5	Feb 14	Lecture	Value Tagging	Cobra
	Feb 16	Lecture	Runtime Errors	
6	Feb 21	Lecture	Functions	Diamondback
	Feb 23	Lecture Take-Home Test 1 (assigned)	Compile-time Errors	
7	Feb 28	Lecture Take-Home Test 1 (due)	System Calls	
	Mar 02	Lecture	Register Allocation	
	Mar 07	**Spring Break**		
	Mar 09			
8	Mar 14		Heap Allocation and Pairs	Egg-Eater
	Mar 16			
9	Mar 21		First-Class Functions and Closures	Foxsnake
	Mar 23	CR/NC/W Deadline (Mar 24)		
10	Mar 28			
	Mar 30		Garbage Collection	
11	Apr 04		Mark/Compact	Garbage Snake
	Apr 06	Take-Home Test 2 (assigned)	Memory Management Strategies	
12	Apr 11	Take-Home Test 2 (due)		
	Apr 13		Tail Call Optimization	
13	Apr 18			Hoop Snake
	Apr 20		Parsing	
14	Apr 25			
	Apr 27			
	May 09	**Take-Home Test 3**		

Xournal

File Edit View Page Tools Options Help

Sans 12

Page 1 of 1 Layer: Layer 1

Terminal Xournal CS75: Schedule - Mozilla...

T: 30 °F A: 17 °F H: 88 % 11:22 93%

Xournal

File Edit View Page Tools Options Help

Sans 12

Page 1 of 1 Layer: Layer 1

Terminal Xournal CS75: Schedule - Mozilla...

T: 30 °F A: 17 °F H: 88 % 11:23 91%

Xournal

File Edit View Page Tools Options Help

Sans 12

Data structures that are good

- List
- Dictionary
- Chars
- Strings.

Page 1 of 1 Layer: Layer 1

Terminal | Xournal | CS75: Schedule - Mozilla... | 11:24 | 91%

Xournal

File Edit View Page Tools Options Help

Sans 12

Data structures that are good (features)

- List.
- Dictionary
- Chars
- Strings

Page 1 of 1 Layer: Layer 1

Terminal | Xournal | CS75: Schedule - Mozilla...

T: 30 °F A: 17 °F H: 88 % 11:25 91%

Data structures that are good (features)

- List
- Dictionary
- Chars
- Strings
- Higher-order functions ("lambda" — λx.x — fun x → x) Lam

Data structures that are good (features)

- List
- Dictionary
- Chars
- Strings
- Higher-order functions ("lambda" — λx.x — fun x → x)
- Objects/clas

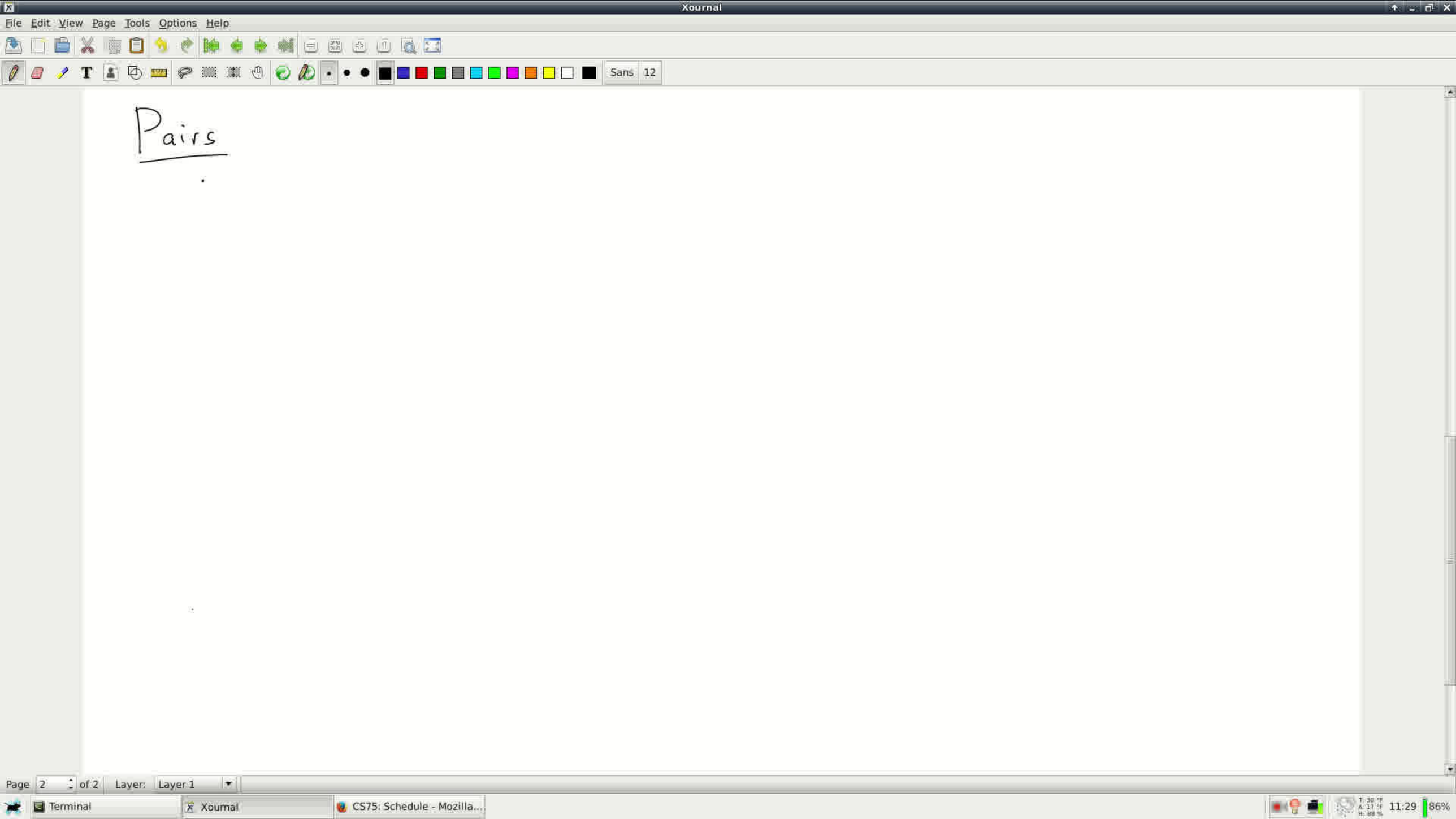
lambda x: x Lambda calculus OCaml

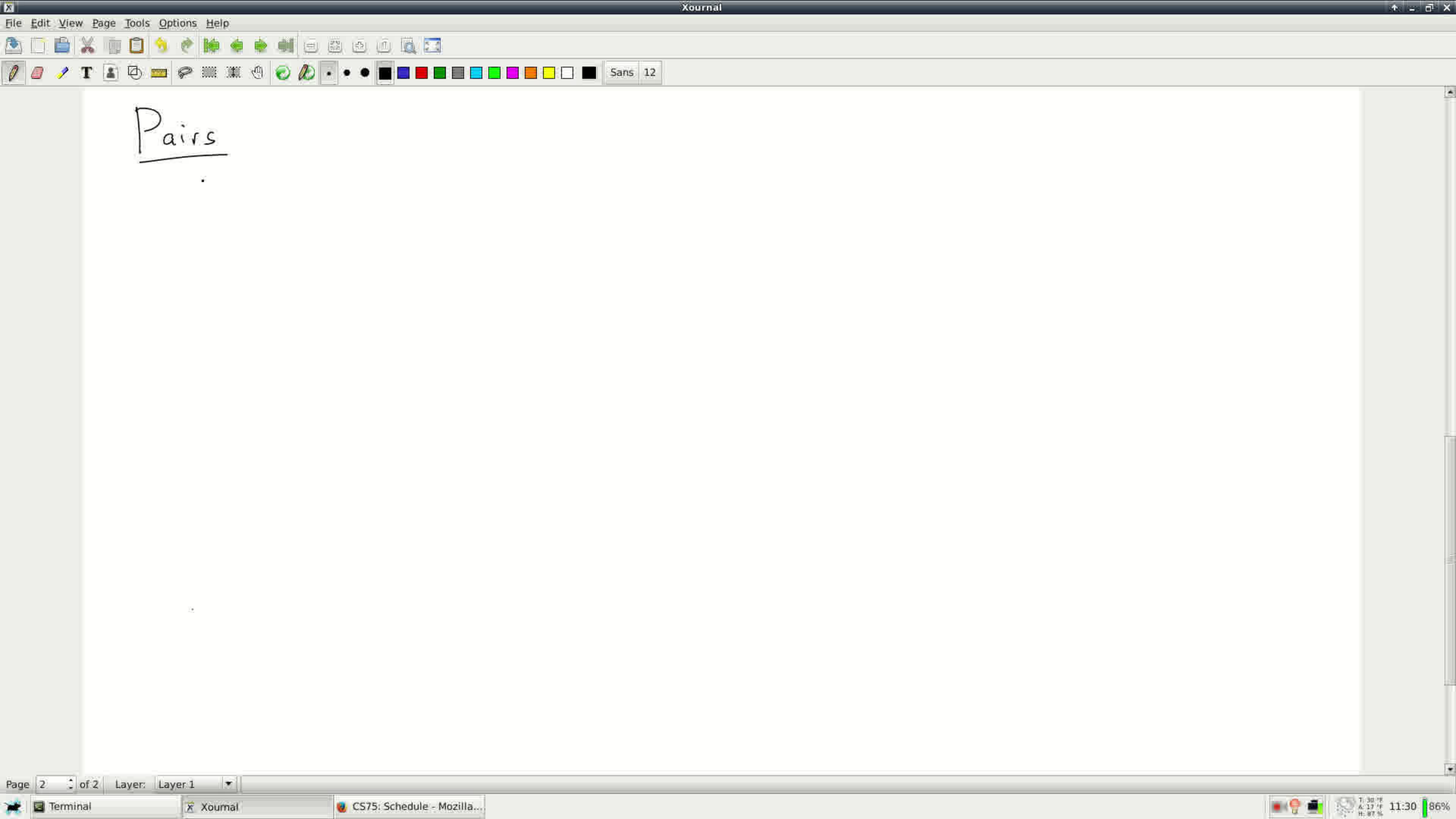
Data structures that are good (features)

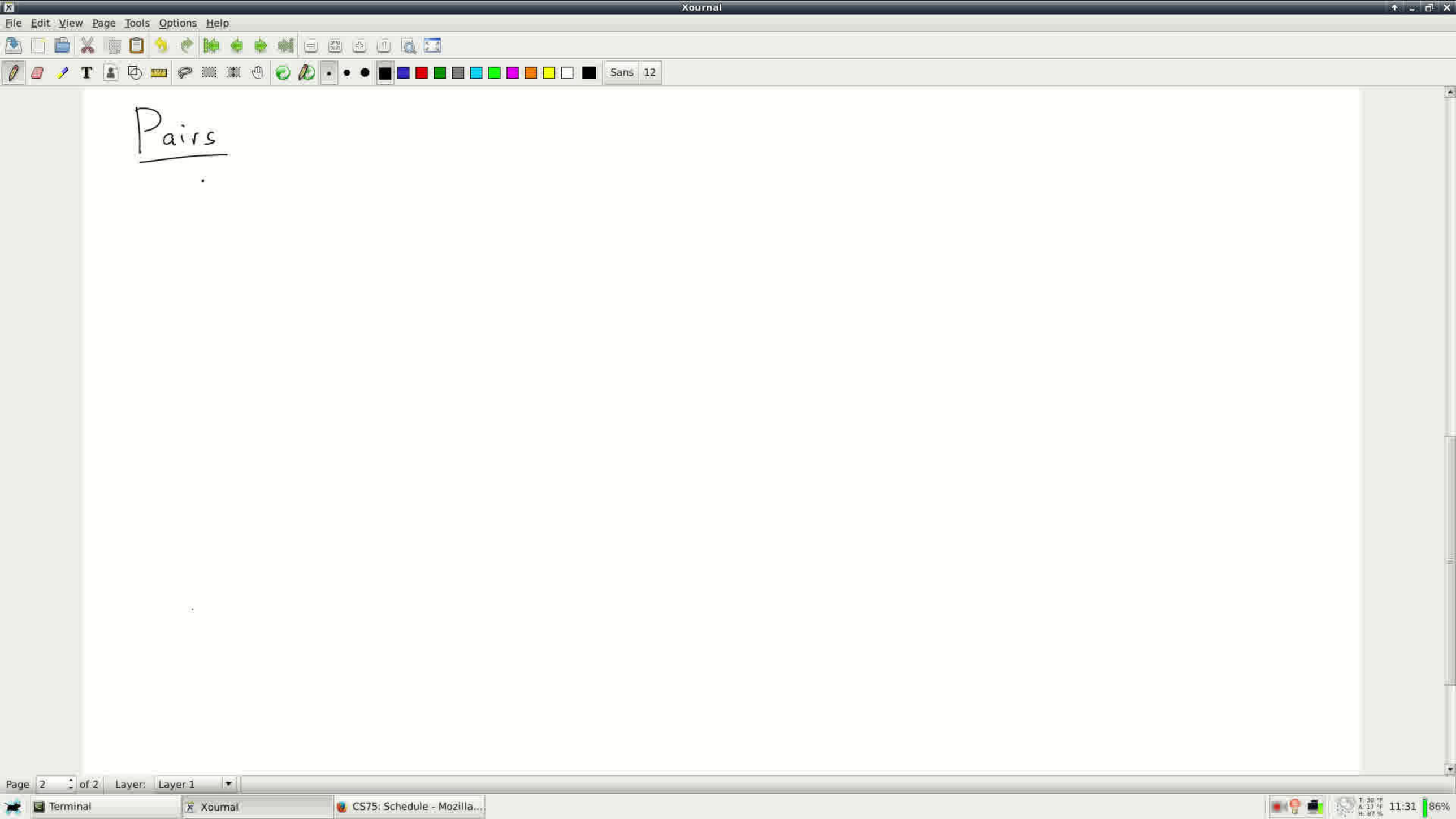
- List
- Dictionary
- Chars
- Strings
- Higher-order functions ("lambda" — λx.x — lambda x: x — Lambda calculus — fun x → x — OCaml)
- Objects/classes
- Structs/records

Data structures that are good (features)

- List
- Dictionary
- Chars
- Strings
- Higher-order functions ("lambda" — λx.x — fun x → x)
  - lambda x: x
  - Lambda calculus
  - OCaml
- Objects/classes
- Structs/records


Xournal
File Edit View Page Tools Options Help
Sans 12
Pairs
Page 2 of 2 Layer: Layer 1
Terminal
Xournal
CS75: Schedule - Mozilla...
11:29
86%


Xournal
File Edit View Page Tools Options Help
Sans 12
Pairs
Page 2 of 2 Layer: Layer 1
Terminal
Xournal
CS75: Schedule - Mozilla...
11:30
86%


Xournal
File Edit View Page Tools Options Help
Sans 12
Pairs
Page 2 of 2 Layer: Layer 1
Terminal
Xournal
CS75: Schedule - Mozilla...
11:31
86%

Pairs

```
type argument =
    | XConstant
    | Xf
```

Pairs

```
type argument =
    | XConstant of ....
    | XRegister of ....
    | XMemory of ....
```

Pairs

```
type argument =
    | XConstant of ....
    | XRegister of ....
    | XMemory of ....
```

Pairs

```
type argument =
    | XConstant of ....
    | XRegister of ....
    | XMemory of ....
```

```
<expr> ::= ....
        | (<expr>, <expr>)
```

Pairs

```
type argument =
    | XConstant of ....
    | XRegister of ....
    | XMemory of ....
```

<expr> ::= ....

| (<expr>, <expr>)

| EPair of expr * expr .

Pairs

```
type argument =
    | XConstant of ...
    | XRegister of ...
    | XMemory of ...
```

Concrete Syntax

```
<expr> ::= ....
        | ( expr>, <expr> )
```

Abstract Syntax

```
| EPair of expr *expr
```

Pairs

```
type argument =
    | XConstant of ....
    | XRegister of ....
    | XMemory of ....
```

Concrete Syntax

```
<expr> ::= ....
        | ( expr>, <expr> )
```

Abstract Syntax

```
type expr =
    | EPair of expr *expr
    ....
    .
```

```
type a_expr =
```

```
type c_expr =
```

```
type i_expr =
```

Pairs

```
type argument =
    | XConstant of ....
    | XRegister of ....
    | XMemory of ....
```

Concrete Syntax

```
<expr> ::= ....
    | ( <expr>, <expr> )
```

```
( let x = 5 in x, 7 )
```

Abstract Syntax

```
type expr =
    | EPair of expr * expr
    ....
```

```
type a_expr =
```

```
type c_expr =
```

```
type i_expr =
```

Pairs

```
type argument =
    | XConstant of ...
    | XRegister of ...
    | XMemory of ...
```

Concrete Syntax

```
<expr> ::= ....
    | ( <expr>, <expr> )
```

```
(let x = 5 in x, 7)
```

Abstract Syntax

```
type expr =
    | EPair of expr * expr
    ....
    .
```

```
type a_expr = ...
```

```
type c_expr =
    | CPair of.
```

```
type i_expr = ....
```

Pairs

```
type argument =
    | XConstant of ....
    | XRegister of ....
    | XMemory of ....
```

Concrete Syntax

```
<expr> ::= ....
        | ( <expr>, <expr> )
```

```
(let x=5 in x, 7)
((2.
```

Abstract Syntax

```
type expr =
    | EPair of expr * expr
    ....
    .
```

```
type a_expr = ...
```

```
type c_expr =
    | CPair of
```

```
type i_expr = ....
```

Pairs

```
type argument =
    | XConstant of ....
    | XRegister of ....
    | XMemory of ....
```

Concrete Syntax

```
<expr> ::= ....
        | ( <expr>, <expr> )
```

Abstract Syntax

```
type expr =
    | EPair of expr * expr
    ....
```

```
type a_expr = ...
```

```
type c_expr =
    | CPair of
        i_expr * i_expr
```

```
type i_expr = ....
```

```
(let x = 5 in x, 7)
((2, 3), 4)
```

```
let $1 = (2,3) in
($1, 4)
```

<expr> ::= ....

| (<expr>, <expr>)

(let x = 5 in x, 7)

((2, 3), 4)

let $1 = (2,3) in
($1, 4)

Abstract Syntax

```
type expr =
    | EPair of expr * expr
    ....
```

type a_expr = ...

type c_expr =
    | CPair of
        i_expr * i_expr

type i_expr = ...

Memory Repr.

<expr> ::= ....

| (<expr>, <expr>)

(let x = 5 in x, 7)

((2, 3), 4)

let $1 = (2,3) in
($1, 4)

Abstract Syntax

```
type expr =
    | EPair of expr * expr
    ....
```

```
type a_expr = ...
```

```
type c_expr =
    | CPair of
        i_expr * i_expr
```

```
type i_expr = ....
```

Memory Repr.

2^{31} integers.

<expr> ::= ....
| (<expr>, <expr>)

(let x = 5 in x, 7)

((2, 3), 4)

let $1 = (2,3) in
($1, 4)

Abstract Syntax

```
type expr =
    | EPair of expr * expr
    ....
```

```
type a_expr = ...
```

```
type c_expr =
    | CPair of
        i_expr * i_expr
```

```
type i_expr = ....
```

Memory Repr.

2^{31} integers — ends with 0
2 booleans —

<expr> ::= ....

| (<expr>, <expr>)

(let x = 5 in x, 7)

((2, 3), 4)

let $1 = (2,3) in
($1, 4)

Abstract Syntax

```
type expr =
    | EPair of expr * expr
    ....
```

```
type a_expr = ...
```

```
type c_expr =
    | CPair of
        i_expr * i_expr
```

```
type i_expr = ....
```

Memory Repr.

2^{31} integers — ends with 0

2 booleans — ends with 11

2^{30} pointers — ends with 01

<expr> ::= ....
| (<expr>, <expr>)

(let x = 5 in x, 7)

((2, 3), 4)

⇓

let $1 = (2,3) in
($1, 4)

Abstract Syntax

```
type expr =
    | EPair of expr * expr
    ....
```

```
type a_expr = ...
```

```
type c_expr =
    | CPair of
        i_expr * i_expr
```

```
type i_expr = ....
```

Memory Repr.

2^{31} integers — ends with 0
2 booleans — ends with 11
2^{30} pointers — ends with 01

<expr> ::= ....
| (<expr>, <expr>)

(let x=5 in x, 7)

((2,3),4)

let $1 = (2,3) in
($1, 4)

Abstract Syntax

```
type expr =
    | EPair of expr * expr
    ....
```

type a_expr = ...

type c_expr =
    | CPair of
        i_expr * i_expr

type i_expr = ...

⟶ 0x88404079

Memory Repr.

2^{31} integers — ends with 0
2 booleans — ends with 11
2^{30} pointers — ends with 01

<expr> ::= ....
| (<expr>, <expr>)

(let x=5 in x, 7)

((2,3),4)

let $1 = (2,3) in
($1, 4)

Abstract Syntax

```
type expr =
    | EPair of expr * expr
    ....
```

type a_expr = ...

type c_expr =
    | CPair of
        i_expr * i_expr

type i_expr = ...

Memory Repr.

2^{31} integers — ends with 0
2 booleans — ends with 11
2^{30} pointers — ends with 01

→ 0x88404079
0x8840407[10

<expr> ::= ....
| (<expr>, <expr>)

(let x=5 in x, 7)

((2,3),4)

let $1 = (2,3) in
($1, 4)

Abstract Syntax

```
type expr =
    | EPair of expr * expr
    ....
```

type a_expr = ...

type c_expr =
    | CPair of
        i_expr * i_expr

type i_expr = ...

Memory Repr.

2^{31} integers — ends with 0
2 booleans — ends with 11
2^{30} pointers — ends with 01

→ 0x88404078.
0x88404078[10̊01]

Xournal

File Edit View Page Tools Options Help

New Page Before
New Page After
New Page At End
New Pages Keep Background
Delete Page
New Layer
Delete Layer
Paper Size
Paper Color
Paper Style
Apply To All Pages
Load Background
Background Screenshot
Default Paper
Set As Default

Sans 12

(<expr>, <expr>)

type expr =
| EPair of expr * expr
.....

(let x = 5 in x, 7)

((2, 3), 4)

⇓

let $1 = (2,3) in
($1, 4)

type a_expr = ...

type c_expr =
| CPair of
i_expr * i_expr

type i_expr = ...

Memory Repr.

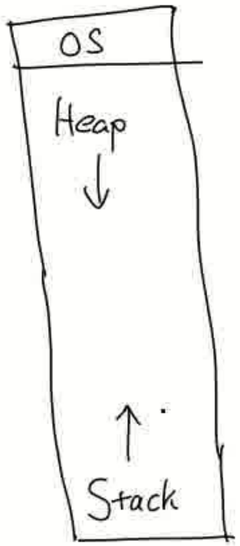
2^{31} integers — ends with 0
2 booleans — ends with 11
2^{30} pointers — ends with 01

⟶ 0x88404078.
0x88404078[10001]

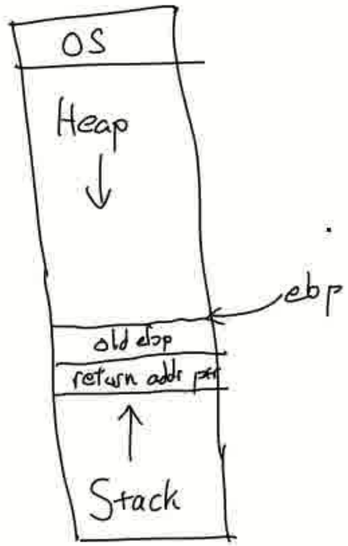
Page 2 of 2 Layer: Layer 1

Terminal | Xournal | CS75: Schedule - Mozilla... | 11:53 | 65%

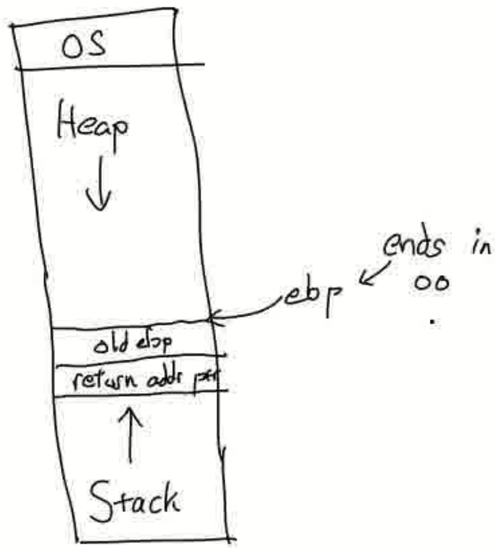
Memory Layout



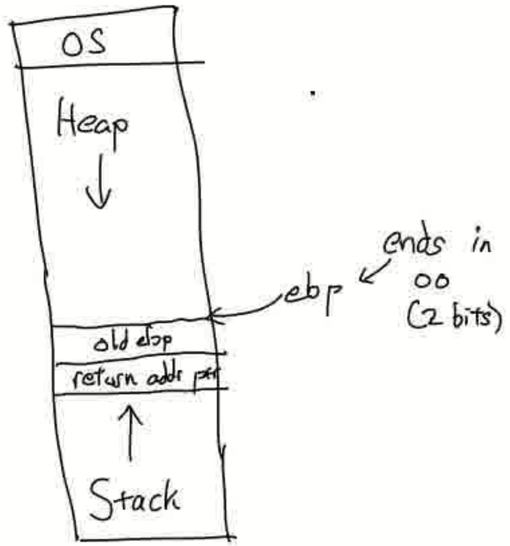
Memory Layout



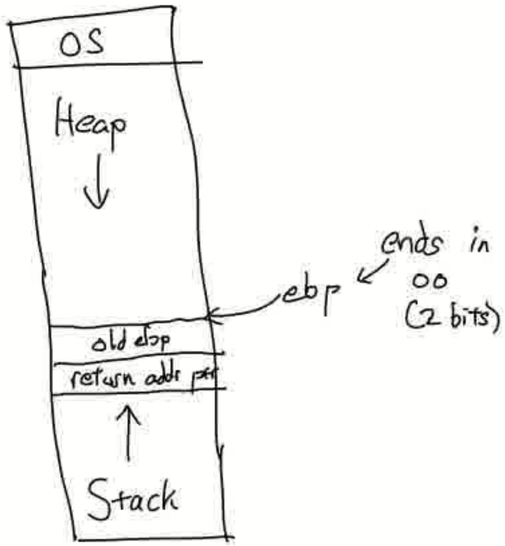
Memory Layout

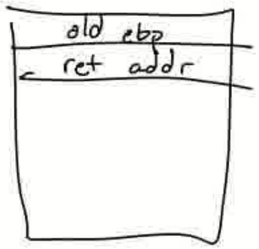


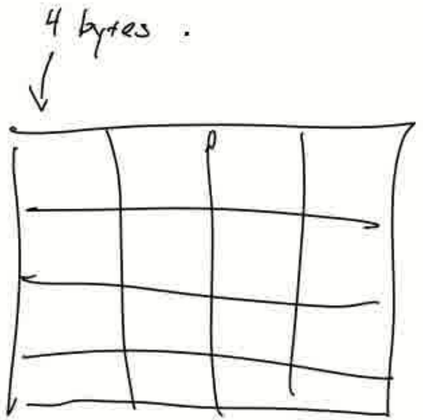
Memory Layout



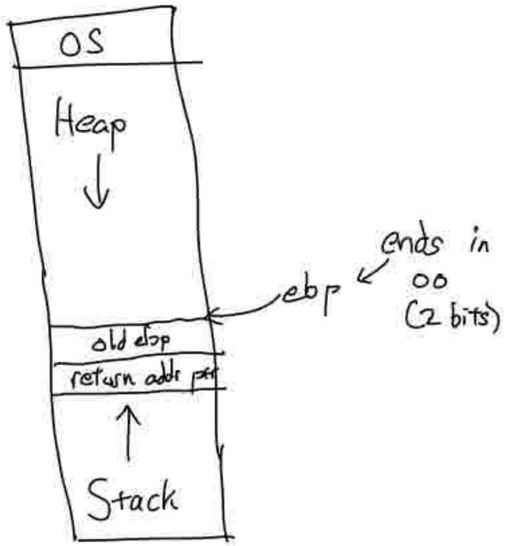
Memory Layout

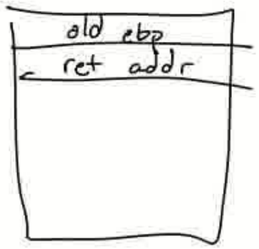


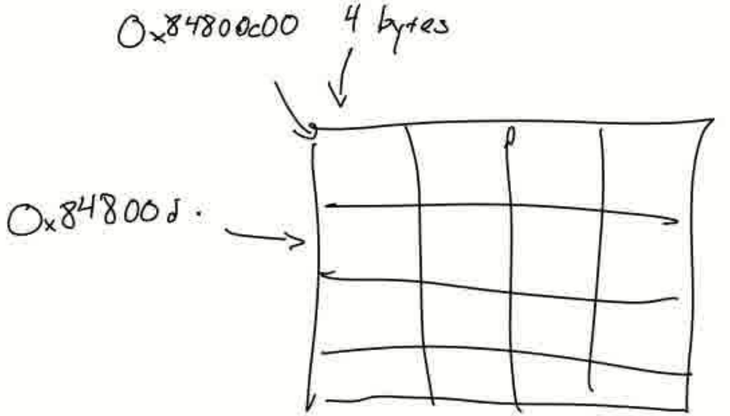




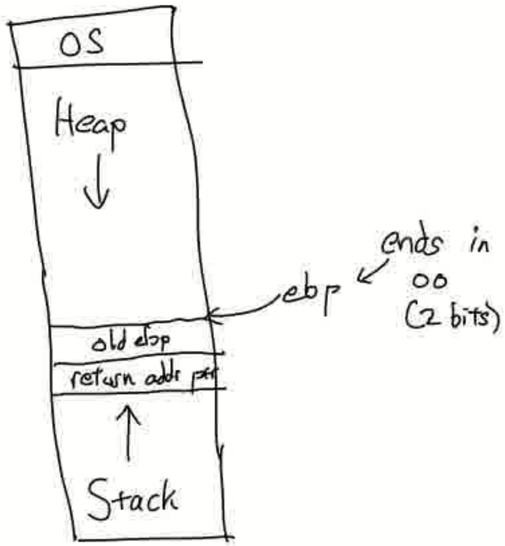
Memory Layout

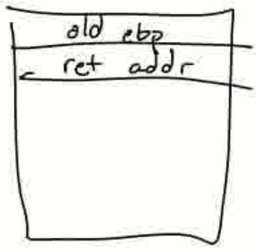


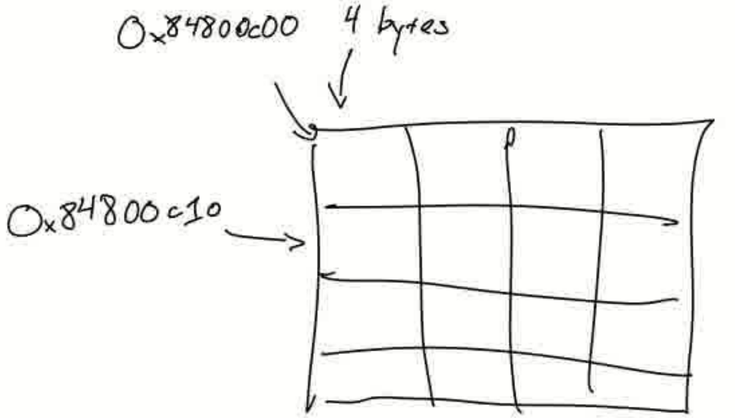




Memory Layout







last digit will be

0, 4, 8, c

i_expr * i_expr

Memory Repr.

2^{31} integers — ends with 0

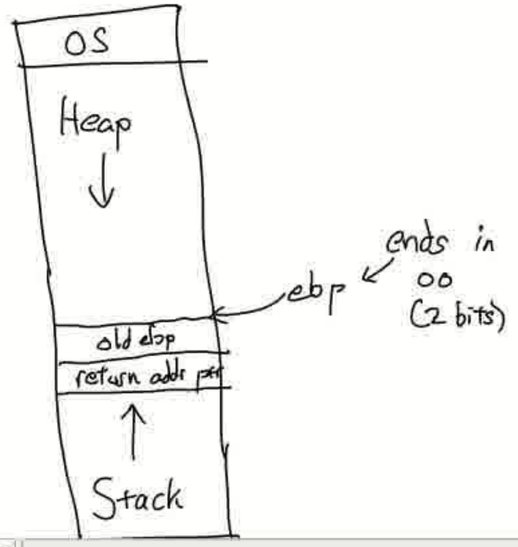
2 booleans — ends with 11

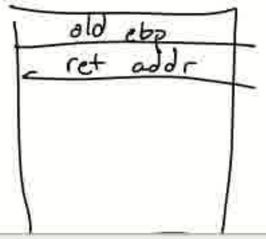
2^{30} pointers — ends with 01

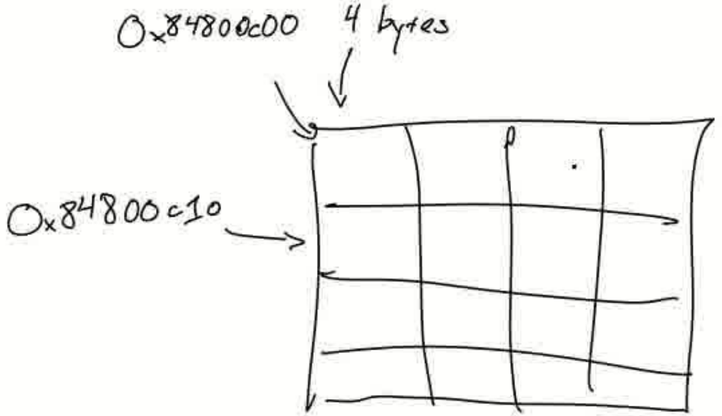
→ 0x88404078

0x88404071[1001]

Memory Layout







last digit will be

0 4 8 c

i_expr * i_expr

Memory Repr.

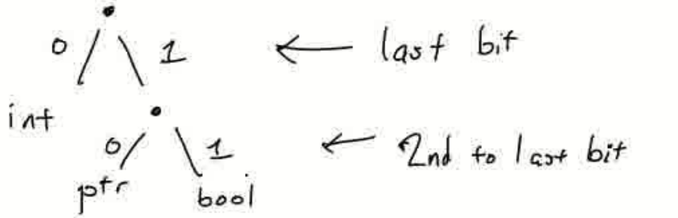
2^{31} integers — ends with 0

2 booleans — ends with 11

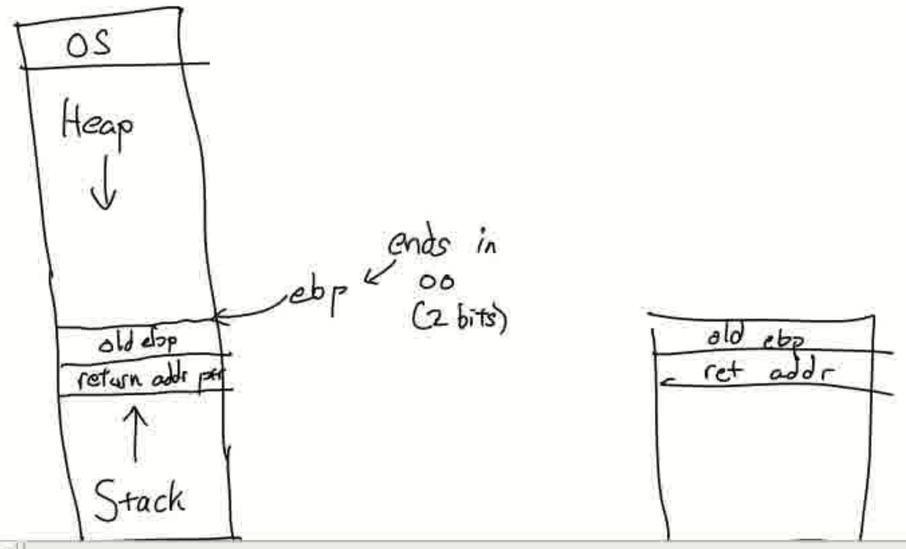
2^{30} pointers — ends with 01

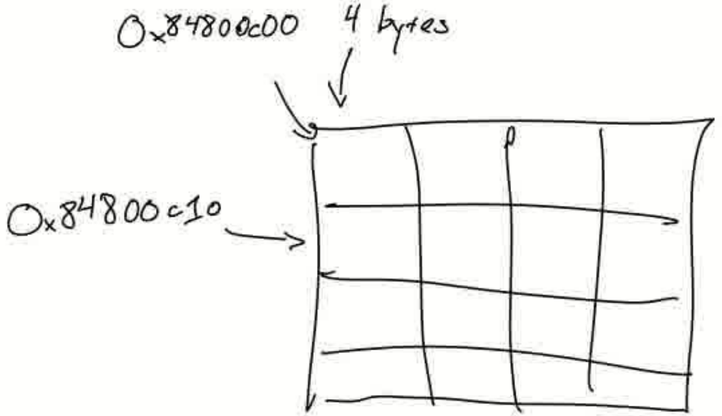
→ 0x88404078

0x88404071[1001]



Memory Layout





last digit will be

0 4 8 c

i_expr * i_expr

Memory Repr.

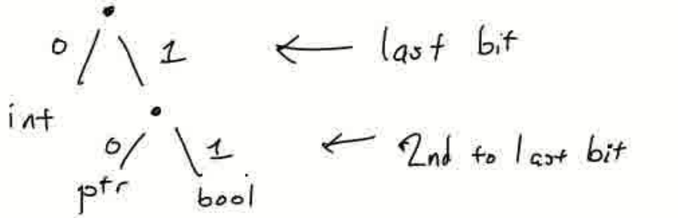
2^{31} integers — ends with 0

2 booleans — ends with 11

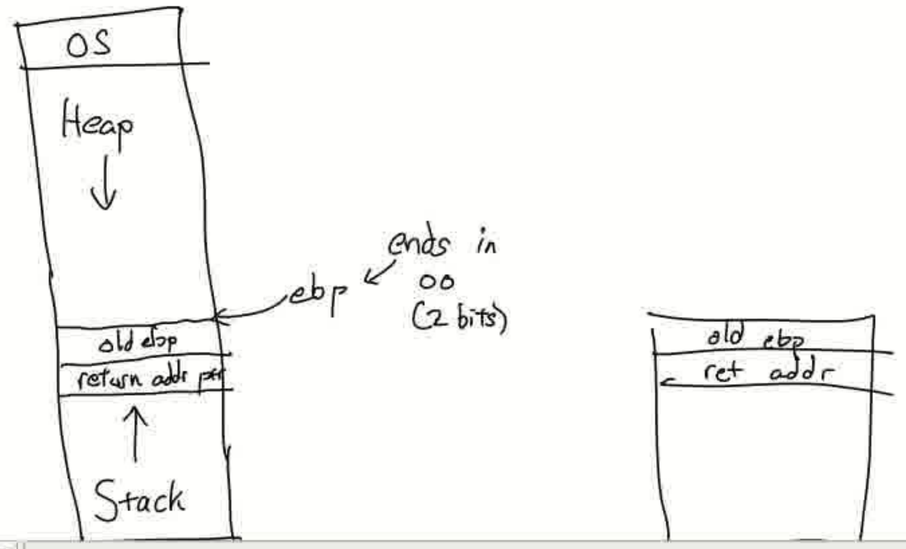
2^{30} pointers — ends with 01

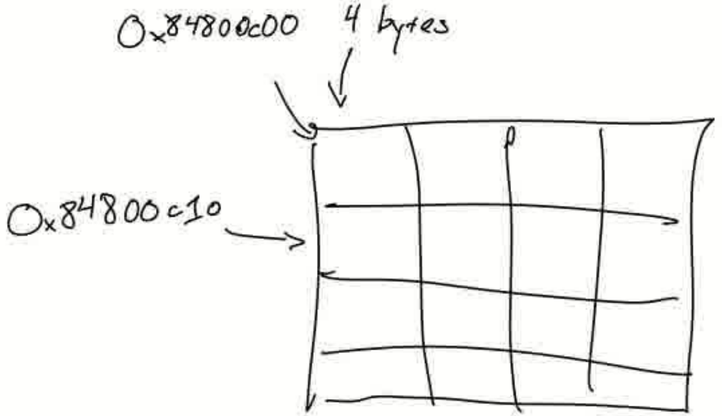
→ 0x88404078

0x88404071[1001]



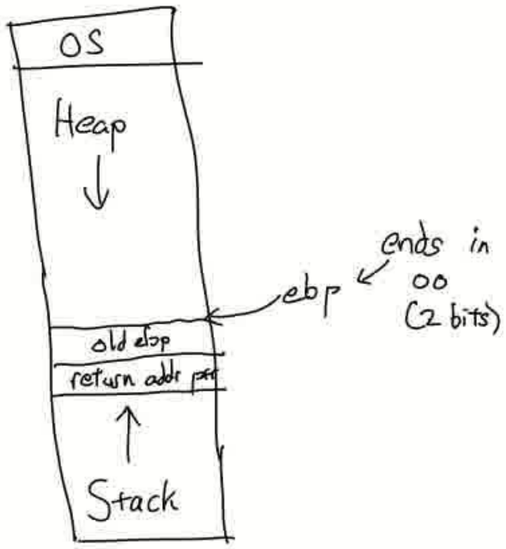
Memory Layout

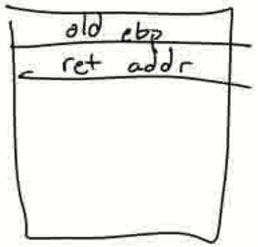


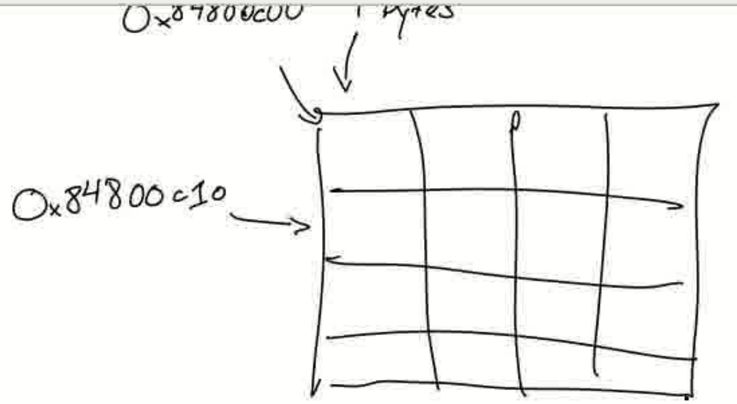


last digit will be

0 4 8 c

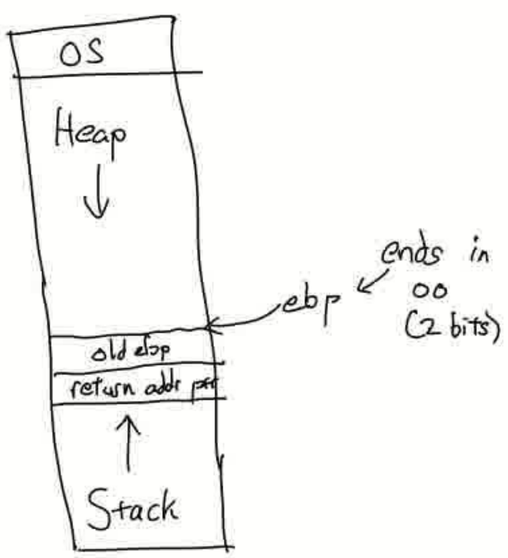


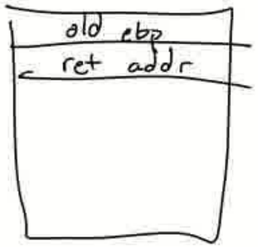


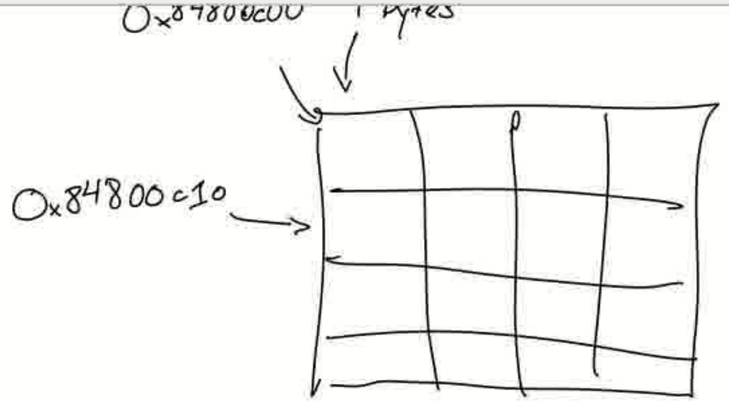


last digit will be

0, 4, 8, c



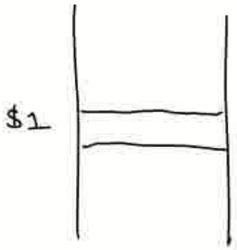


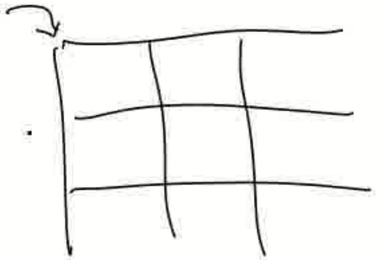


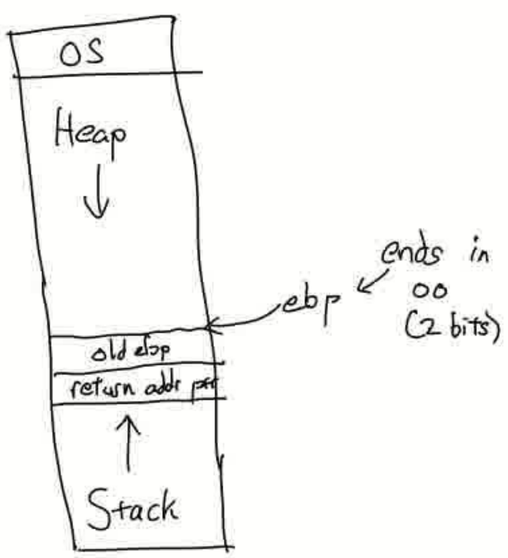
last digit will be

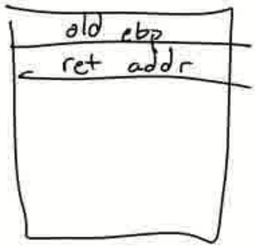
0, 4, 8, c

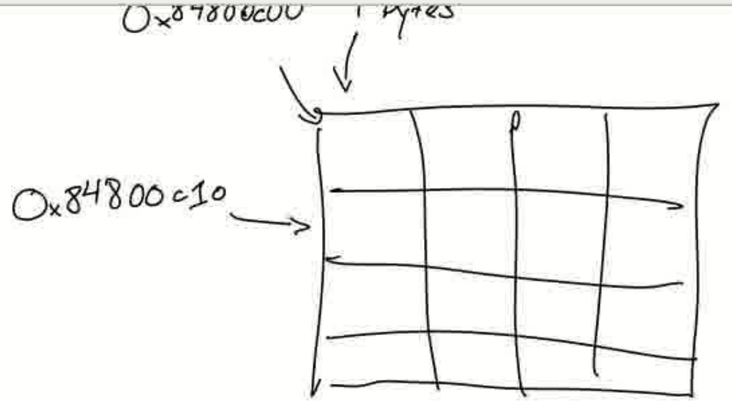
Stack







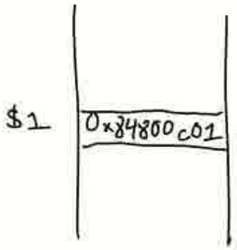


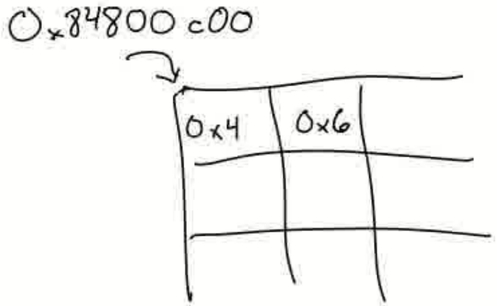


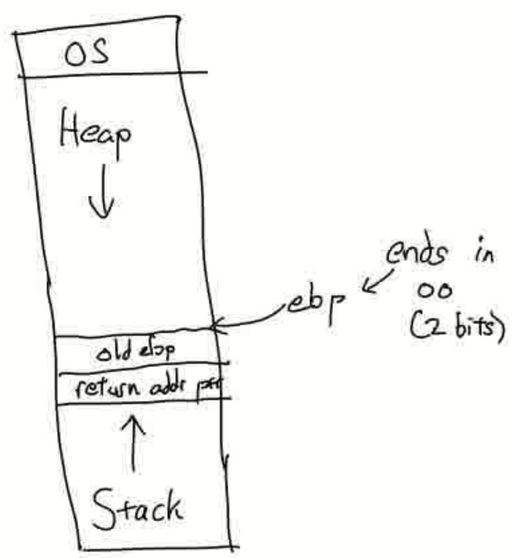
last digit will be

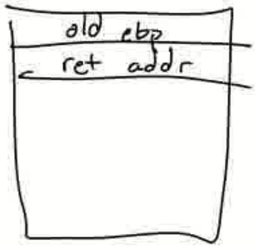
0, 4, 8, c

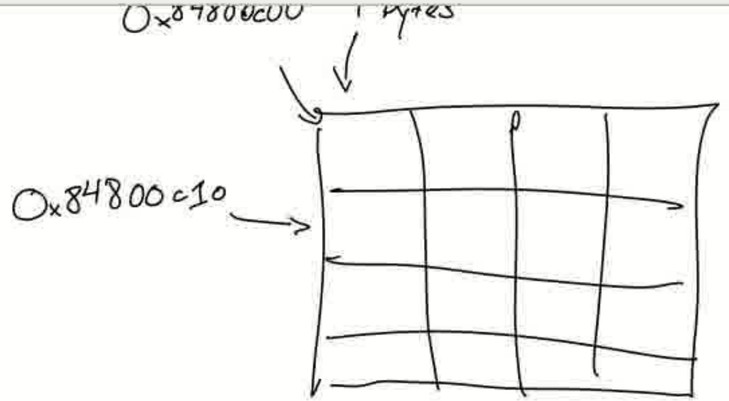
Stack







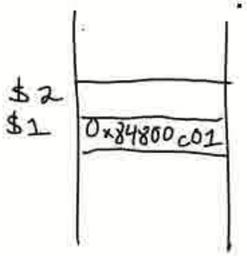


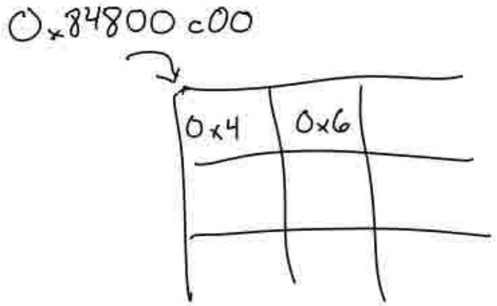


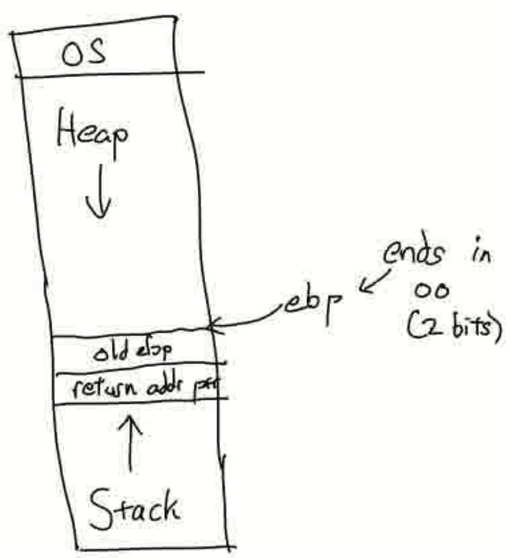
last digit will be

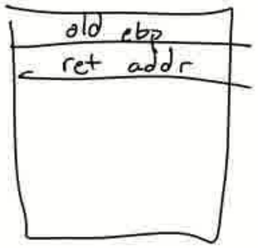
0, 4, 8, c

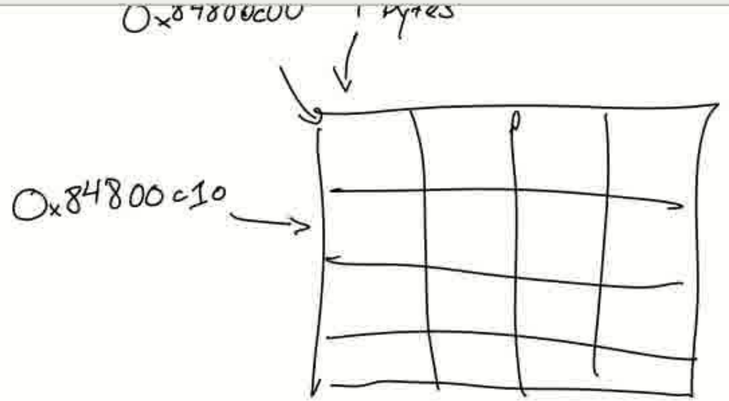
Stack







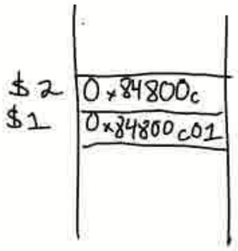


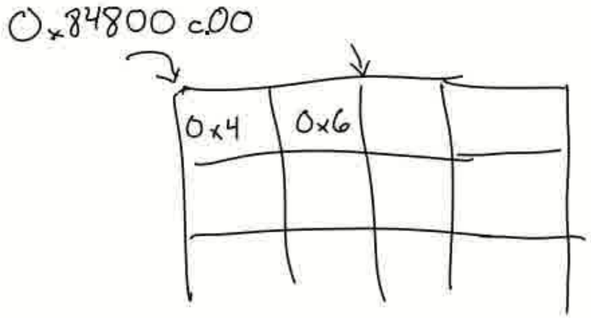


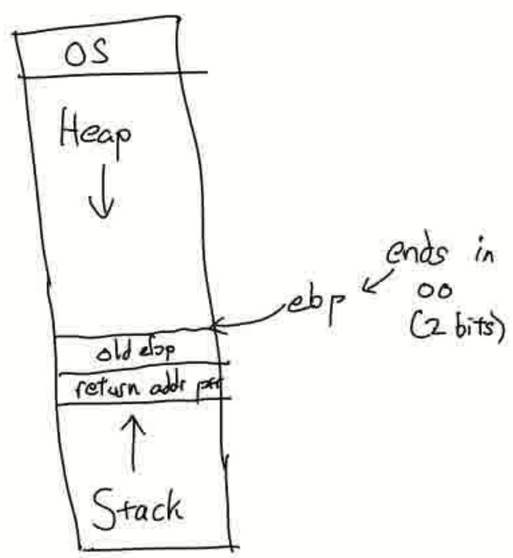
last digit will be

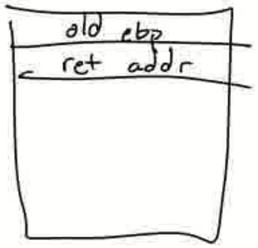
0, 4, 8, c

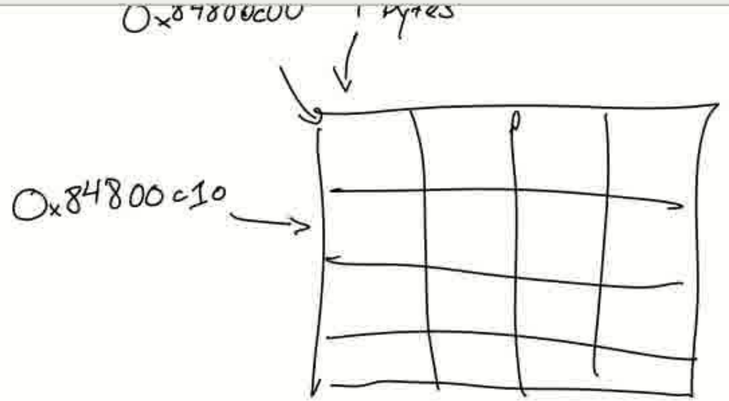
Stack







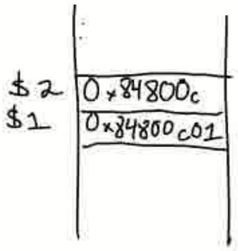


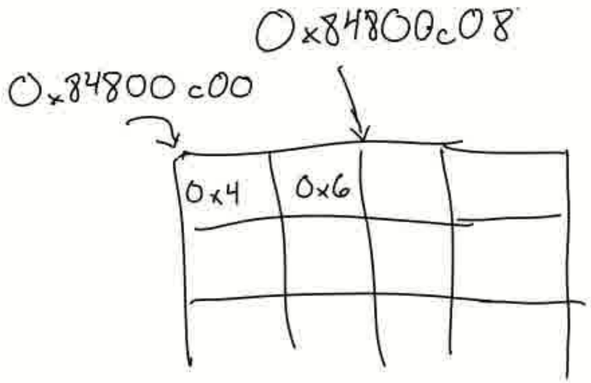


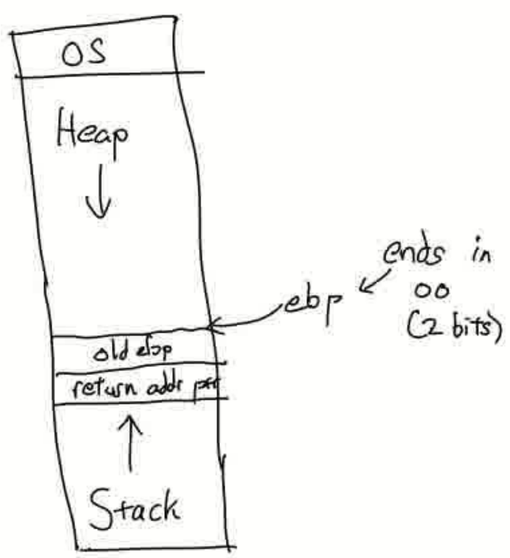
last digit will be

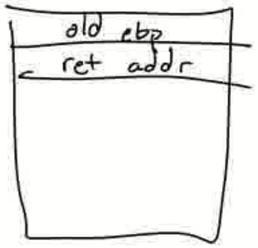
0, 4, 8, c

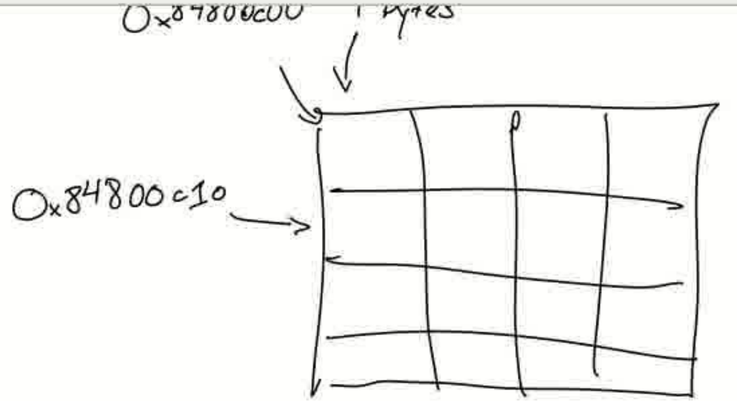
Stack







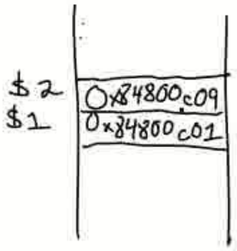


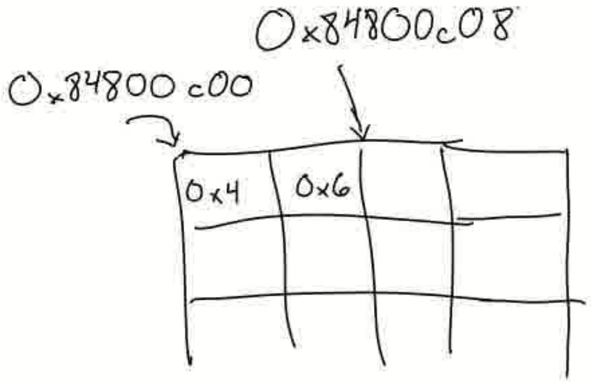


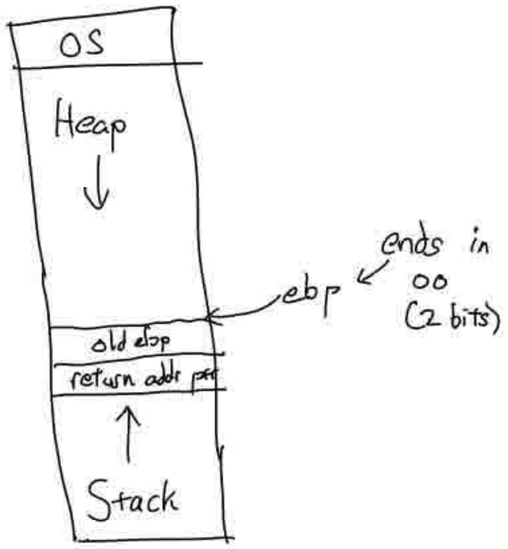
last digit will be

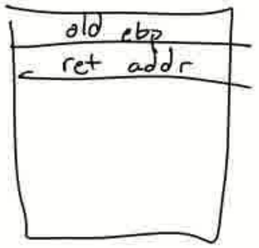
0, 4, 8, c

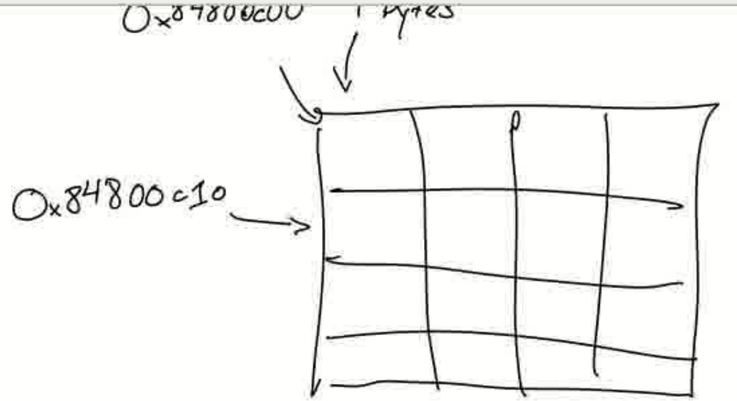
Stack







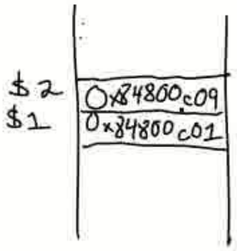


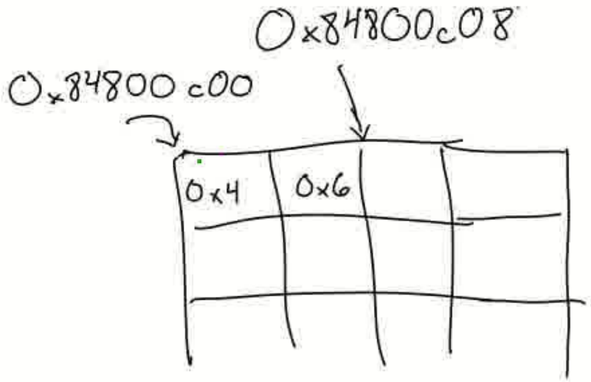


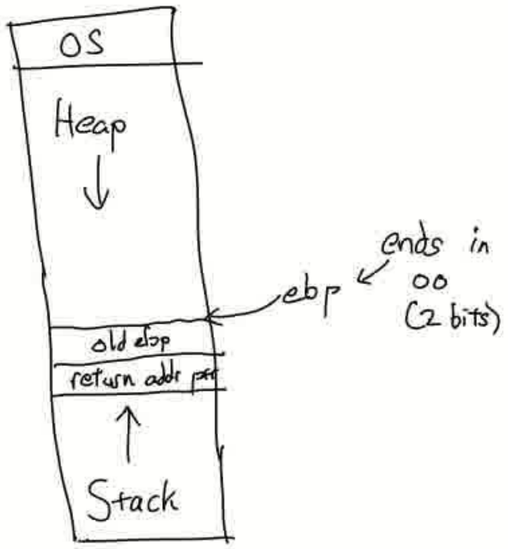
last digit will be

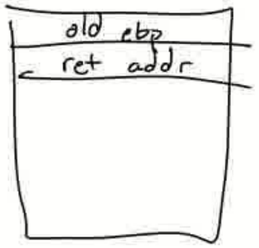
0, 4, 8, c

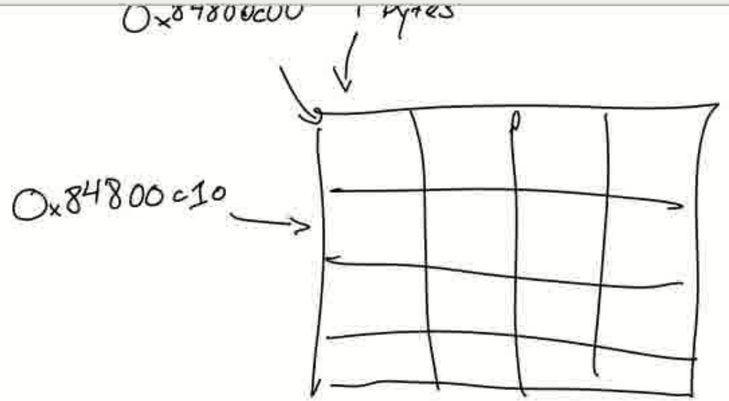
Stack







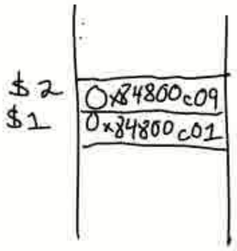


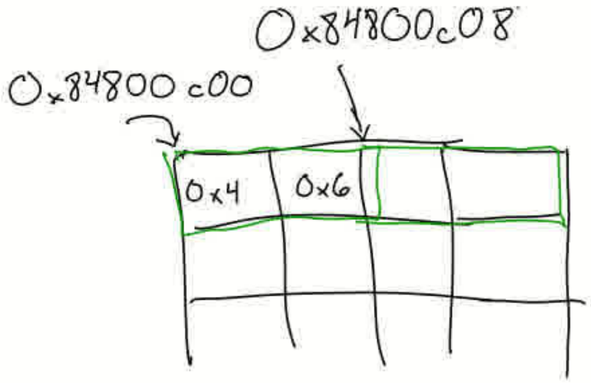


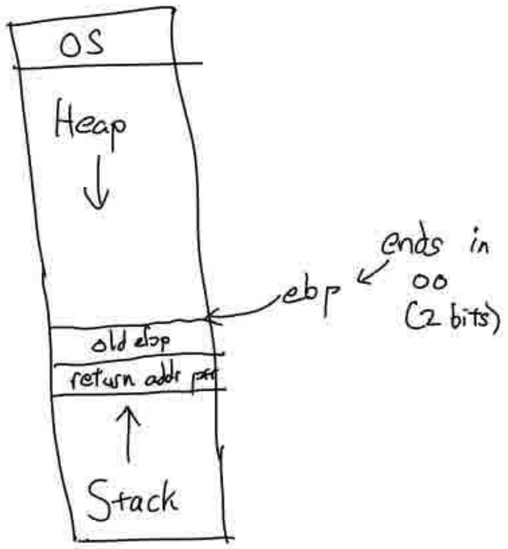
last digit will be

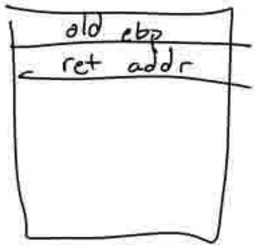
0, 4, 8, c

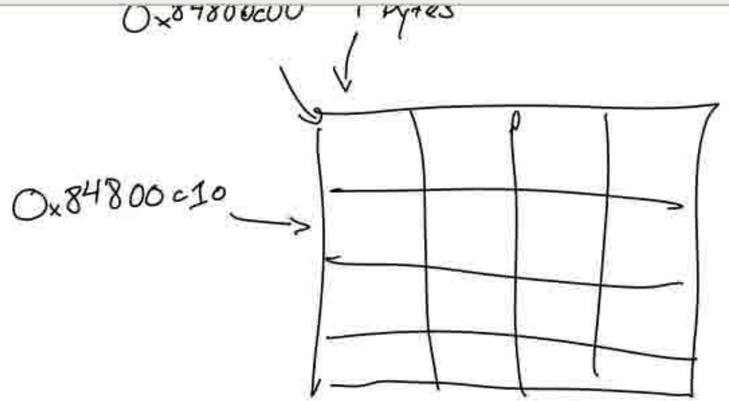
Stack







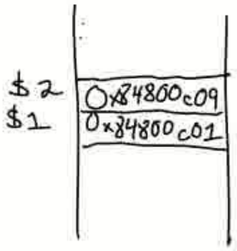


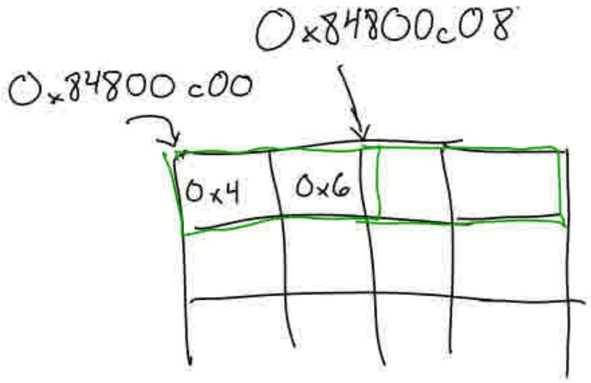


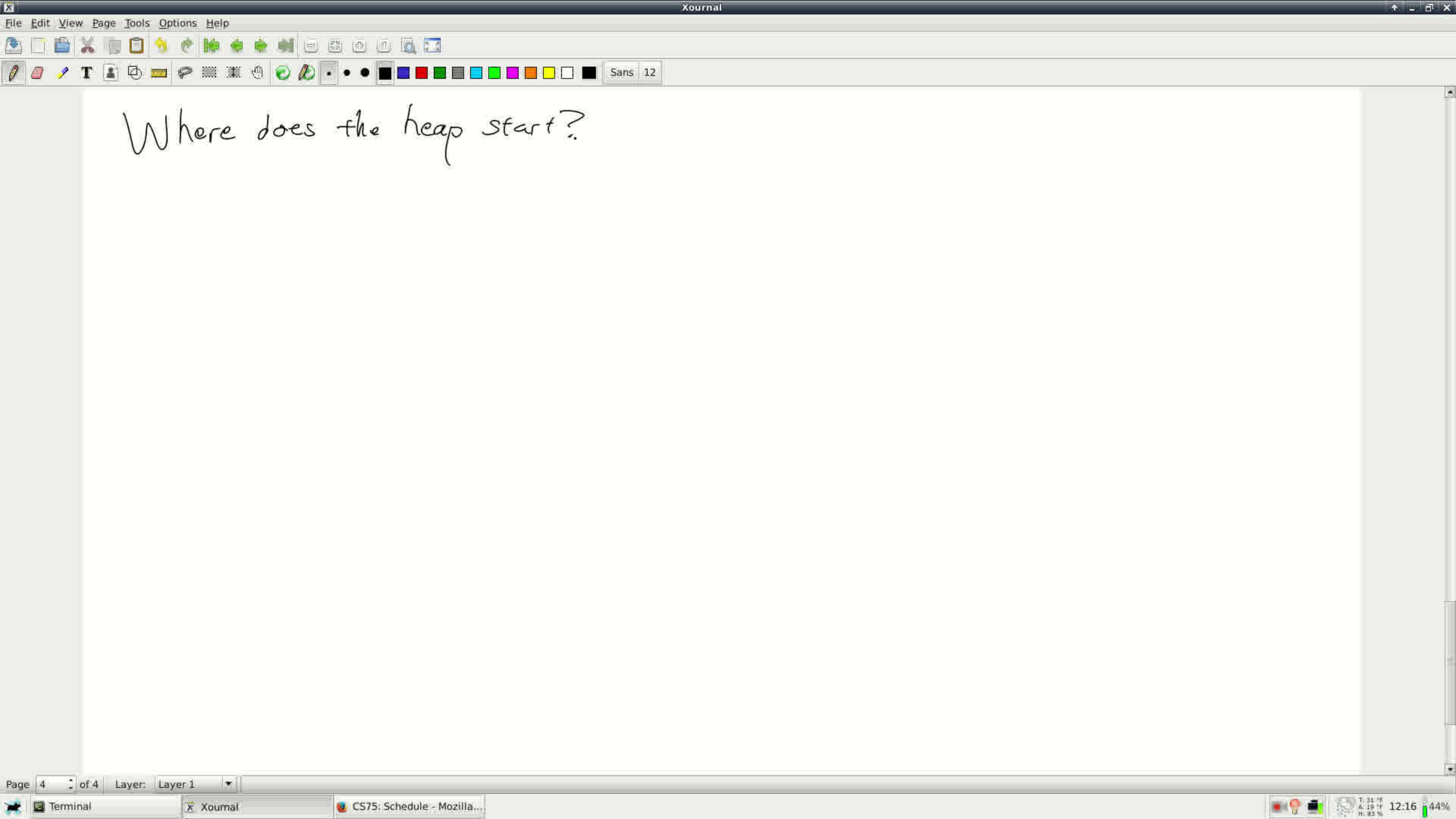
last digit will be

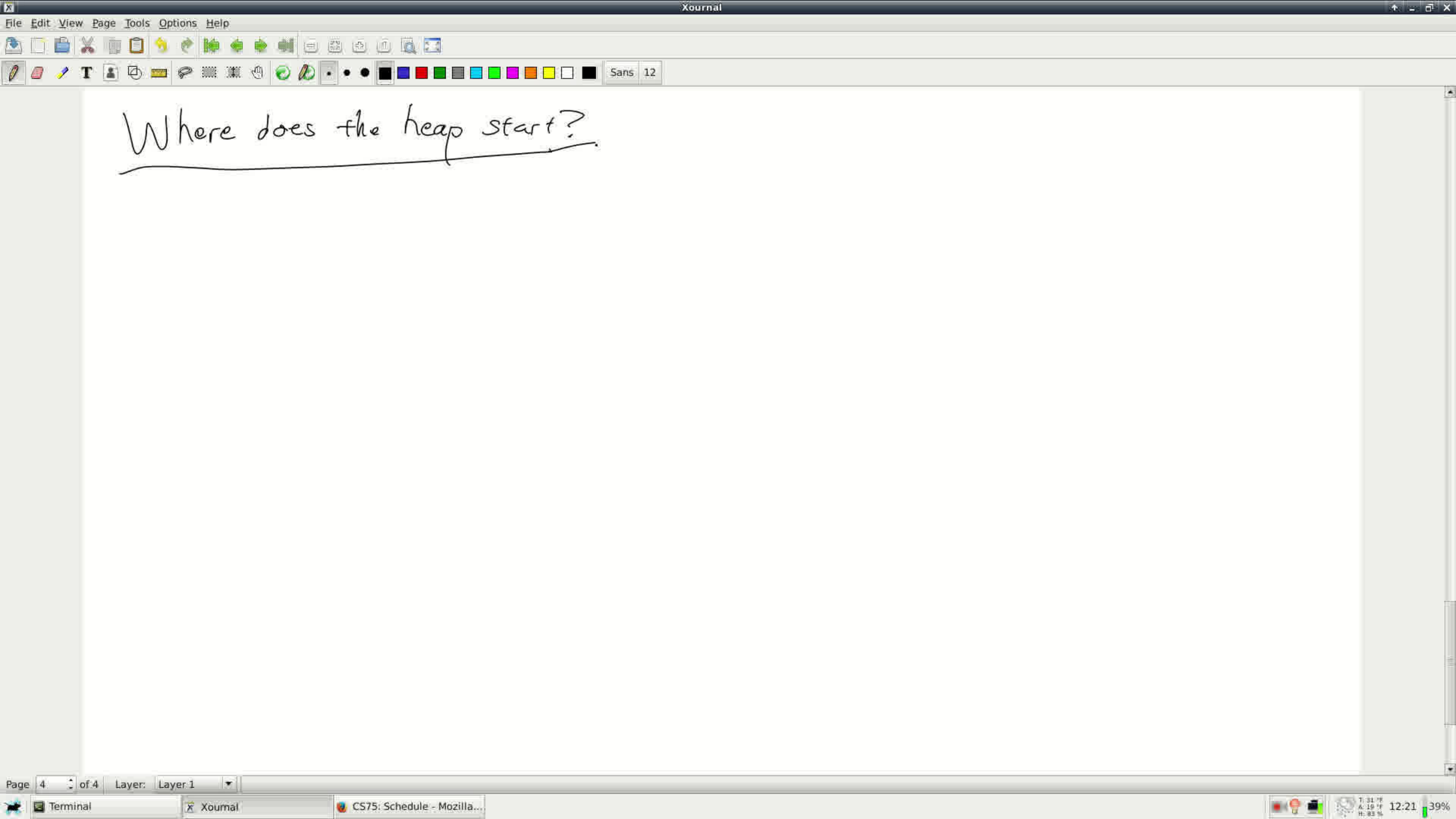
0, 4, 8, c

Stack






Xournal
File Edit View Page Tools Options Help
Sans 12
Where does the heap start?
Page 4 of 4 Layer: Layer 1
Terminal
Xournal
CS75: Schedule - Mozilla...
12:16
44%


Xournal
File Edit View Page Tools Options Help
Sans 12
Where does the heap start?
Page 4 of 4 Layer: Layer 1
Terminal
Xournal
CS75: Schedule - Mozilla...
12:21
39%

BRK(2) Linux Programmer's Manual BRK(2)

NAME

brk, sbrk - change data segment size

SYNOPSIS

```
#include <unistd.h>

int brk(void *addr);

void *sbrk(intptr_t increment);
```

Feature Test Macro Requirements for glibc (see **feature_test_macros**(7)):

brk(), **sbrk**():

```
Since glibc 2.19:
    _DEFAULT_SOURCE ||
        (_XOPEN_SOURCE >= 500) &&
        ! (_POSIX_C_SOURCE >= 200112L)
From glibc 2.12 to 2.19:
    _BSD_SOURCE || _SVID_SOURCE ||
        (_XOPEN_SOURCE >= 500) &&
        ! (_POSIX_C_SOURCE >= 200112L)
Before glibc 2.12:
    _BSD_SOURCE || _SVID_SOURCE || _XOPEN_SOURCE >= 500
```

Manual page brk(2) line 1 (press h for help or q to quit)

BRK(2) Linux Programmer's Manual BRK(2)

NAME

brk, sbrk - change data segment size

SYNOPSIS

```
#include <unistd.h>

int brk(void *addr);

void *sbrk(intptr_t increment);
```

Feature Test Macro Requirements for glibc (see **feature_test_macros**(7)):

brk(), **sbrk**():

```
Since glibc 2.19:
    _DEFAULT_SOURCE ||
        (_XOPEN_SOURCE >= 500) &&
        ! (_POSIX_C_SOURCE >= 200112L)
From glibc 2.12 to 2.19:
    _BSD_SOURCE || _SVID_SOURCE ||
        (_XOPEN_SOURCE >= 500) &&
        ! (_POSIX_C_SOURCE >= 200112L)
Before glibc 2.12:
    _BSD_SOURCE || _SVID_SOURCE || _XOPEN_SOURCE >= 500
```

Manual page brk(2) line 1 (press h for help or q to quit)

Where does the heap start?

In driver.c: call malloc to get some memory (4Mb?).

Where does the heap start?

In driver.c: call malloc to get some memory (4Mb?)

pass that pointer to snake_main.

Where does the heap start?

In driver.c: call malloc to get some memory (4Mb?)

pass that pointer to snake_main

In snake_main — get ptr and store it