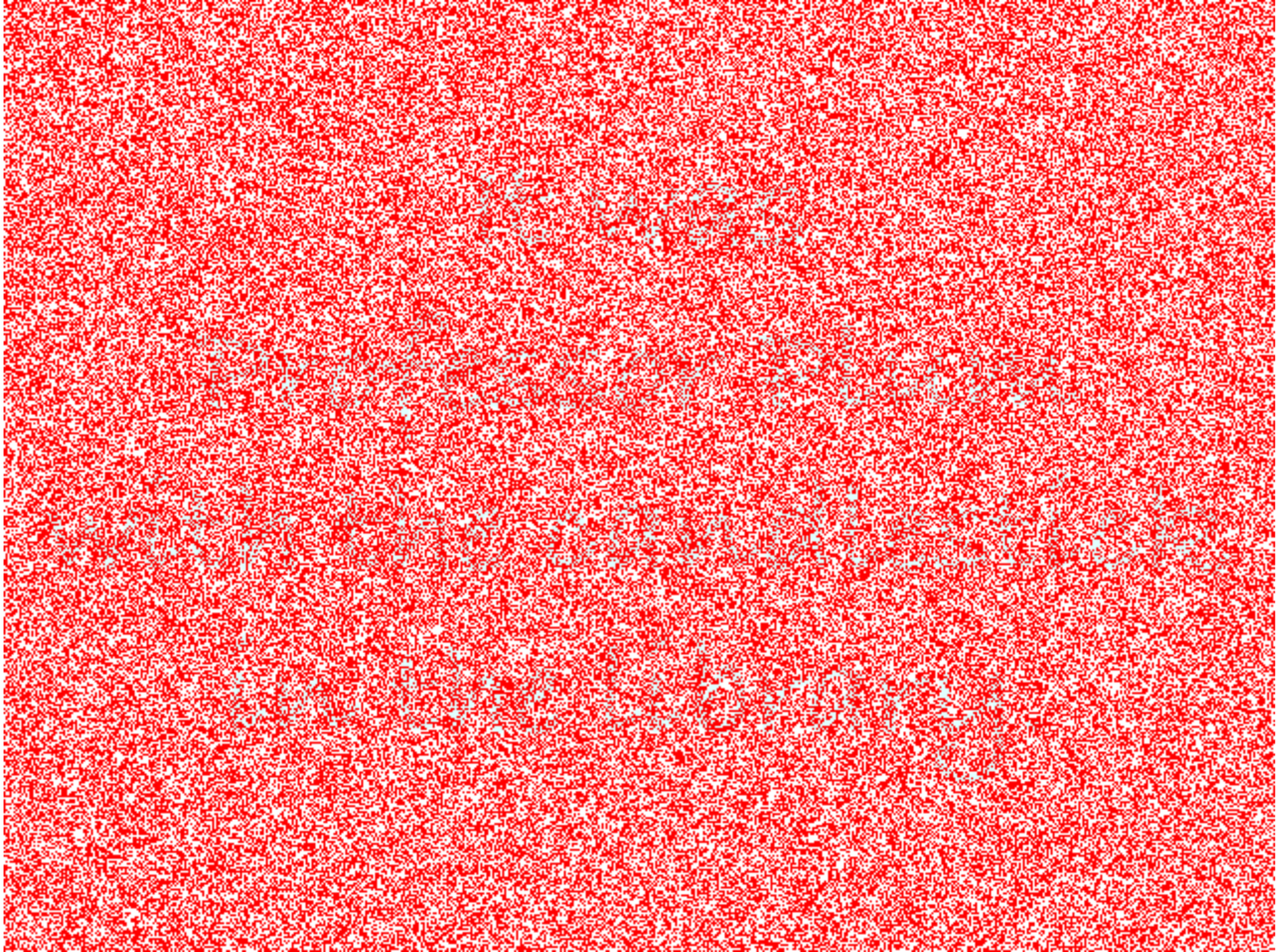
CS50



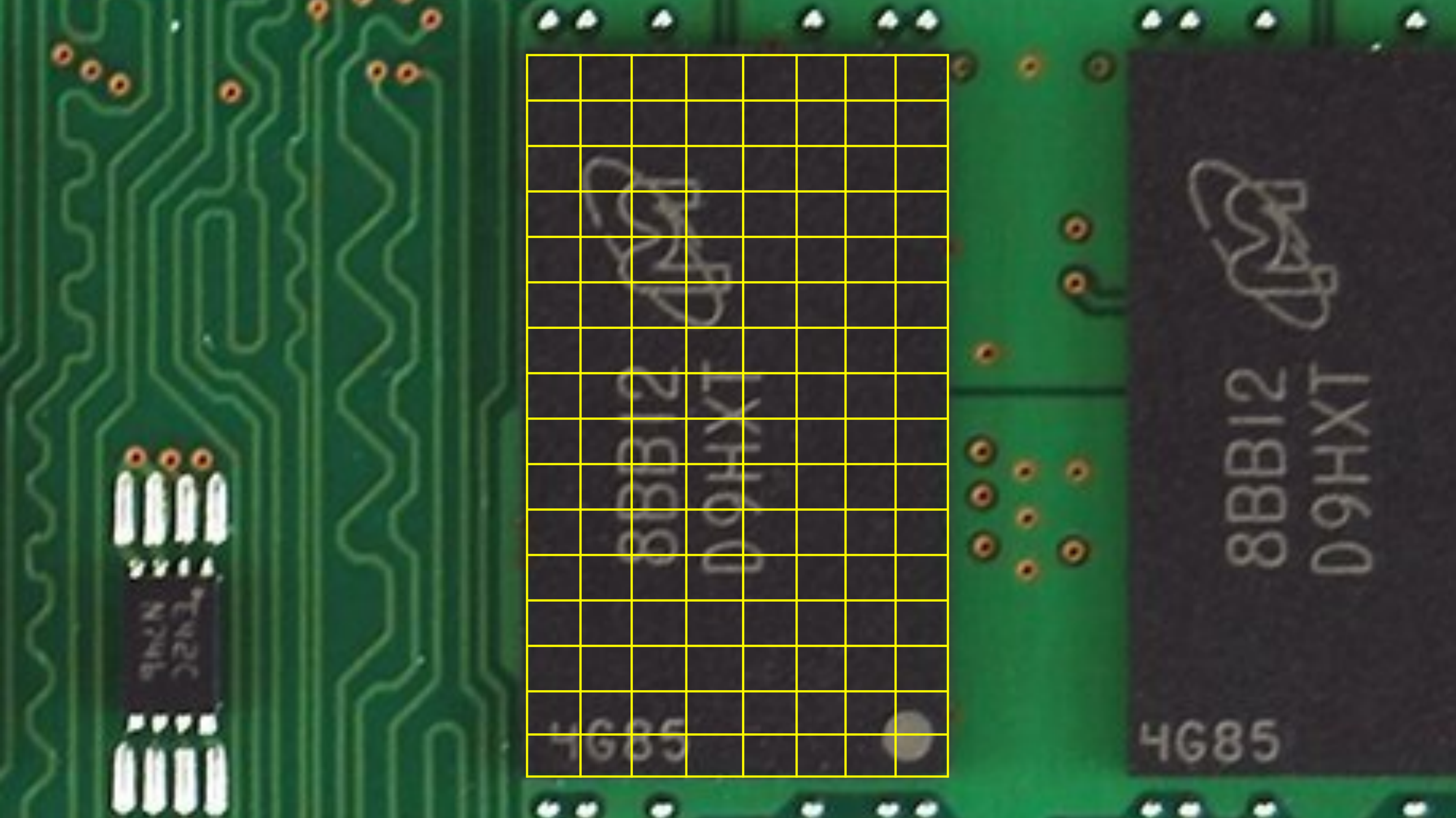
offset	type	name	
0	WORD	bfType	BITMAPFILEHEADER
2	DWORD	bfSize	
6	WORD	bfReserved1	
8	WORD	bfReserved2	
10	DWORD	bfOffBits	
14	DWORD	biSize	BITMAPINFOHEADER
18	LONG	biWidth	
22	LONG	biHeight	
26	WORD	biPlanes	
28	WORD	biBitCount	
30	DWORD	biCompression	
34	DWORD	biSizeImage	
38	LONG	biXPelsPerMeter	
42	LONG	biYPelsPerMeter	
46	DWORD	biClrUsed	
50	DWORD	biClrImportant	
54	BYTE	rgbtBlue	RGBTRIPLE
55	BYTE	rgbtGreen	
56	BYTE	rgbtRed	
57	BYTE	rgbtBlue	RGBTRIPLE
58	BYTE	rgbtGreen	
59	BYTE	rgbtRed	

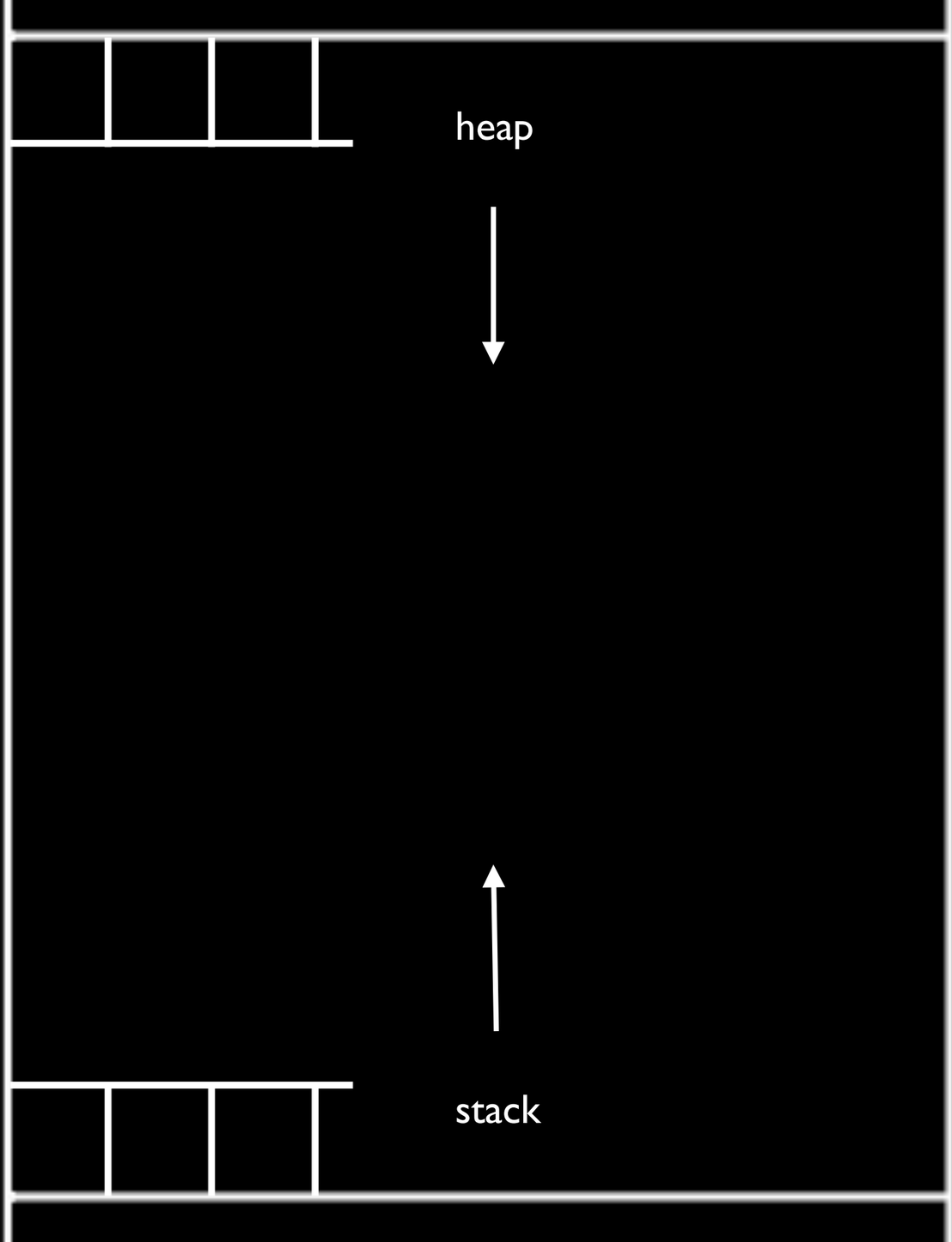
...

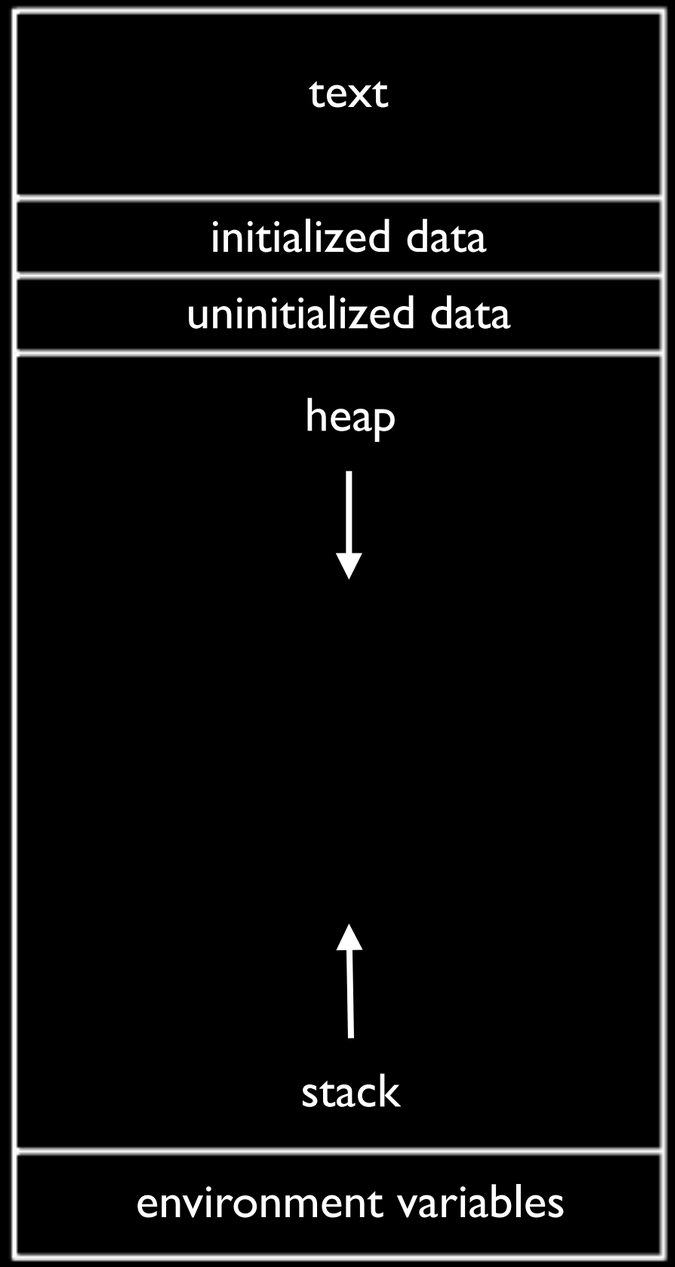
243	BYTE	rgbtBlue	RGBTRIPLE
244	BYTE	rgbtGreen	
245	BYTE	rgbtRed	

```
typedef struct
{
    string name;
    string dorm;
}
student;
```


8BBI2
D9HXT
4G85
8BBI2
D9HXT
4G85


8BB12
D9HXT
4G85


heap
stack


text
initialized data
uninitialized data
heap
stack
environment variables

```
void swap(int a, int b)
{
    int tmp = a;
    a = b;
    b = tmp;
}
```

```
void swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
```

malloc

free

malloc, calloc, realloc

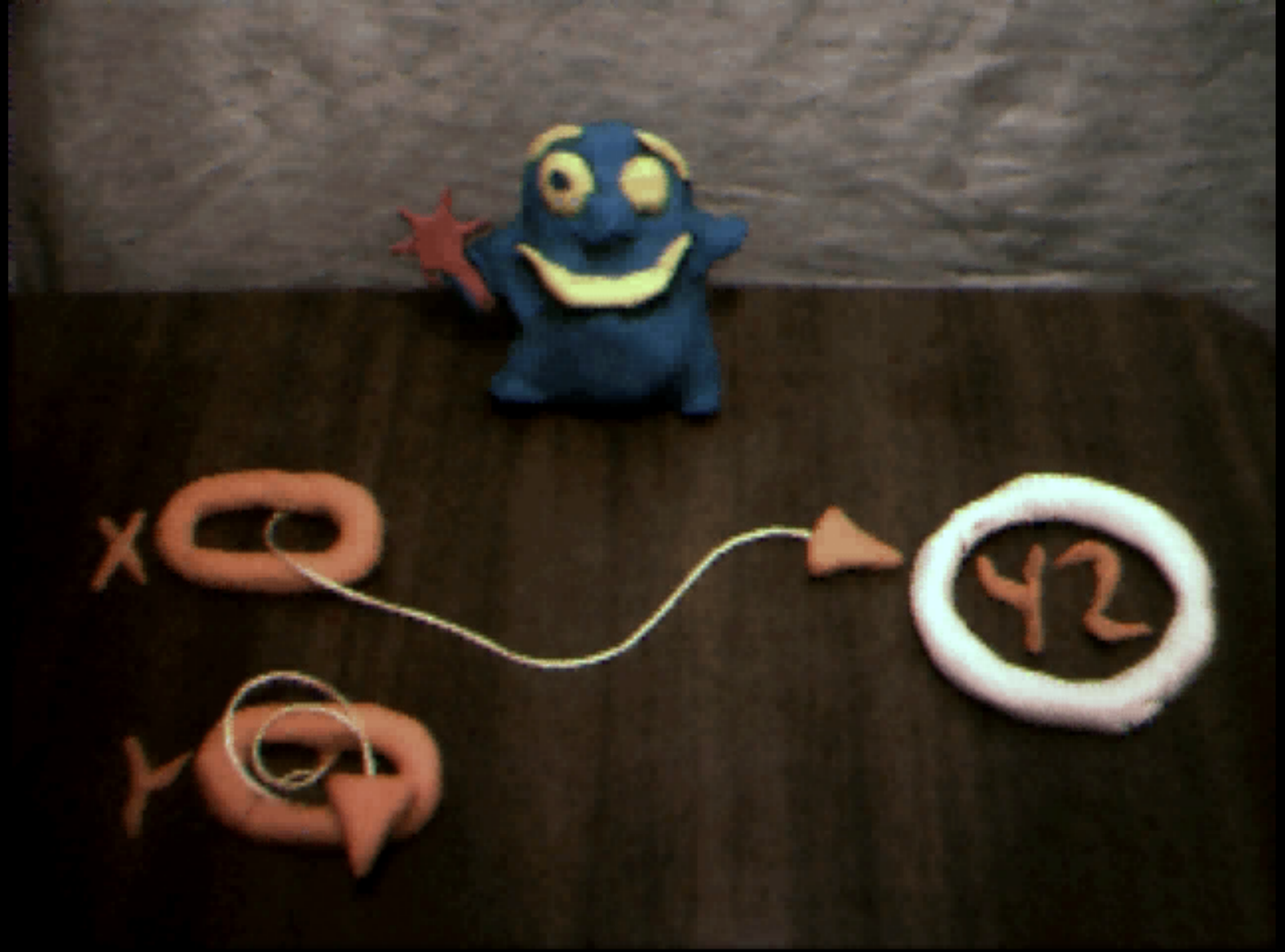
free


POINTER
FUN

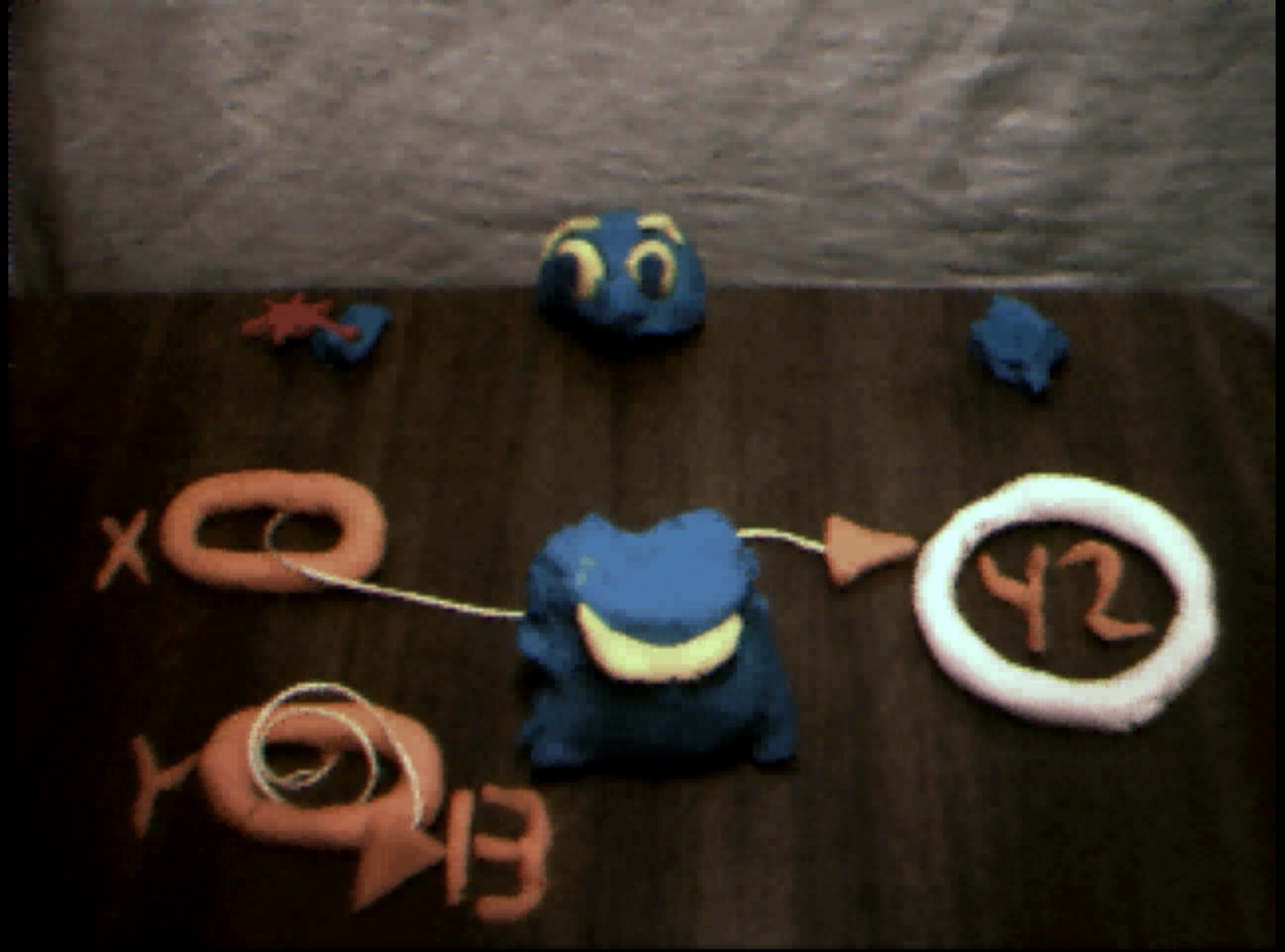
```
int main(void)
{
    int *x;
    int *y;

    x = malloc(sizeof(int));

    *x = 42;

    *y = 13;

    y = x;

    *y = 13;
}
```


```
int* x;
int* y;
```

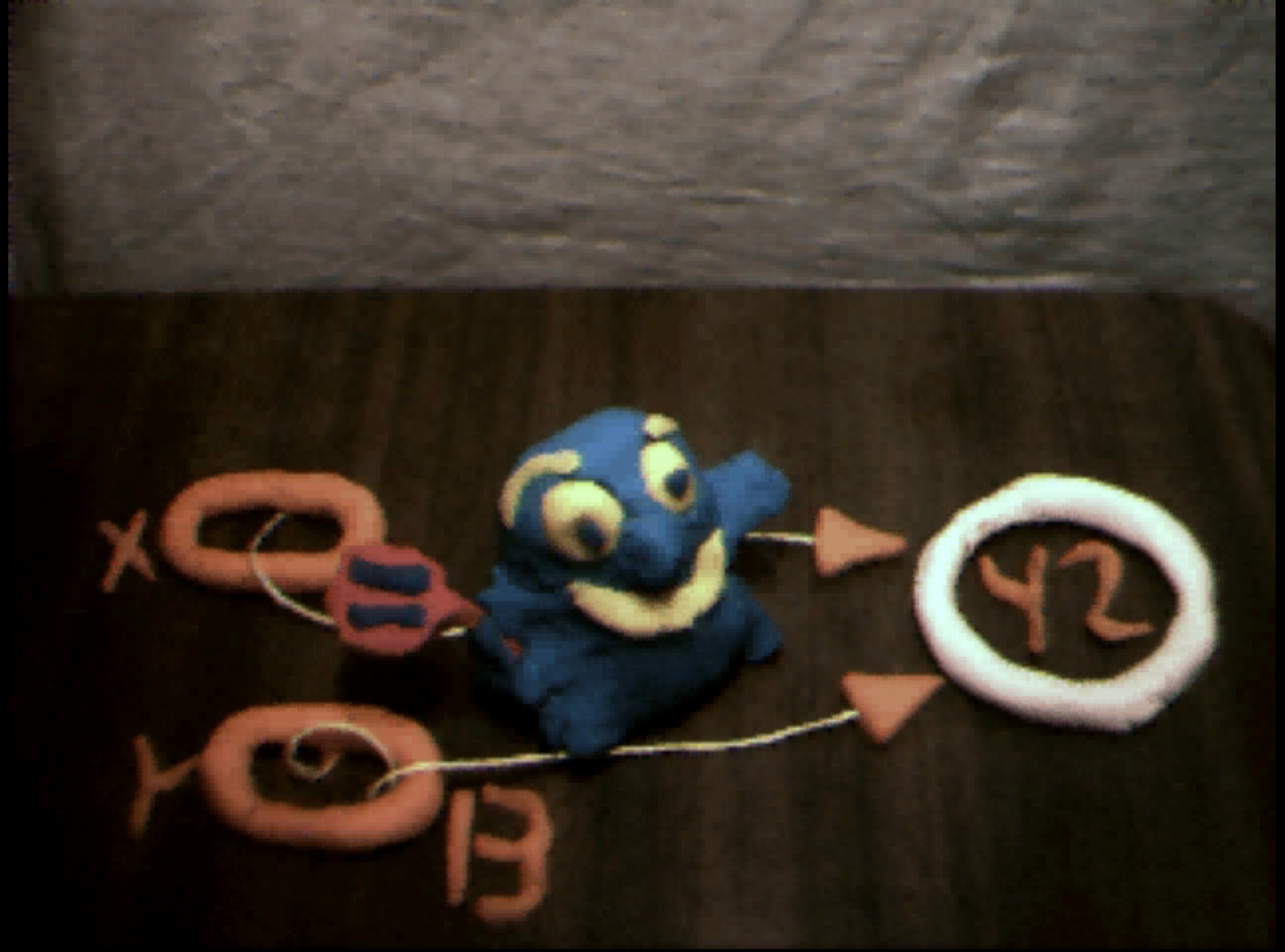

```
x = malloc(sizeof(int));
```



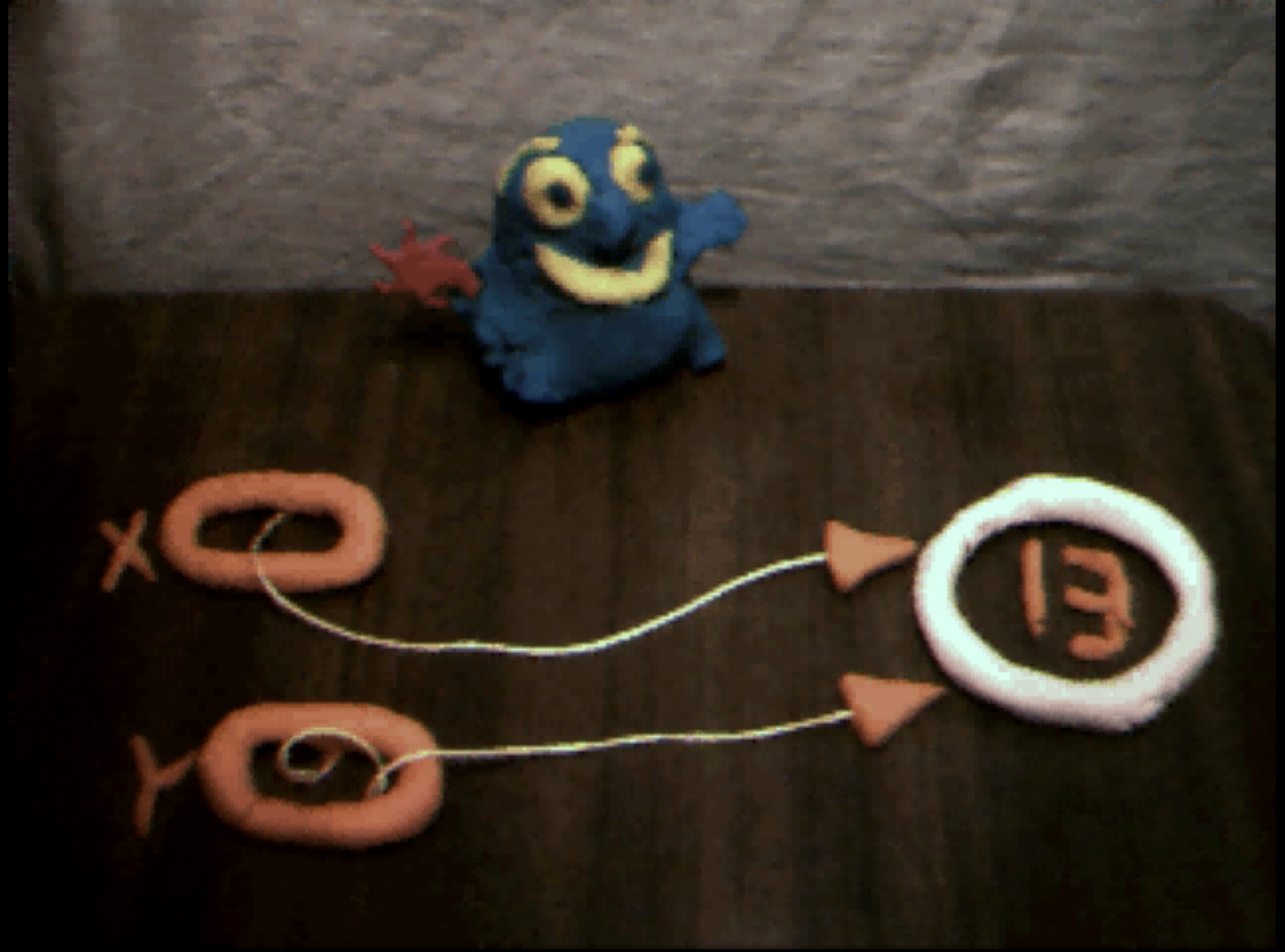

```
*x = 42;
```




```
*y = 13;
```




```
y = x;
```




```
*y = 13;
```


stackoverflow

stack overflow

heap overflow

buffer overflow

```
#include <string.h>

void foo(char *bar)
{
    char c[12];
    memcpy(c, bar, strlen(bar));
}

int main(int argc, char *argv[])
{
    foo(argv[1]);
}
```

adapted from wikipedia.org/wiki/Stack_buffer_overflow

```
#include <string.h>

void foo(char *bar)
{
    char c[12];
    memcpy(c, bar, strlen(bar));
}

int main(int argc, char *argv[])
{
    foo(argv[1]);
}
```

adapted from wikipedia.org/wiki/Stack_buffer_overflow

```
#include <string.h>

void foo(char *bar)
{
    char c[12];
    memcpy(c, bar, strlen(bar));
}

int main(int argc, char *argv[])
{
    foo(argv[1]);
}
```

adapted from wikipedia.org/wiki/Stack_buffer_overflow

```
#include <string.h>

void foo(char *bar)
{
    char c[12];
    memcpy(c, bar, strlen(bar));
}

int main(int argc, char *argv[])
{
    foo(argv[1]);
}
```

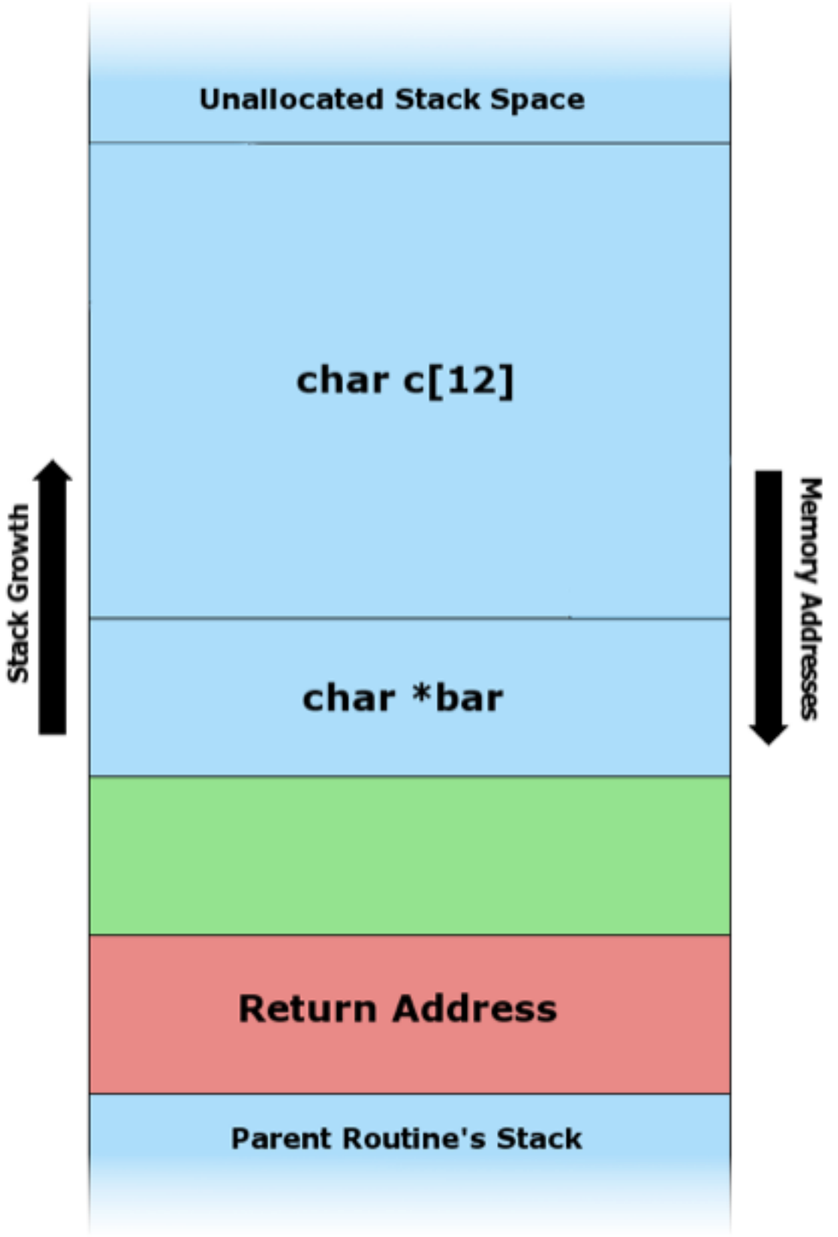
adapted from wikipedia.org/wiki/Stack_buffer_overflow

```
#include <string.h>

void foo(char *bar)
{
    char c[12];
    memcpy(c, bar, strlen(bar));
}

int main(int argc, char *argv[])
{
    foo(argv[1]);
}
```

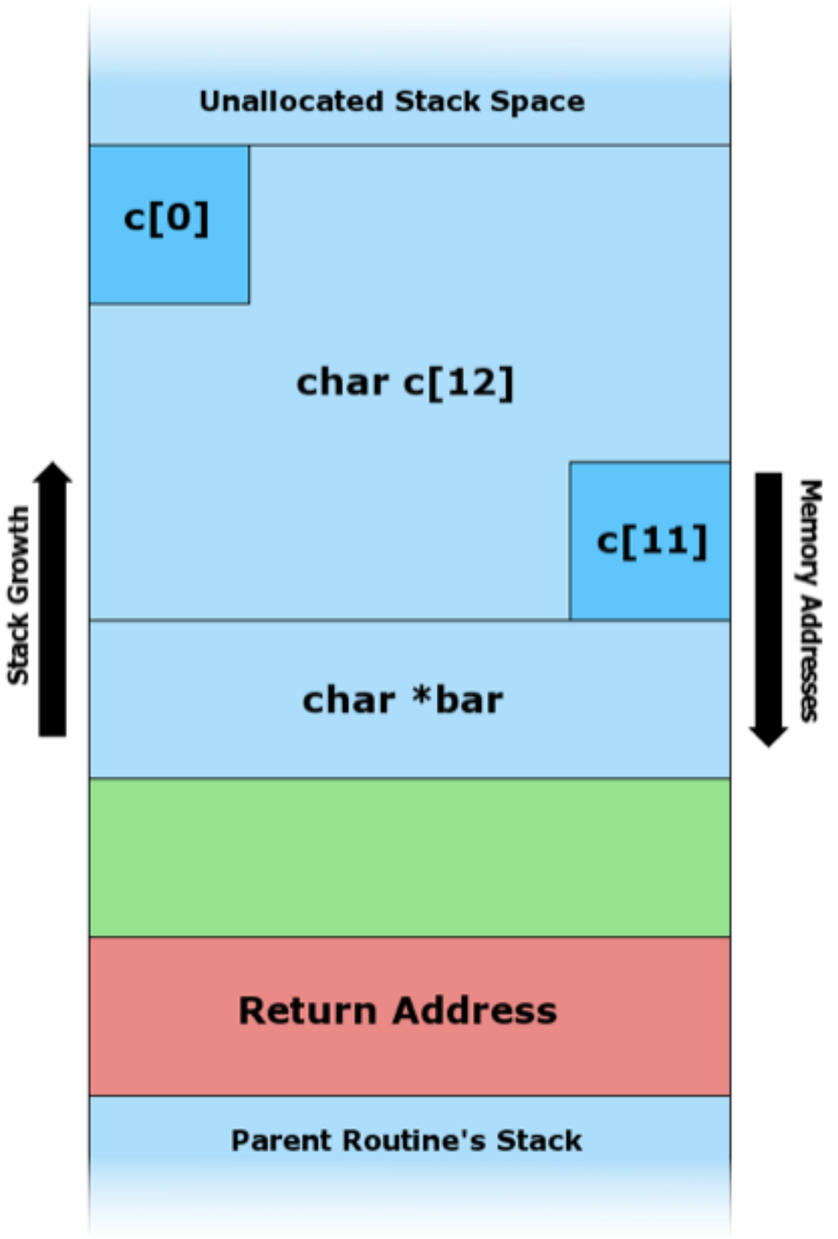
adapted from wikipedia.org/wiki/Stack_buffer_overflow

```c
#include <string.h>

void foo(char *bar)
{
    char c[12];
    memcpy(c, bar, strlen(bar));
}

int main(int argc, char *argv[])
{
    foo(argv[1]);
}
```

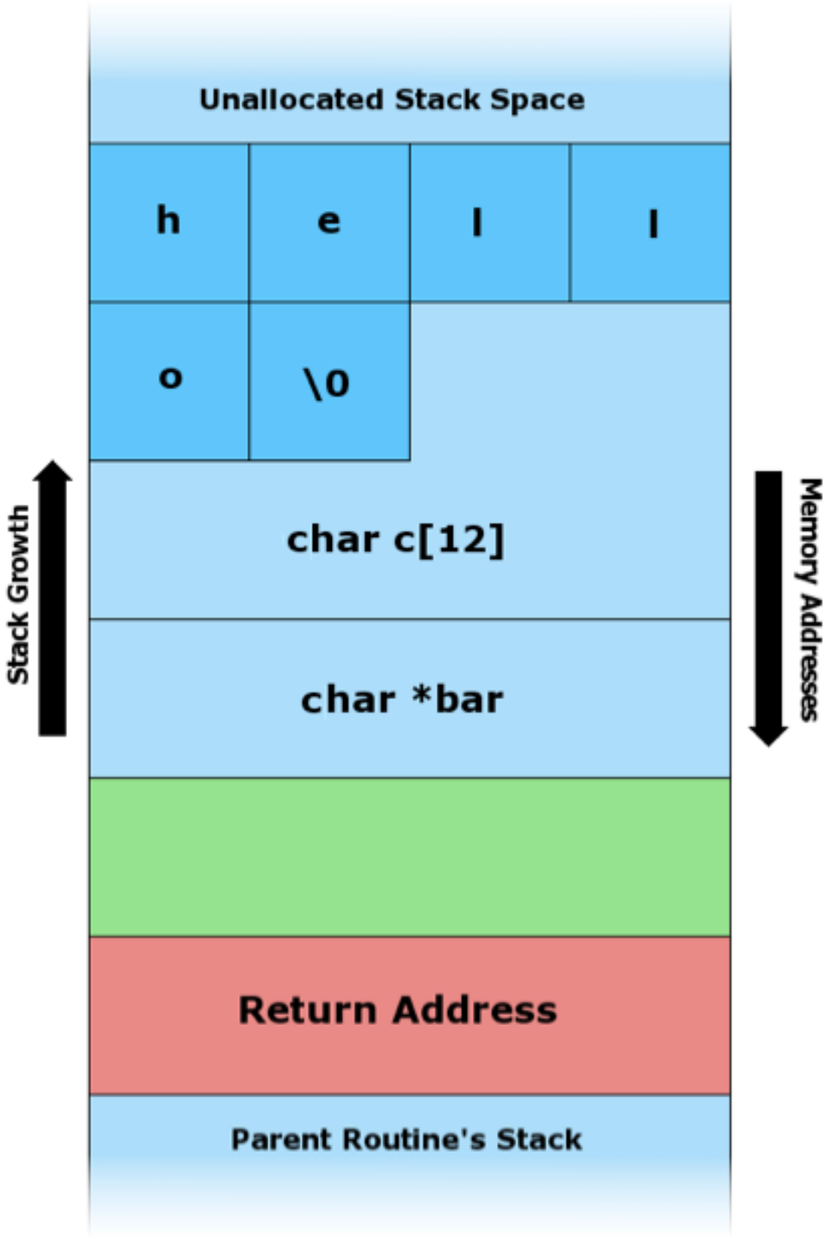
adapted from wikipedia.org/wiki/Stack_buffer_overflow



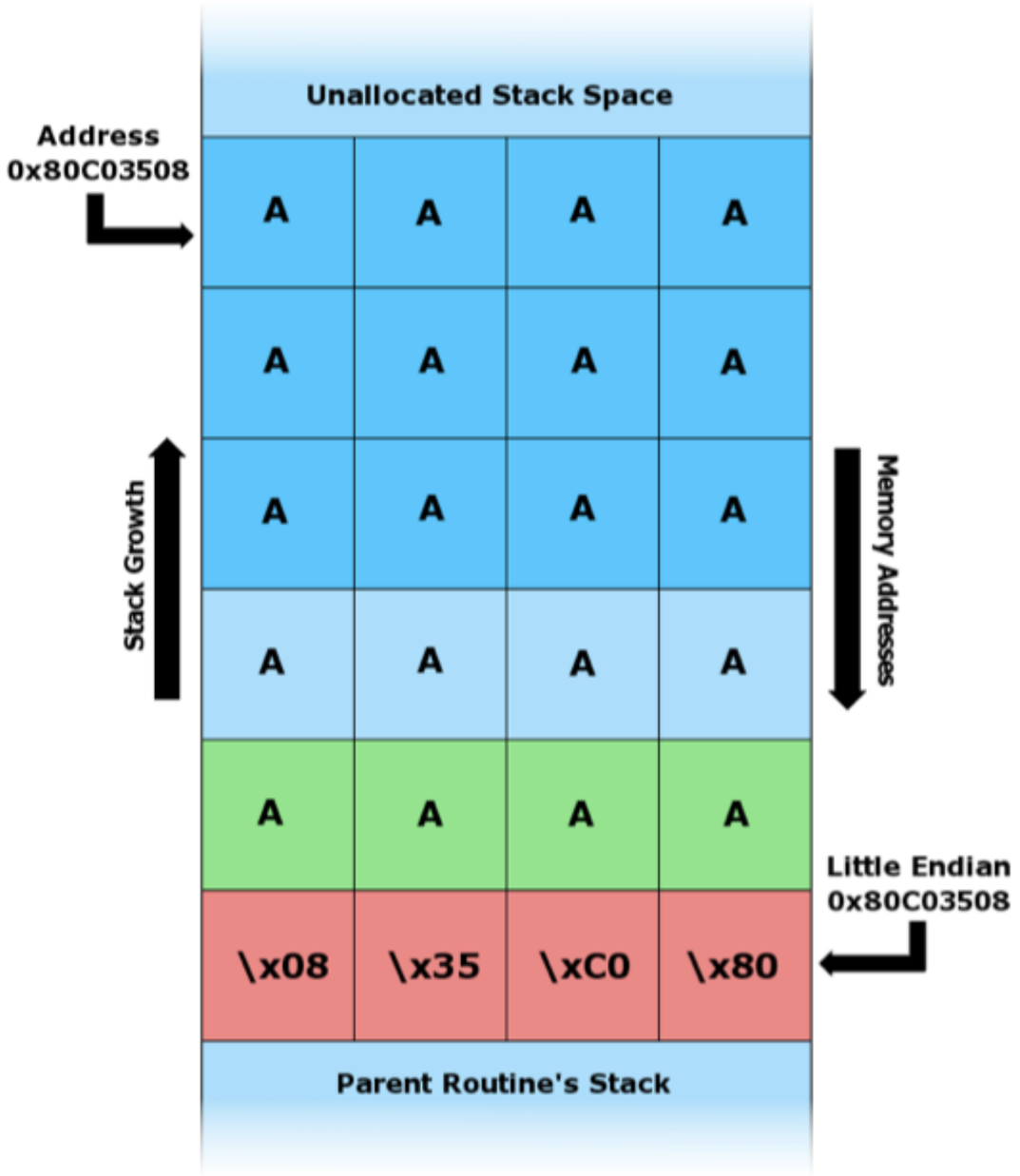
adapted from wikipedia.org/wiki/Stack_buffer_overflow



adapted from wikipedia.org/wiki/Stack_buffer_overflow



adapted from wikipedia.org/wiki/Stack_buffer_overflow



adapted from wikipedia.org/wiki/Stack_buffer_overflow

valgrind

255 216 255

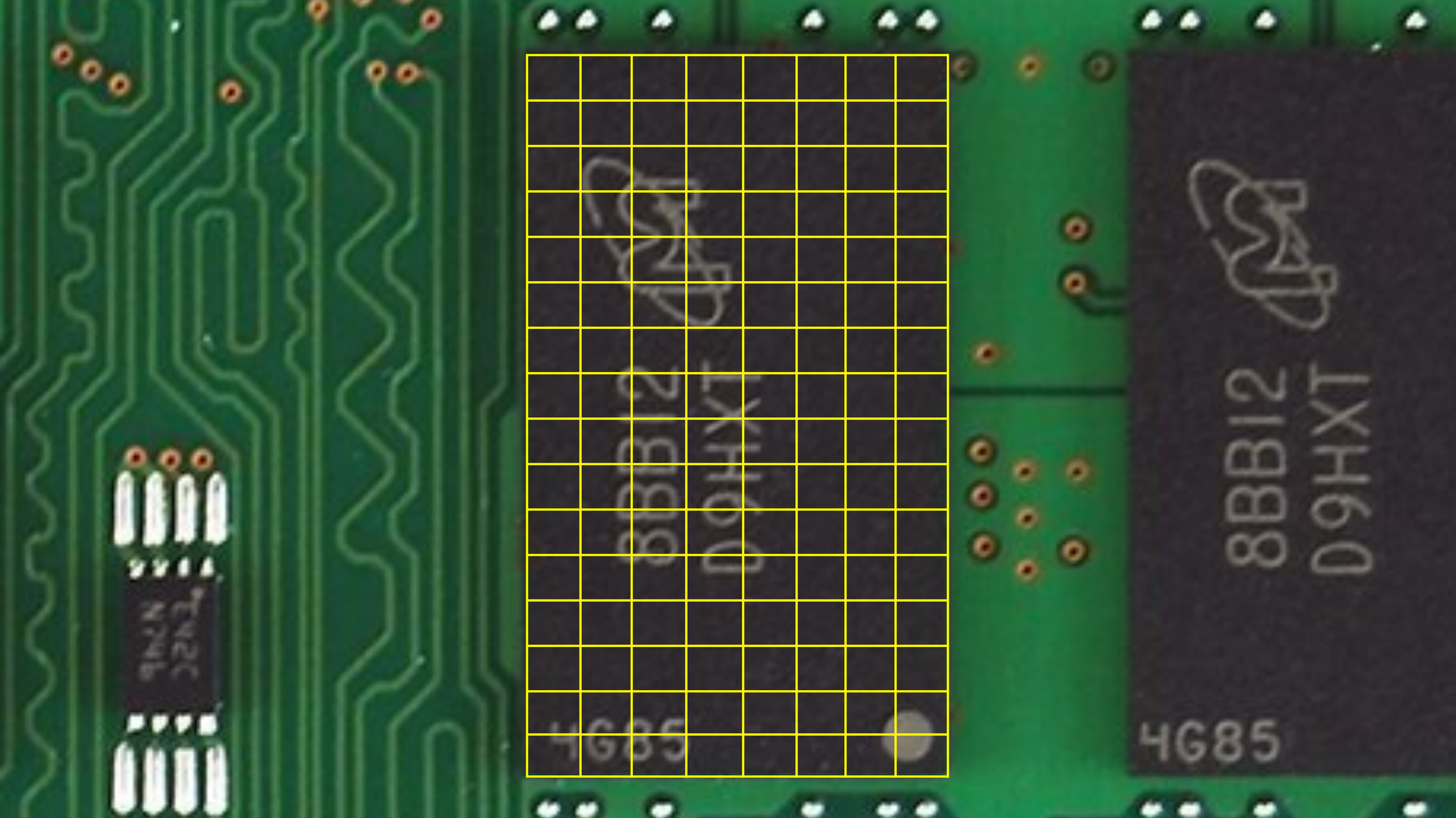
0xff 0xd8 0xff

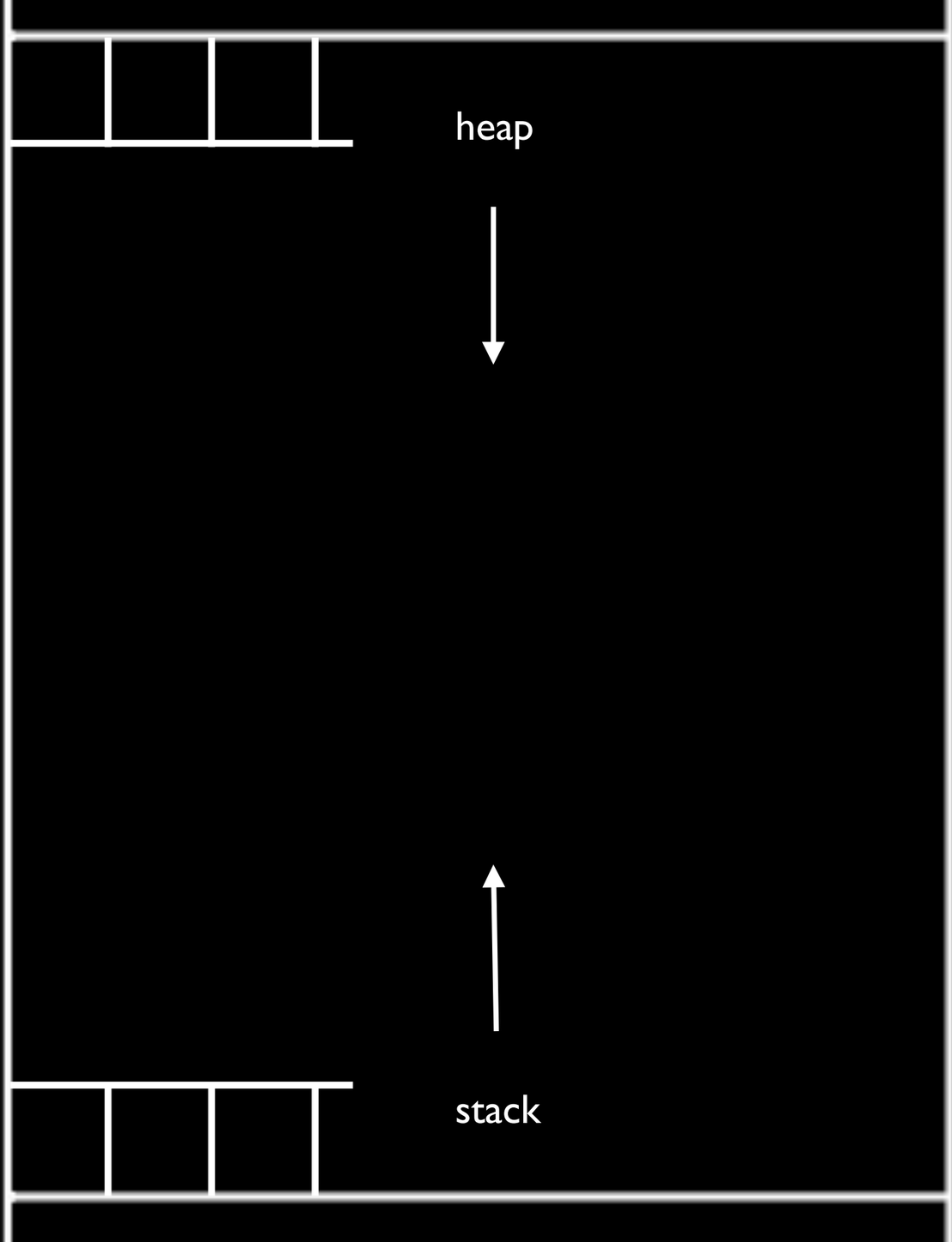
http://xkcd.com/138/

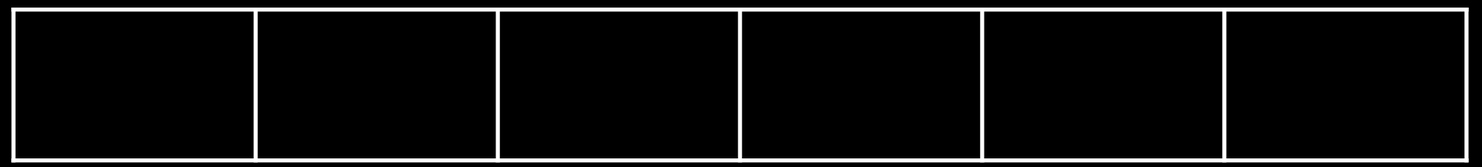
debug50

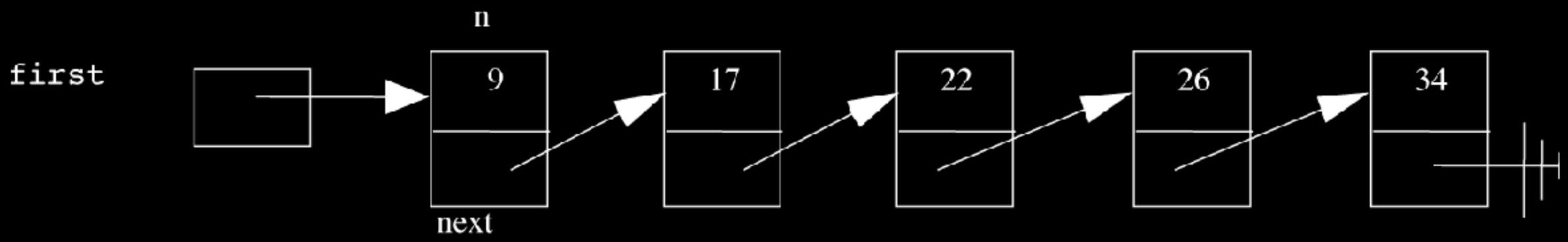
ddb50


ddb
50

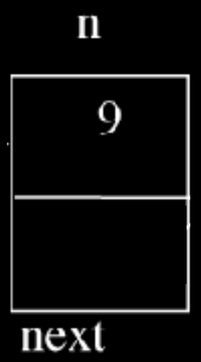

8BB12
D9HXT
4G85


heap
stack



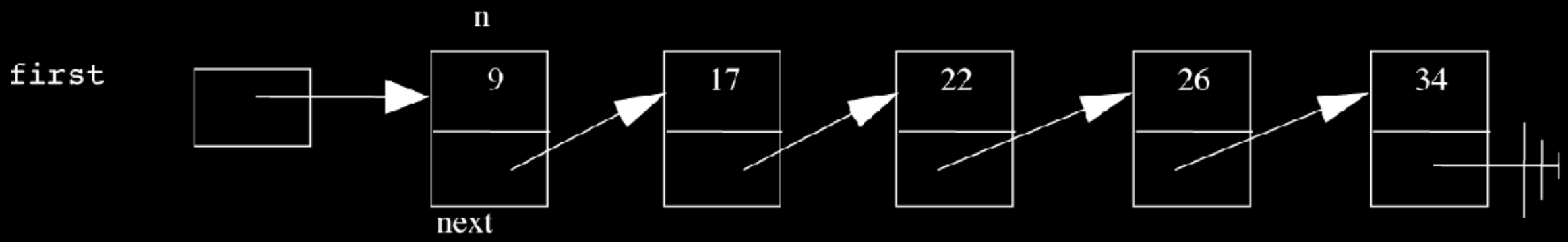


https://cs.calvin.edu/activities/books/c++/ds/1e/


n
9
next

```
typedef struct
{
    string name;
    string dorm;
}
student;
```

```
typedef struct node
{
    int n;
    struct node *next;
}
node;
```



https://cs.calvin.edu/activities/books/c++/ds/1e/


THANKS FOR YOUR

```
push

pop

...
```

```
typedef struct
{
    int numbers[CAPACITY];
    int size;
}
stack;
```

```
typedef struct
{
    int *numbers;
    int size;
}
stack;
```

enqueue

dequeue

...

```
typedef struct
{
    int front;
    int numbers[CAPACITY];
    int size;
}
queue;
```

```
typedef struct
{
    int front;
    int *numbers;
    int size;
}
queue;
```

Jack Learns the Facts About Queues and Stacks

tree

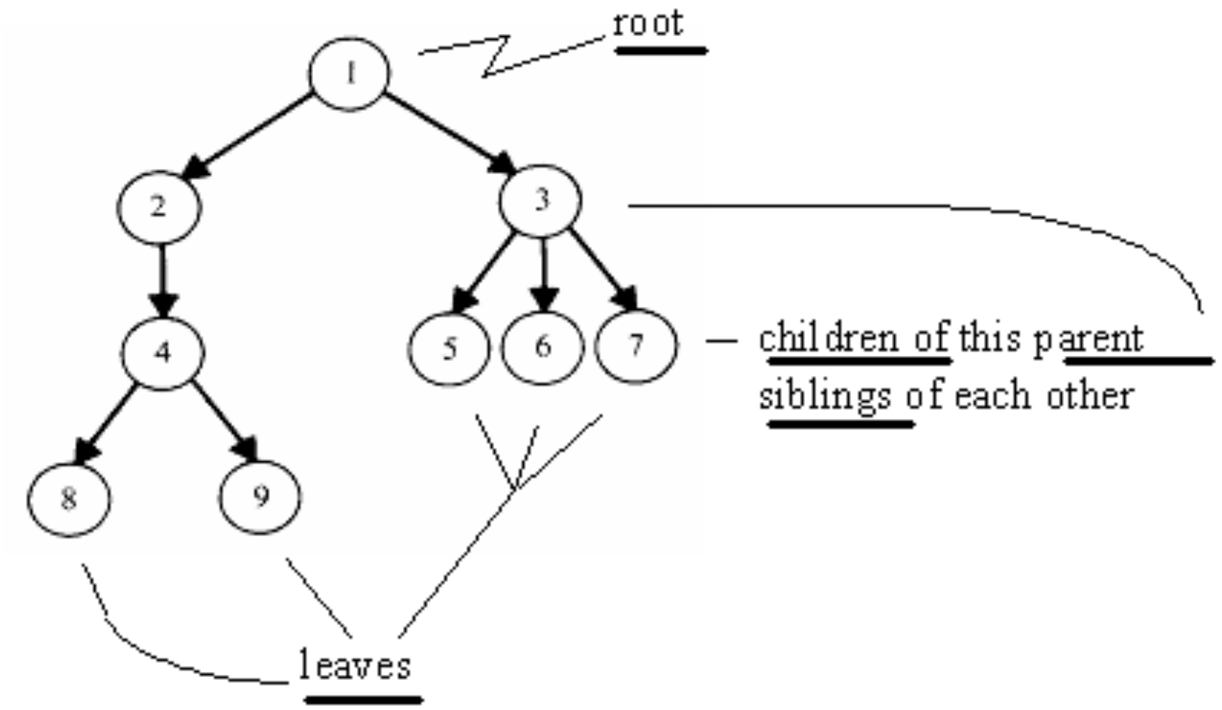


Figure by Larry Nyhoff.

22 33 44 55 66 77 88


22
33
44
55
66
77
88

binary search tree

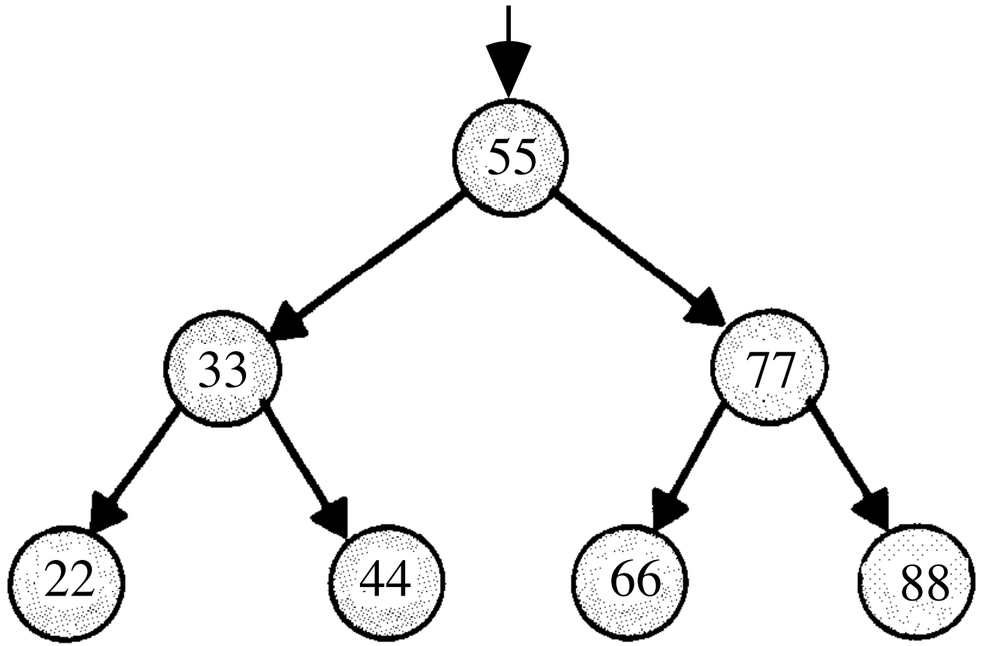


Figure from http://cs.calvin.edu/books/c++/ds/1e/.

```
typedef struct node
{
    int n;
    struct node *left;
    struct node *right;
}
node;
```

```
bool search(int n, node *tree)
{




}
```

```
bool search(int n, node *tree)
{
    if (tree == NULL)
    {
        return false;
    }









}
```

```
bool search(int n, node *tree)
{
    if (tree == NULL)
    {
        return false;
    }
    else if (n < tree->n)
    {
        return search(n, tree->left);
    }









}
```

```
bool search(int n, node *tree)
{
    if (tree == NULL)
    {
        return false;
    }
    else if (n < tree->n)
    {
        return search(n, tree->left);
    }
    else if (n > tree->n)
    {
        return search(n, tree->right);
    }



}
```

```
bool search(int n, node *tree)
{
    if (tree == NULL)
    {
        return false;
    }
    else if (n < tree->n)
    {
        return search(n, tree->left);
    }
    else if (n > tree->n)
    {
        return search(n, tree->right);
    }
    else
    {
        return true;
    }
}
```

$O(n^2)$

$O(n \log n)$

$O(n)$

$O(\log n)$

$O(1)$

...

$O(n^2)$

$O(n \log n)$

$O(n)$

$O(\log n)$

$O(1)$

...

$O(n^2)$

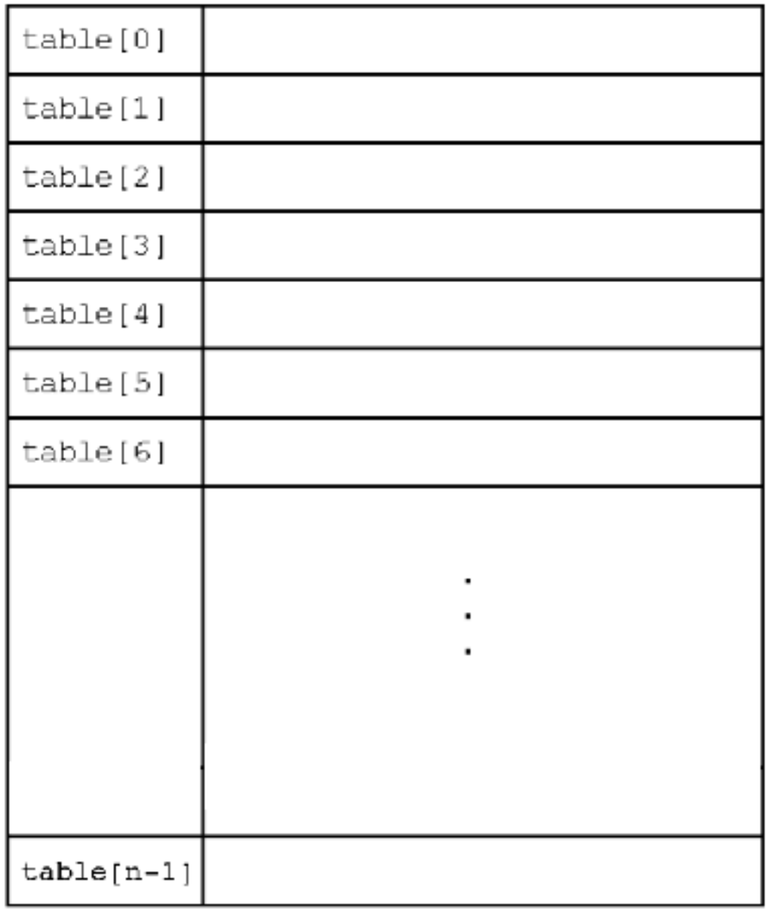
$O(n \log n)$

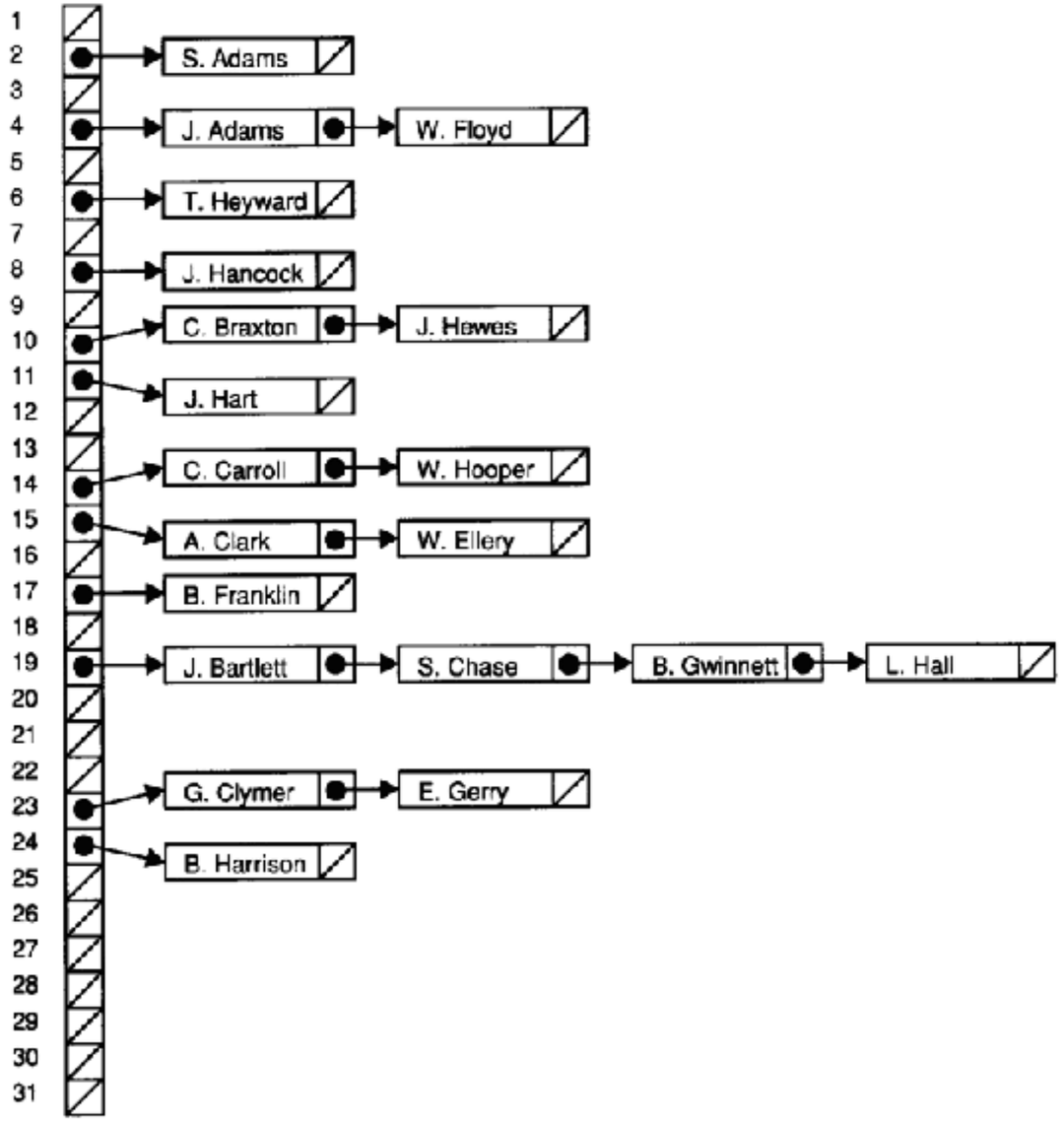
$O(n)$

$O(\log n)$

$O(1)$

...


table[0]
table[1]
table[2]
table[3]
table[4]
table[5]
table[6]
table[n-1]


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
S. Adams
J. Adams
W. Floyd
T. Heyward
J. Hancock
C. Braxton
J. Hewes
J. Hart
C. Carroll
W. Hooper
A. Clark
W. Ellery
B. Franklin
J. Bartlett
S. Chase
B. Gwinnett
L. Hall
G. Clymer
E. Gerry
B. Harrison

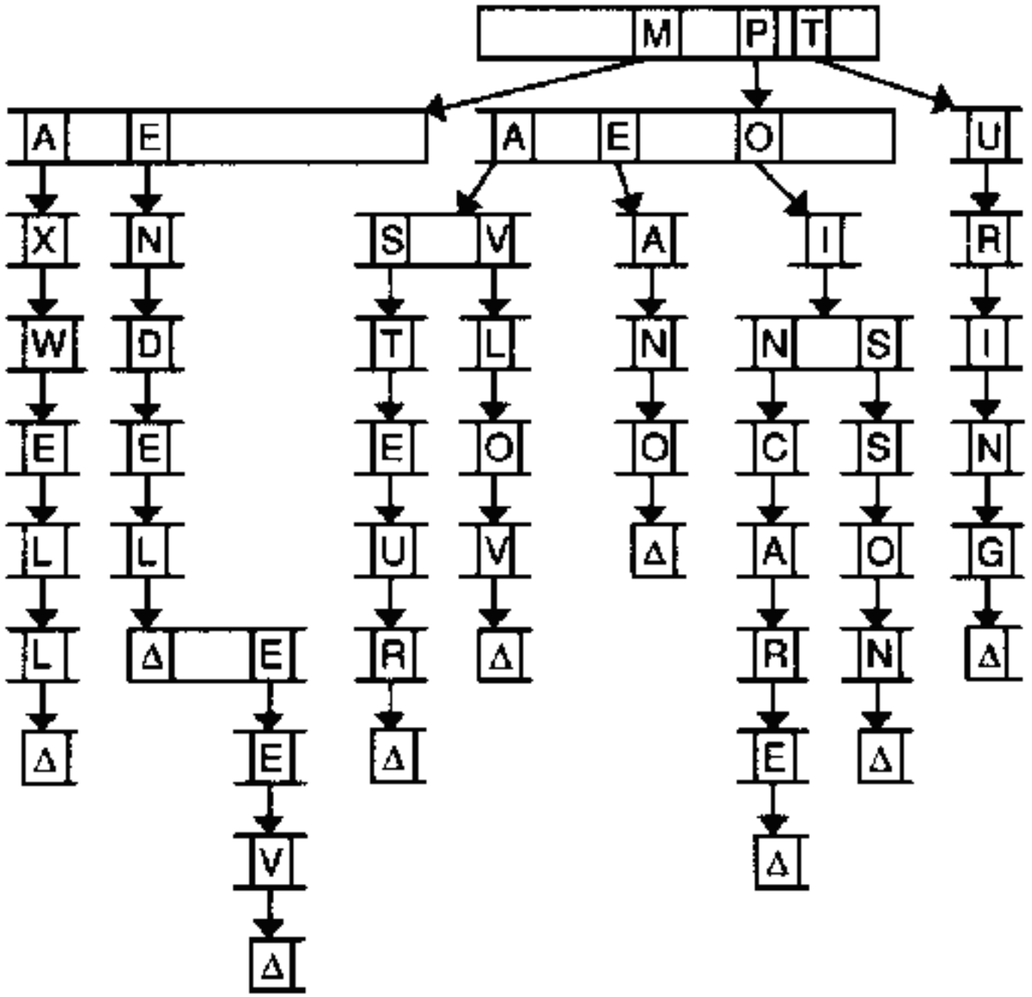
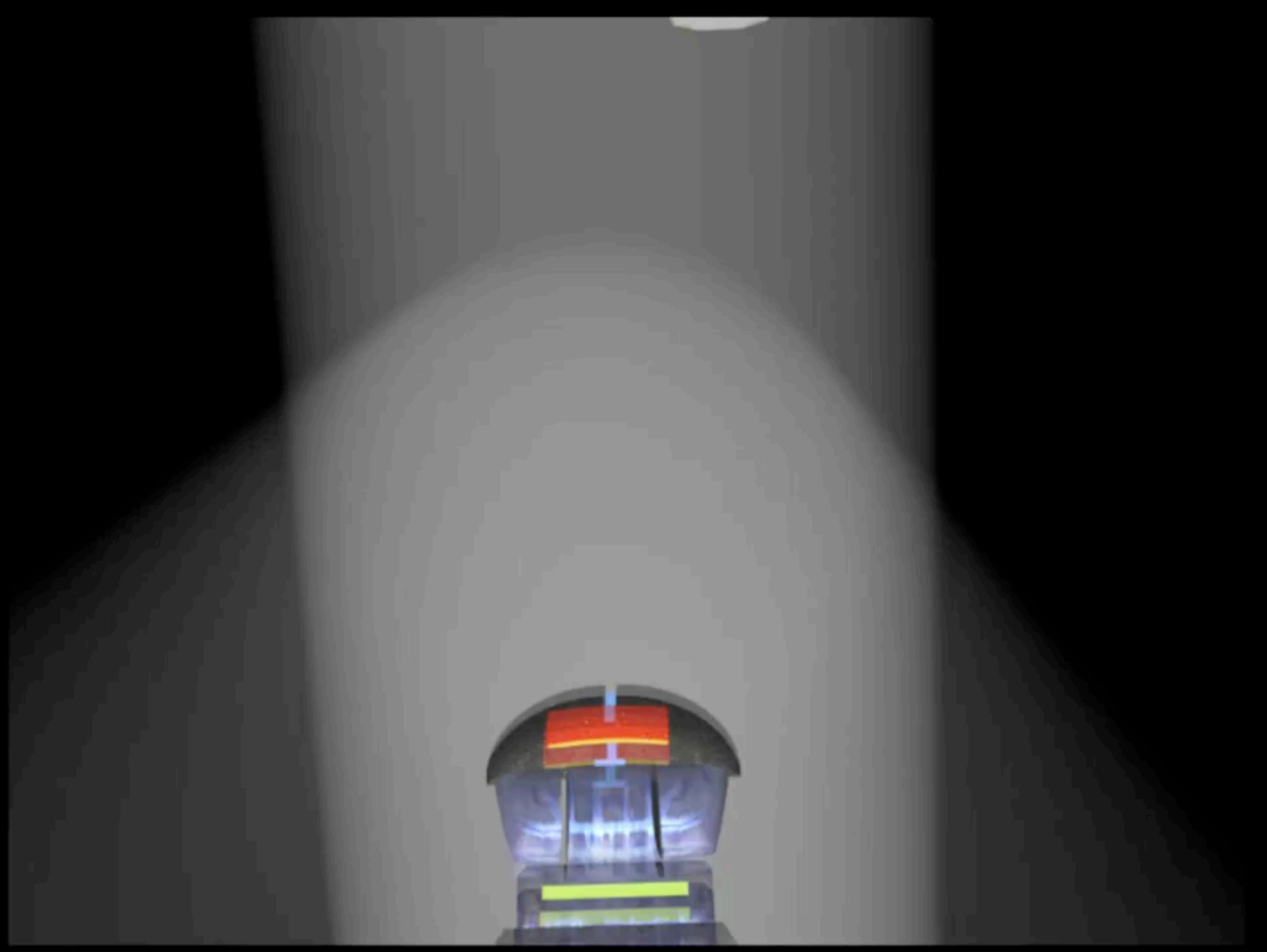


Figure from Lewis and Denenberg's Data Structures & Their Algorithms.

```
typedef struct node
{
    bool word;
    struct node *children[27];
}
node;
```



CS50