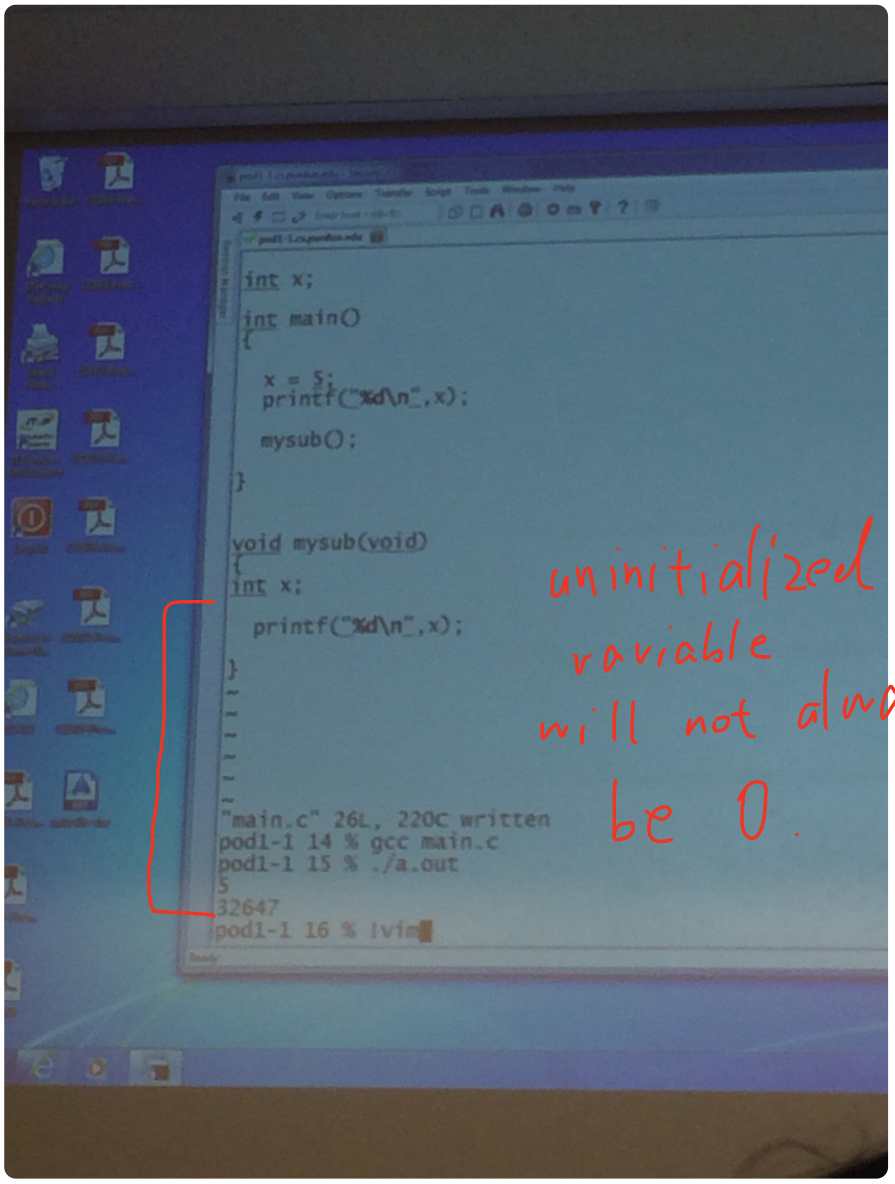
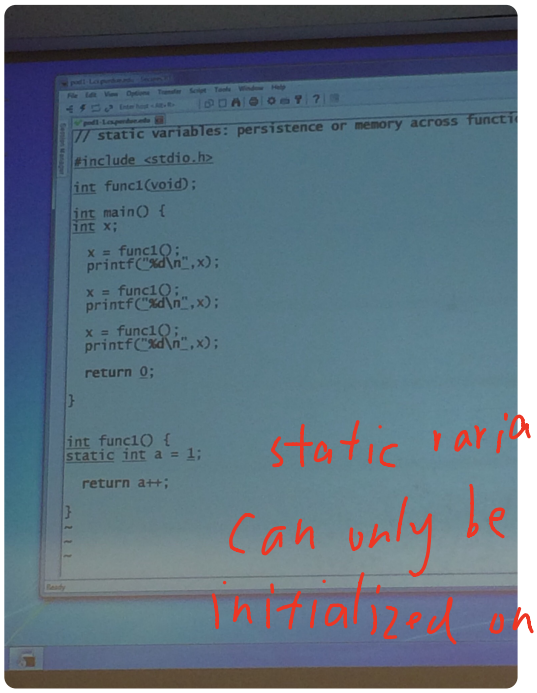
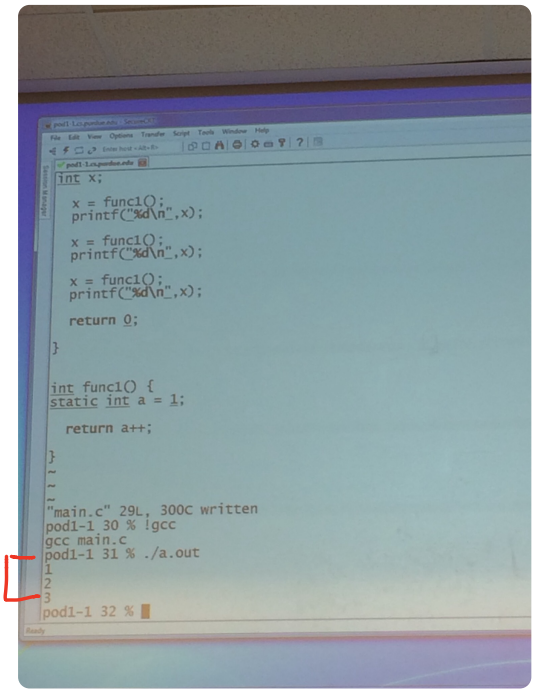

int x;
int main()
{
x = 5;
printf("%d\n",x);
mysub();
}
void mysub(void)
{
int x;
printf("%d\n",x);
}
"main.c" 26L, 220C written
pod1-1 14 % gcc main.c
pod1-1 15 % ./a.out
5
32647
pod1-1 16 % !vim
uninitialized
variable
will not always
be 0.



static variable can only be initialized once!



static int a = 1;

is located in data (global variable)

it's always be there

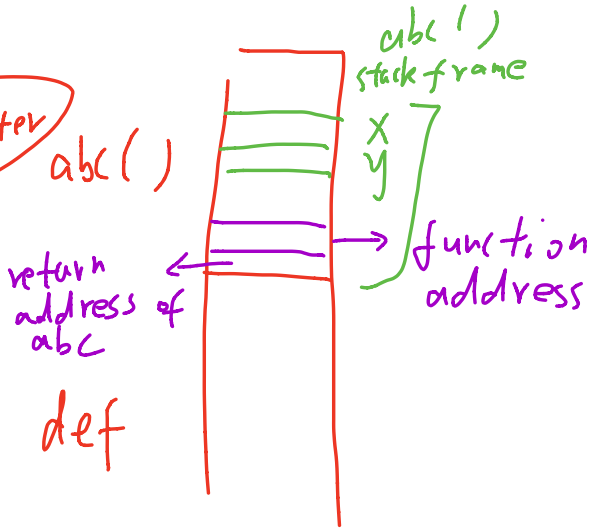
but, it's still local to his own scope within his curly braket

```c
// Pointers of functions and passing them as

#include <stdio.h>

int twice(int);
int runfunc(int (*g)(), int);

int main()
{
int x, y;
int (*f)(int);
 x = 3;
 y = twice(x);
 printf("result = %d\n",y);

 f = &twice;
 x = 4;
 y = (*f)(x);
 printf("result = %d\n",y);

 x = 5;
 y = runfunc(&twice, x);
 printf("result = %d\n",y);

}


// multiply by 2

int twice(int a)
{
"main.c" 46L, 521C
```

(*f) function pointer

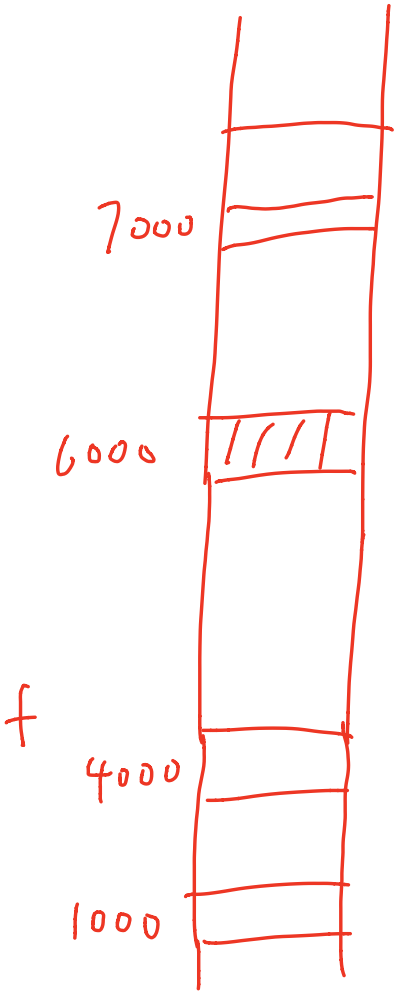
return an int;

can write as f = twice;




```c
   printf("result = %d\n",y);

   f = &twice;
   x = 4;
   y = (*f)(x);
   printf("result = %d\n",y);

   x = 5;
   y = runfunc(&twice, x);
   printf("result = %d\n",y);

}


// multiply by 2

int twice(int a)
{
  return 2*a;
}


// run function with a single argument

int runfunc(int (*h)(), int a)
{
int b;

  b = (*h)(a);
  return b;

}
```

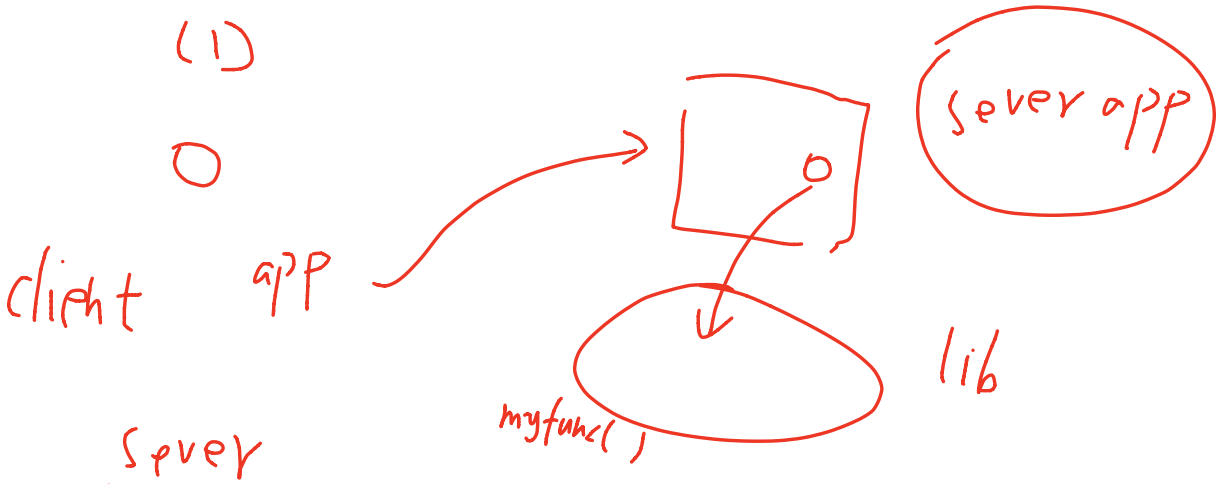
can write as b = h(a)




```c
// run function with a single argument

int runfunc(int (*h)(int), int a)
{
int b;

  b = (*h)(a);
  return b;

}
```

Call back function

(1)



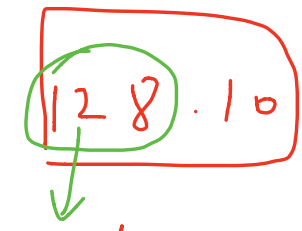
Sever

If the client connect me, then tell the client to run this function in lib

(2) Segmentation fault

If seg fault happen, don't kill me, tell linux, run this function. signal handler.

Lab 4



dotted decimal

purdue

32 bit = 4 byte

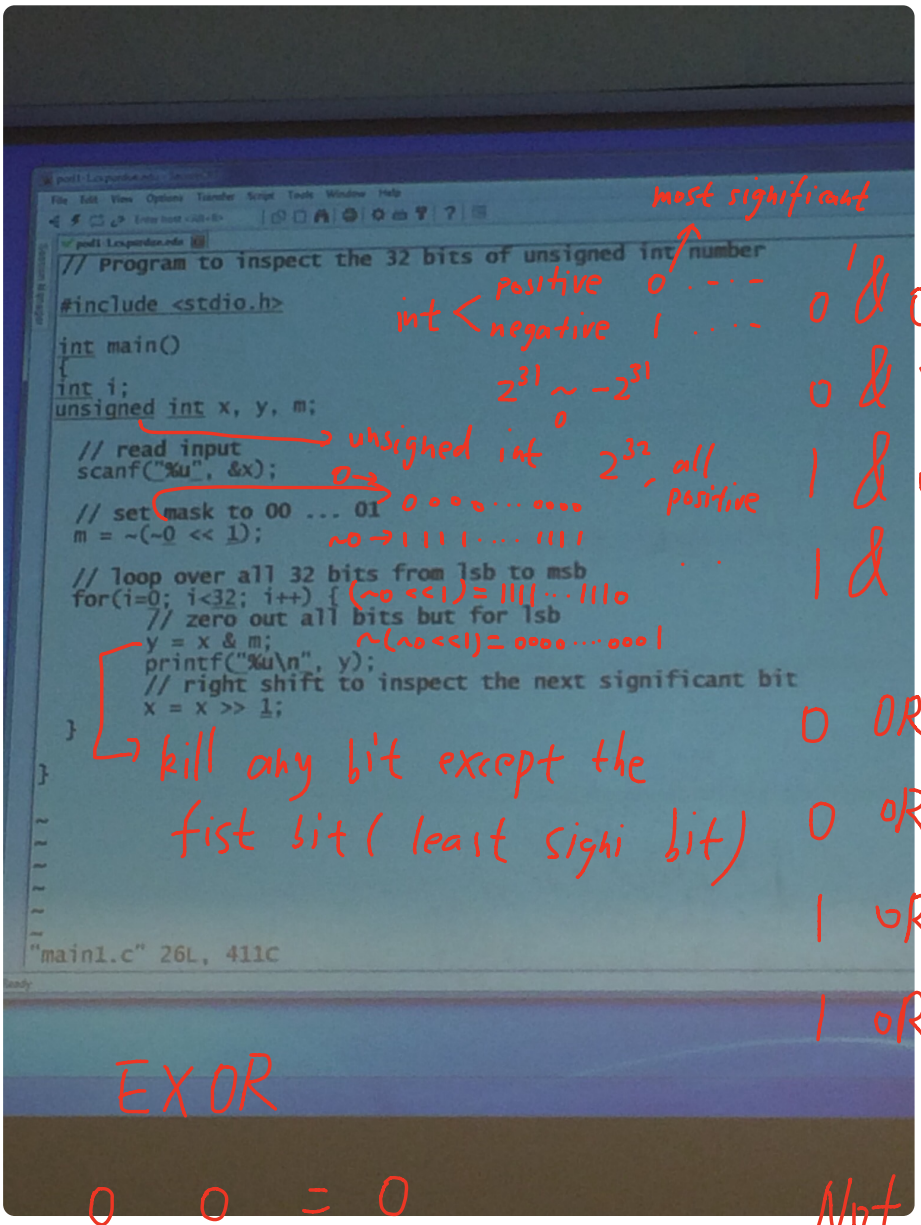
ls legacy

Address { int x → variable

Label : function name
main ( )
abc ( )
def ( )

```
abc( ){
    def( );  → jump to def address
}

abc( ){
    xyz( &def )
          ↓
       optional
}

xyz (int (*fp)( ))

 int def( );
```

Bit Processing



```
// Program to inspect the 32 bits of unsigned int number

#include <stdio.h>

int main()
{
int i;
unsigned int x, y, m;

  // read input
  scanf("%u", &x);

  // set mask to 00 ... 01
  m = ~(~0 << 1);

  // loop over all 32 bits from lsb to msb
  for(i=0; i<32; i++) {
      // zero out all bits but for lsb
      y = x & m;
      printf("%u\n", y);
      // right shift to inspect the next significant bit
      x = x >> 1;
  }
}
"main1.c" 26L, 411C
```

most significant

int < positive 0 ...
negative 1 ...

$2^{31} \sim -2^{31}$

0

unsigned int 2^{32}, all positive

0 → 0000 ... 0000

~0 → 1111 ... 1111

(~0 << 1) = 1111 ... 1110

~(~0 << 1) = 0000 ... 0001

kill any bit except the fist bit (least sighi bit)

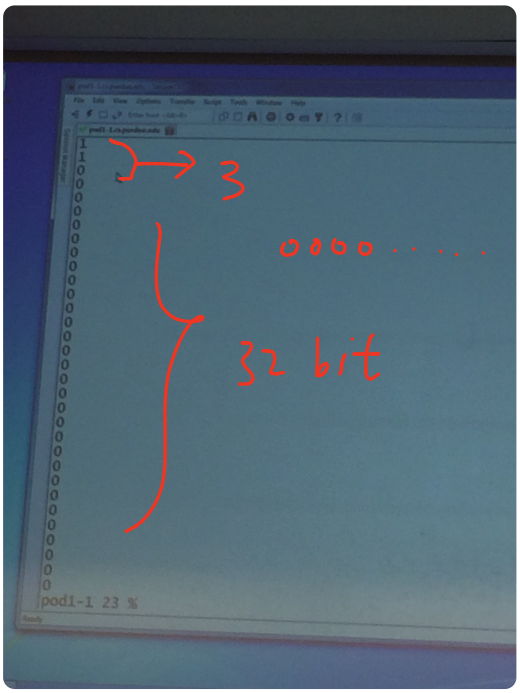
0 & 0 = 0
0 & 1 = 0
1 & 0 = 0
1 & 1 = 1

0 OR 0 = 0
0 OR 1 = 1
1 OR 0 = 1
1 OR 1 = 1

EXOR

0 0 = 0
0 1 = 1
1 0 = 1
1 1 = 0

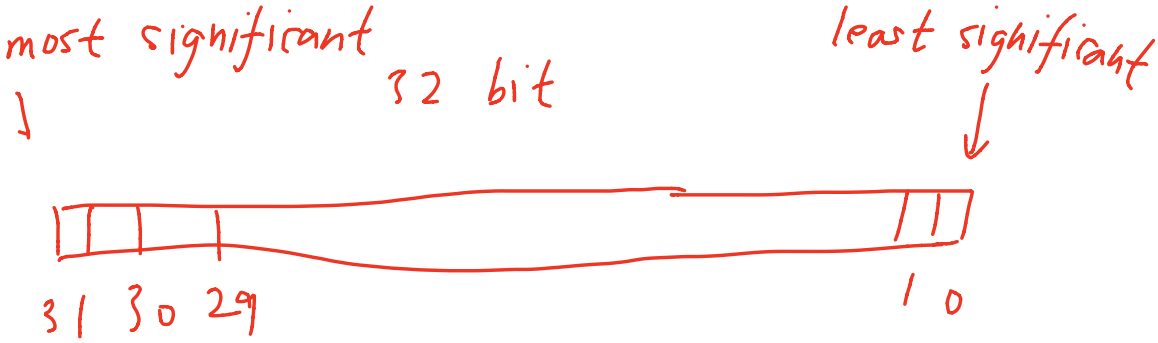
Not 0 = 1
Not 1 = 0


11000000000000000000000000000000
3
0000 0011
32 bit
podi-1 23 %

Input data int x ;

mask int m ;

output data int y ;



$X_0 = ?$

x :	X_{31}	X_{30}	X_{29}	.....	X_2	X_1	X_0
	And						
m	0	0	0		0	0	1
y :	0	0	0		0	0	X_0

x_0 And 1 is what X_0 is left

what's X_1 = ?

1. shift X to right for 1 bit

X_0 will lost for ever

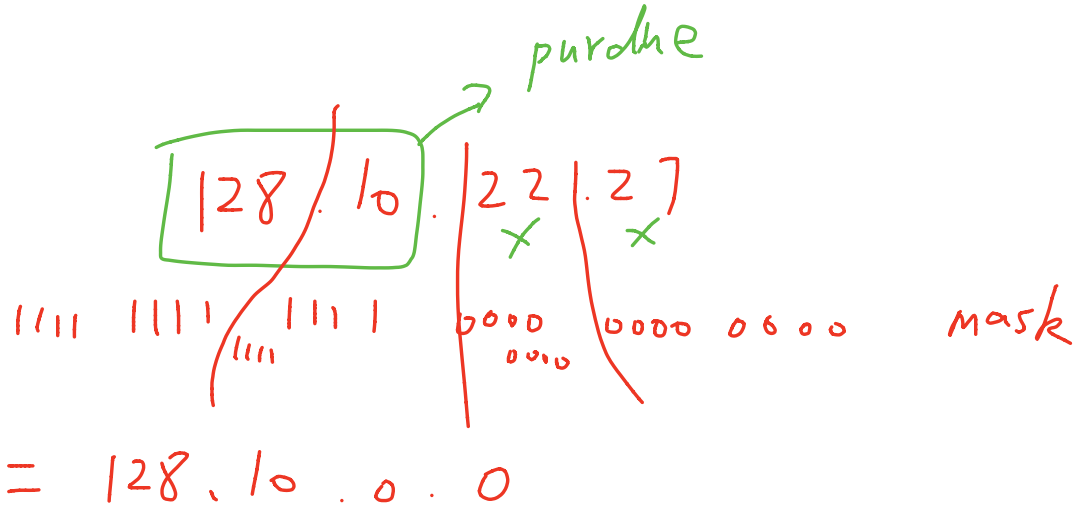
$Z \rightarrow 0\ X_{31}\ X_{30}\ X_{29} \cdots X_3\ X_2\ X_1$

mask: 0 0 0 0 ... 0 0 1

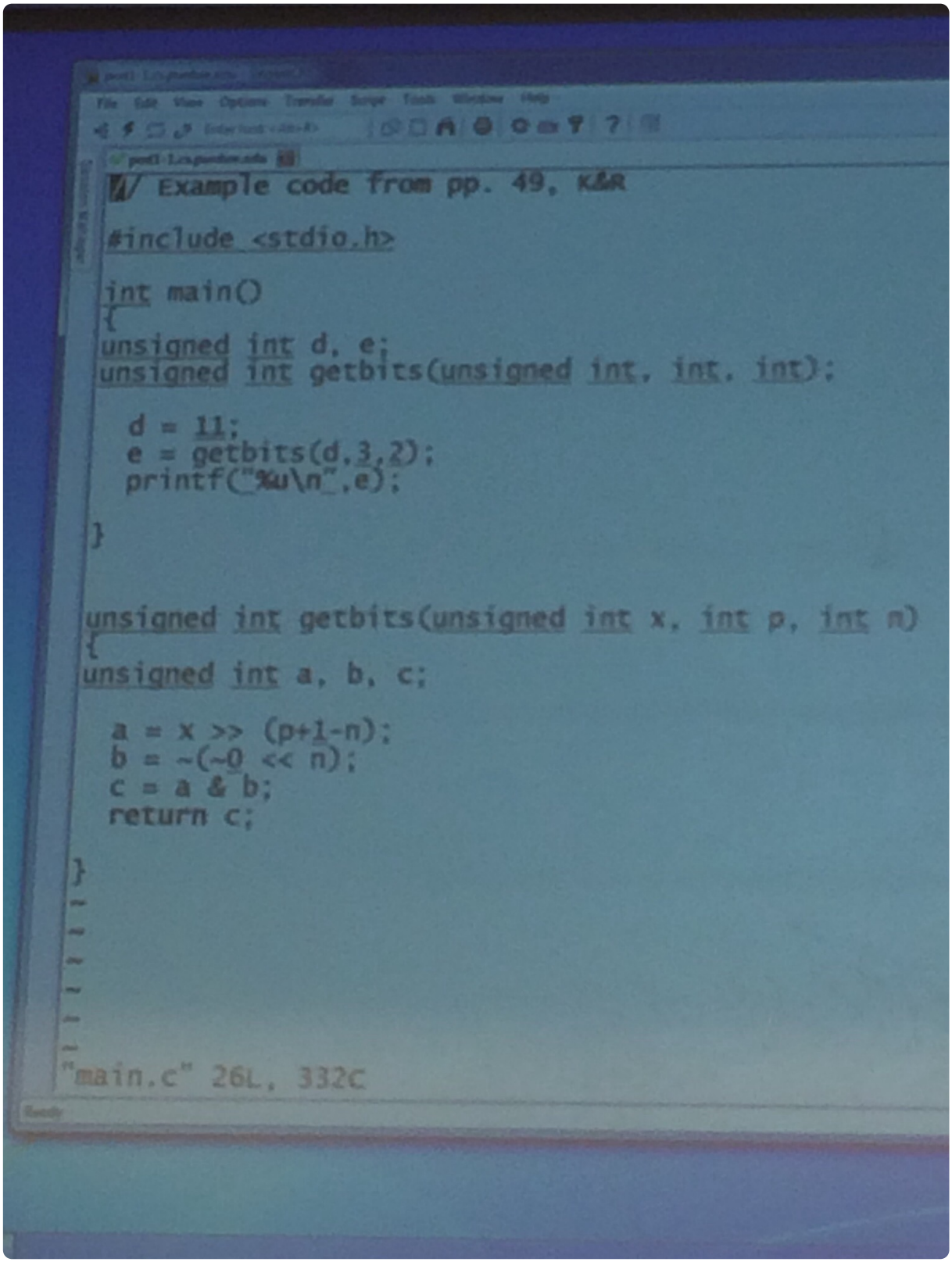
y: 0 0 0 0 0 0 X_1

→ with the same mask

$Z = X \gg 1$



= 128.10.0.0



```
// Example code from pp. 49, K&R

#include <stdio.h>

int main()
{
unsigned int d, e;
unsigned int getbits(unsigned int, int, int);

  d = 11;
  e = getbits(d,3,2);
  printf("%u\n",e);

}


unsigned int getbits(unsigned int x, int p, int n)
{
unsigned int a, b, c;

  a = x >> (p+1-n);
  b = ~(~0 << n);
  c = a & b;
  return c;

}
~
~
~
~
~
~
"main.c" 26L, 332C
```