrapv -ggdb3 -o huge_array ./huge_array
echo 33 | valgrind ./huge_array 2>&1
```

```
==27323== Memcheck, a memory error detector
==27323== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==27323== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==27323== Command: ./huge_array
==27323==
==27323== Warning: client switching stacks?  SP change: 0x1fff0003e0 --> 0x1fi
==27323==          to suppress, use: --max-stackframe=8388624 or greater
==27323== Invalid write of size 8
==27323==    at 0x108728: main (huge_array.c:14)
==27323==  Address 0x1ffe8003c8 is on thread 1's stack
==27323==  in frame #0, created by main (huge_array.c:13)
==27323==
==27323== Invalid write of size 8
```

```
==27323==    at 0x4C36657: memset (in /usr/lib/valgrind/vgpreload_memcheck-amc
==27323==    by 0x10872C: main (huge_array.c:14)
==27323==  Address 0x1ffe8003d0 is on thread 1's stack
==27323==  in frame #1, created by main (huge_array.c:13)
==27323==
==27323== Invalid write of size 8
==27323==    at 0x4C3665A: memset (in /usr/lib/valgrind/vgpreload_memcheck-amc
==27323==    by 0x10872C: main (huge_array.c:14)
==27323==  Address 0x1ffe8003d8 is on thread 1's stack
==27323==  in frame #1, created by main (huge_array.c:13)
==27323==
==27323== Invalid write of size 8
==27323==    at 0x4C3665E: memset (in /usr/lib/valgrind/vgpreload_memcheck-amc
==27323==    by 0x10872C: main (huge_array.c:14)
==27323==  Address 0x1ffe8003e0 is on thread 1's stack
==27323==  in frame #1, created by main (huge_array.c:13)
==27323==
==27323== Invalid write of size 8
==27323==    at 0x4C36662: memset (in /usr/lib/valgrind/vgpreload_memcheck-amc
==27323==    by 0x10872C: main (huge_array.c:14)
==27323==  Address 0x1ffe8003e8 is on thread 1's stack
==27323==  in frame #1, created by main (huge_array.c:13)
==27323==
==27323== Invalid read of size 8
==27323==    at 0x4C366D5: memset (in /usr/lib/valgrind/vgpreload_memcheck-amc
==27323==    by 0x10872C: main (huge_array.c:14)
==27323==  Address 0x1ffe8003c8 is on thread 1's stack
==27323==  in frame #0, created by memset (???:)
==27323==
==27323== Invalid read of size 4
==27323==    at 0x10872D: main (huge_array.c:15)
==27323==  Address 0x1ffe8003d0 is on thread 1's stack
==27323==  in frame #0, created by main (huge_array.c:13)
==27323==
==27323== Warning: client switching stacks?  SP change: 0x1ffe8003d0 --> 0x1f
==27323==          to suppress, use: --max-stackframe=8388624 or greater
0
==27323==
==27323== HEAP SUMMARY:
==27323==     in use at exit: 0 bytes in 0 blocks
```

```
==27323==   total heap usage: 1 allocs, 1 frees, 4,096 bytes allocated
==27323==
==27323== All heap blocks were freed -- no leaks are possible
==27323==
==27323== For counts of detected and suppressed errors, rerun with: -v
==27323== ERROR SUMMARY: 1048565 errors from 7 contexts (suppressed: 0 from 0)
```

Run GDB

```
gcc -std=c99 -pedantic -Wall -Wextra -ftrapv -ggdb3 -o huge_array ./huge_array
gnome-terminal -- gdb ./huge_array
```

(h) ./malloc$_{\text{oob}}$.c `./malloc_oob.c`

```
gcc -std=c99 -pedantic -Wall -Wextra -ftrapv -ggdb3 -o malloc_oob ./malloc_oob
echo 33 | valgrind ./malloc_oob 2>&1
```

```
==27315== Memcheck, a memory error detector
==27315== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==27315== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==27315== Command: ./malloc_oob
==27315==
==27315== Invalid read of size 4
==27315==    at 0x1088CF: main (malloc_oob.c:26)
==27315==  Address 0x522f250 is 192 bytes inside an unallocated block of size
==27315==
How big?
0
==27315==
==27315== HEAP SUMMARY:
==27315==     in use at exit: 0 bytes in 0 blocks
==27315==   total heap usage: 3 allocs, 3 frees, 8,324 bytes allocated
==27315==
==27315== All heap blocks were freed -- no leaks are possible
==27315==
==27315== For counts of detected and suppressed errors, rerun with: -v
==27315== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Run GDB

```
gcc -std=c99 -pedantic -Wall -Wextra -ftrapv -ggdb3 -o malloc_oob ./malloc_oob
gnome-terminal -- gdb ./malloc_oob
```

(i) ./malloc$_{\text{uninit}}$.c ./malloc_uninit.c

```
gcc -std=c99 -pedantic -Wall -Wextra -ftrapv -ggdb3 -o malloc_uninit ./malloc
echo 33 | valgrind ./malloc_uninit 2>&1
```

```
==27303== Memcheck, a memory error detector
==27303== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==27303== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==27303== Command: ./malloc_uninit
==27303==
==27303== Conditional jump or move depends on uninitialised value(s)
==27303==    at 0x4E988DA: vfprintf (vfprintf.c:1642)
==27303==    by 0x4EA0F25: printf (printf.c:33)
==27303==    by 0x108884: main (malloc_uninit.c:25)
==27303==
==27303== Use of uninitialised value of size 8
==27303==    at 0x4E9486B: _itoa_word (_itoa.c:179)
==27303==    by 0x4E97F0D: vfprintf (vfprintf.c:1642)
==27303==    by 0x4EA0F25: printf (printf.c:33)
==27303==    by 0x108884: main (malloc_uninit.c:25)
==27303==
==27303== Conditional jump or move depends on uninitialised value(s)
==27303==    at 0x4E94875: _itoa_word (_itoa.c:179)
==27303==    by 0x4E97F0D: vfprintf (vfprintf.c:1642)
==27303==    by 0x4EA0F25: printf (printf.c:33)
==27303==    by 0x108884: main (malloc_uninit.c:25)
==27303==
==27303== Conditional jump or move depends on uninitialised value(s)
==27303==    at 0x4E98014: vfprintf (vfprintf.c:1642)
==27303==    by 0x4EA0F25: printf (printf.c:33)
==27303==    by 0x108884: main (malloc_uninit.c:25)
==27303==
==27303== Conditional jump or move depends on uninitialised value(s)
==27303==    at 0x4E98B4C: vfprintf (vfprintf.c:1642)
==27303==    by 0x4EA0F25: printf (printf.c:33)
==27303==    by 0x108884: main (malloc_uninit.c:25)
==27303==
How big?
0
0
0
```

```
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
==27303==
==27303== HEAP SUMMARY:
==27303==     in use at exit: 132 bytes in 1 blocks
==27303==   total heap usage: 3 allocs, 2 frees, 8,324 bytes allocated
==27303==
==27303== LEAK SUMMARY:
==27303==    definitely lost: 132 bytes in 1 blocks
==27303==    indirectly lost: 0 bytes in 0 blocks
==27303==      possibly lost: 0 bytes in 0 blocks
==27303==    still reachable: 0 bytes in 0 blocks
```

```
==27303==          suppressed: 0 bytes in 0 blocks
==27303== Rerun with --leak-check=full to see details of leaked memory
==27303==
==27303== For counts of detected and suppressed errors, rerun with: -v
==27303== Use --track-origins=yes to see where uninitialised values come from
==27303== ERROR SUMMARY: 165 errors from 5 contexts (suppressed: 0 from 0)
```

Run GDB

```
gcc -std=c99 -pedantic -Wall -Wextra -ftrapv -ggdb3 -o malloc_uninit ./malloc_
gnome-terminal -- gdb ./malloc_uninit
```

(j) ./segv.c `./segv.c`

```
gcc -std=c99 -pedantic -Wall -Wextra -ftrapv -ggdb3 -o segv ./segv.c
echo 33 | valgrind ./segv 2>&1
```

```
==27291== Memcheck, a memory error detector
==27291== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==27291== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==27291== Command: ./segv
==27291==
==27291== Invalid read of size 4
==27291==    at 0x1088CF: main (segv.c:26)
==27291==  Address 0x55ff9c0 is 3,999,792 bytes inside an unallocated block of
==27291==
How big?
0
==27291==
==27291== HEAP SUMMARY:
==27291==     in use at exit: 0 bytes in 0 blocks
==27291==   total heap usage: 3 allocs, 3 frees, 8,324 bytes allocated
==27291==
==27291== All heap blocks were freed -- no leaks are possible
==27291==
==27291== For counts of detected and suppressed errors, rerun with: -v
==27291== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Run GDB

```
gcc -std=c99 -pedantic -Wall -Wextra -ftrapv -ggdb3 -o segv ./segv.c
gnome-terminal -- gdb ./segv
```

(k) ./simple$_{\text{uninit}}$.c ./simple_uninit.c

```
gcc -std=c99 -pedantic -Wall -Wextra -ftrapv -ggdb3 -o simple_uninit ./simple
echo 33 | valgrind ./simple_uninit 2>&1
```

```
==27279== Memcheck, a memory error detector
==27279== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==27279== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==27279== Command: ./simple_uninit
==27279==
Enter an int:
33
==27279==
==27279== HEAP SUMMARY:
==27279==     in use at exit: 0 bytes in 0 blocks
==27279==   total heap usage: 2 allocs, 2 frees, 8,192 bytes allocated
==27279==
==27279== All heap blocks were freed -- no leaks are possible
==27279==
==27279== For counts of detected and suppressed errors, rerun with: -v
==27279== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Run GDB

```
gcc -std=c99 -pedantic -Wall -Wextra -ftrapv -ggdb3 -o simple_uninit ./simple
gnome-terminal -- gdb ./simple_uninit
```

(l) ./stack.c ./stack.c

```
gcc -std=c99 -pedantic -Wall -Wextra -ftrapv -ggdb3 -o stack ./stack.c
echo 33 | valgrind ./stack 2>&1
```

```
==27253== Memcheck, a memory error detector
==27253== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==27253== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==27253== Command: ./stack
==27253==
Stack 0x522d040: 0 items starting at (nil)
Enter some lines. Press ctrl-d (EOF) to end.
Stack 0x522d040: 1 items starting at 0x522f1d0
Stack 0x522d040: 0 items starting at (nil)
33
```

```
==27253==
==27253== HEAP SUMMARY:
==27253==     in use at exit: 0 bytes in 1 blocks
==27253==   total heap usage: 7 allocs, 6 frees, 8,456 bytes allocated
==27253==
==27253== LEAK SUMMARY:
==27253==    definitely lost: 0 bytes in 1 blocks
==27253==    indirectly lost: 0 bytes in 0 blocks
==27253==      possibly lost: 0 bytes in 0 blocks
==27253==    still reachable: 0 bytes in 0 blocks
==27253==         suppressed: 0 bytes in 0 blocks
==27253== Rerun with --leak-check=full to see details of leaked memory
==27253==
==27253== For counts of detected and suppressed errors, rerun with: -v
==27253== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Run GDB

```
gcc -std=c99 -pedantic -Wall -Wextra -ftrapv -ggdb3 -o stack ./stack.c
gnome-terminal -- gdb ./stack
```

(m) ./$\text{stack}_{\text{limit}}$.c `./stack_limit.c`

```
gcc -std=c99 -pedantic -Wall -Wextra -ftrapv -ggdb3 -o stack_limit ./stack_lim
echo 33 | valgrind ./stack_limit 2>&1
```

Run GDB

```
gcc -std=c99 -pedantic -Wall -Wextra -ftrapv -ggdb3 -o stack_limit ./stack_lim
gnome-terminal -- gdb ./stack_limit
```

(n) ./$\text{use}_{\text{afterfree}}$.c `./use_after_free.c`

```
gcc -std=c99 -pedantic -Wall -Wextra -ftrapv -ggdb3 -o use_after_free ./use_af
echo 33 | valgrind ./use_after_free 2>&1
```

Run GDB

```
gcc -std=c99 -pedantic -Wall -Wextra -ftrapv -ggdb3 -o use_after_free ./use_af
gnome-terminal -- gdb ./use_after_free
```

i. Generator (ignore)

```
# look a bash for loop!
for file in ./array_oob.c ./array_uninit.c ./bad_realloc.c ./bad_str.c ./d
#for file in ./array_oob.c ./array_uninit.c ./bad_realloc.c
```

```
do
exe=‘basename -s .c $file‘
echo
echo \*\*\*\* $file
echo file:$file
echo
echo \#+BEGIN_SRC sh
echo gcc -std=c99 -pedantic -Wall -Wextra -ftrapv -ggdb3 -o $exe $file
echo echo 33 \| valgrind ./$exe 2\>\&1
echo \#+END_SRC
echo
echo Run GDB
echo \#+BEGIN_SRC sh
echo gcc -std=c99 -pedantic -Wall -Wextra -ftrapv -ggdb3 -o $exe $file
echo gnome-terminal -- gdb ./$exe
echo \#+END_SRC
done
```