ESE5320 Homework 1

Zhengyang "Rico" Zhuang

September 5, 2025

Answers

1. **GDB Tutorial**

   1) To add a breakpoint, use `break <FUNCTION/LINE NUMBER>`.
   2) To delete a breakpoint, first do `"info breakpoint"` to find breakpoint index, then `"delete <INDEX>"`.
   3) To To inspect a variable's value, use `"print <VAR_NAME>"`.
   4) To step through execution without a breakpoint: `next` or `step`

2. **C**

   1)
```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

int main(void) {
    int top = 20;
    int *p_top = &top;
    int *h50 = malloc(sizeof *h50); *h50 = 50;
    int *h5  = malloc(sizeof *h5 ); *h5  = 5;
    int *h6  = malloc(sizeof *h6 ); *h6  = 6;
    int *h7  = malloc(sizeof *h7 ); *h7  = 7;

    int *p_h50 = h50;
    int *p_h5  = h5;
```

```
    printf("Stack:\n");
    printf("%11p  %14d\n", (void*)&top, top);
    printf("%11p  %14p\n", (void*)&p_top, (void*)p_top);
    printf("%11p  %14p\n", (void*)&p_h50, (void*)p_h50);
    printf("%11p  %14p\n", (void*)&p_h5,  (void*)p_h5);

    printf("\nHeap:\n");
    printf("h50: %p = %d\n", (void*)h50, *h50);
    printf(" h5: %p = %d\n", (void*)h5,  *h5 );
    printf(" h6: %p = %d\n", (void*)h6,  *h6 );
    printf(" h7: %p = %d\n", (void*)h7,  *h7 );

    free(h50); free(h5); free(h6); free(h7);
    return 0;
}
```

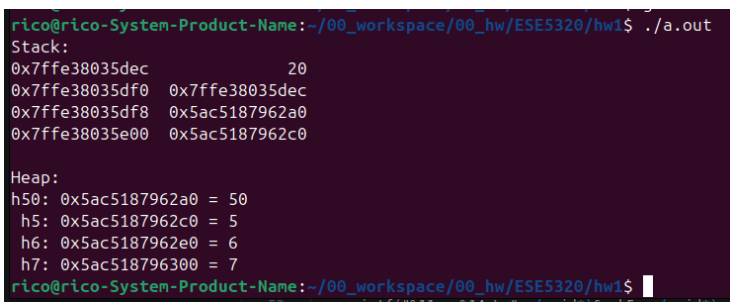

Figure 1: Output from C code above

2)
```
#include <stdio.h>
int main(void) {
    int a[2][4] = { {10, 20, 30, 40}, {50, 60, 70, 80} };
    int *row[2] = { a[0], a[1] };
    for (int i = 0; i < 2; ++i) {
        for (int j = 0; j < 4; ++j) {
            printf("row[%d][%d] = %d\n", i, j, row[i][j]);
        }
    }
    return 0;
}
```

```
row[0][0] = 10
row[0][1] = 20
row[0][2] = 30
row[0][3] = 40
row[1][0] = 50
row[1][1] = 60
row[1][2] = 70
row[1][3] = 80
```

```c
#include <stdio.h>

int main(void) {
    int a[2][4] = { {10, 20, 30, 40}, {50, 60, 70, 80} };
    int *row0 = a[0];
    int *row1 = a[1];
    int *rows[2] = { row0, row1 };
    int **pp = rows;
    for (int i = 0; i < 2; ++i) {
        for (int j = 0; j < 4; ++j) {
            printf("pp[%d][%d] = %d\n", i, j, pp[i][j]);
        }
    }
    return 0;
}
```

```
pp[0][0] = 10
pp[0][1] = 20
pp[0][2] = 30
pp[0][3] = 40
pp[1][0] = 50
pp[1][1] = 60
pp[1][2] = 70
pp[1][3] = 80
```

3) `&x[2].d[0]->b`

4) `#include <stdio.h>`

```
int main(void) {
    int a[2][4] = { {10, 20, 30, 40}, {50, 60, 70, 80} };
    int *row0 = a[0];
    int *row1 = a[1];
    int *rows[2] = { row0, row1 };
    int **pp = rows;
    for (int i = 0; i < 2; ++i) {
        for (int j = 0; j < 4; ++j) {
            printf("pp[%d][%d] = %d\n", i, j, pp[i][j]);
        }
    }
    return 0;
}
```

```
Byte  0: 0x1F
Byte  1: 0x85
Byte  2: 0xEB
Byte  3: 0x51
Byte  4: 0xB8
Byte  5: 0x1E
Byte  6: 0x09
Byte  7: 0x40
Byte  8: 0xAE
Byte  9: 0x47
Byte 10: 0xE1
Byte 11: 0x7A
Byte 12: 0x14
Byte 13: 0xAE
Byte 14: 0x05
Byte 15: 0x40
```

5)
```
#include <stdio.h>
#include <stdlib.h>

void temp(int i) {
    int a[2];
    int b[3];
    int *c;
    int *d;
```

```
    c = (int *)malloc(sizeof(int) * 4);
    d = (int *)malloc(sizeof(int) * 5);

    printf("a (stack, 2 ints) : %p\n", (void*)a);
    printf("b (stack, 3 ints) : %p\n", (void*)b);
    printf("c (heap, 4 ints)  : %p\n", (void*)c);
    printf("d (heap, 5 ints)  : %p\n", (void*)d);

    free(c);
    free(d);

    return;
}

int main(void) {
    temp(0);
    return 0;
}
```

```
a (stack, 2 ints) : 0x7ffc6eff5c94
b (stack, 3 ints) : 0x7ffc6eff5c9c
c (heap, 4 ints)  : 0x5bac942112a0
d (heap, 5 ints)  : 0x5bac942112c0
```

6) c[0] becomes 13
   invaild array indexing is detected and the program stops
7) char and unsigned char sums differ because unsigned char use all 8 bits for numbers but signed char has to use a bit for the sign, so they overflow at different sums

   char and unsigned char sums differ from their 'intsum' because intsum can use 32bits but chars can only use 8 so they overflow after 255.
8) Preprocessor: Handles stuff like include and define before compiling
   compiler: Translates the preprocessed C code into assembly or

machine code
Linker: Combines object files and libraries into executable

9) add the include path to the makefile -I
copy the headerfile to local directory
check name and path to make sure that they are correct

10) object file missing
function not defined
function signature does not match

3. **Debug an Application**

1) Done
2) Done
3) Done
4)
   1) On line 5 I changed "`while (*s) s++;`" to "`while (*s++) l++;`"
   2) Makefile:

```
release:
 gcc -Wall -o program program.c

debug:
 gcc -g -Wall -o program program.c
```

   3) The secret message is: Well Done!!
   4) while loop not executed at all, must be buggy len(). Steppting through len, l doesn't change at all. Look back at the code and noticed that s is incremented in while loop but l is not.
5) the -g flag of gcc adds debug symbols that maps addresses back to source files, and is read by gdb