ECE 264 Spring 2023
Advanced C Programming

Aravind Machiry
Purdue University

Homework 17 & 18
Huffman Compression

yunglu@purdue.edu

HW 17 & HW 18

- HW17: Rebuild the Huffman compression tree from post-order traversal and print the code book
- HW18: Use the code book to compress the end of a file and save the bits (need bitwise operations). Only the end (excerpt) of a file is used so that it is shorter and easier to debug.

Remember!!

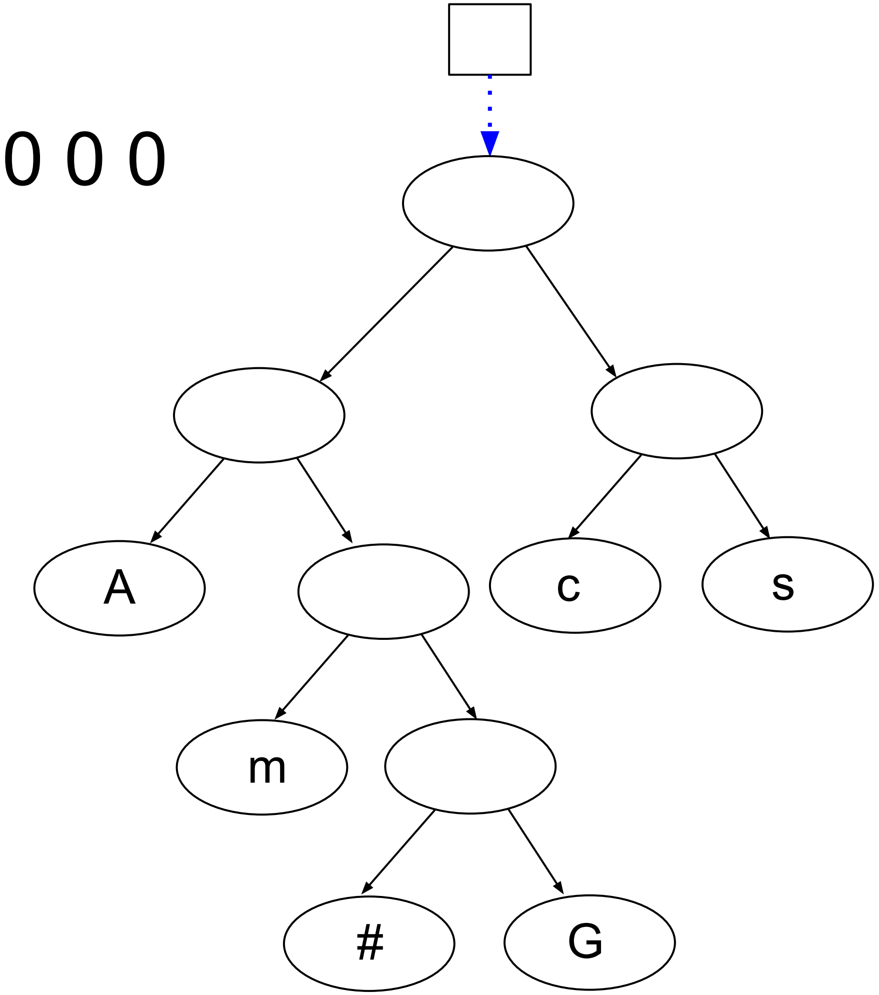
```
typedef struct treenode
{
  struct treenode * left;
  struct treenode * right;
  char value; // character
  int occurrence;
} TreeNode;
```

```
typedef struct listnode
{
  struct listnode * next;
  TreeNode * tnptr;
} ListNode;
```

HW 17

- Input: 1A 1m 1# 1 G 0 00 1c 1s 0 0 0
- Build the tree
- Output the code book:

A	0	0		
m	0	1	0	
#	0	1	1	0
G	0	1	1	1
c	1	0		
s	1	1		



HW 18 (bits)

compress #AcGms#Ac

A	0	0		
m	0	1	0	
#	0	1	1	0
G	0	1	1	1
c	1	0		
s	1	1		

data	#	A	c	G	m	s	#	A	c
code (bits)	0110	00	10	0111	010	11	0110	00	10
byte	0110 00 10			0111 010 1		1 0110 00 1			0
byte	0110 00 10			0111 010 1		1 0110 00 1			0

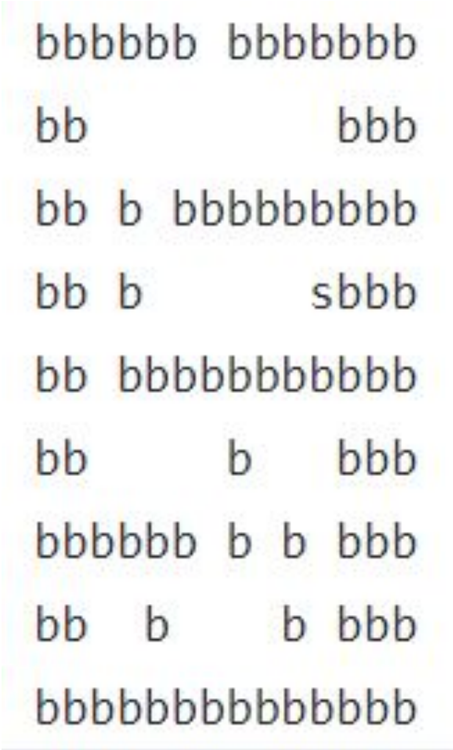
add 7 zeros

xxd –b output:

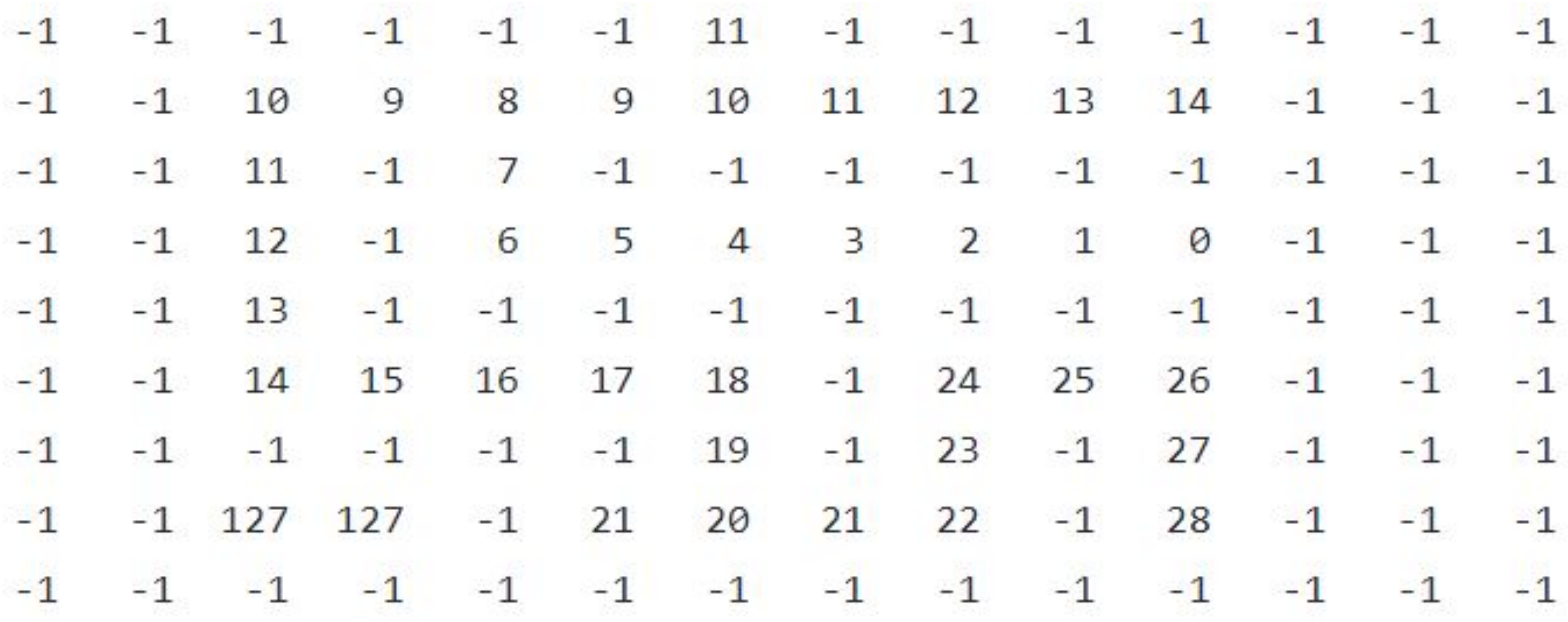
```
0110 00 10    0111 010 1     1 0110 00 1     0000 0000
```

Homework 19
Maze

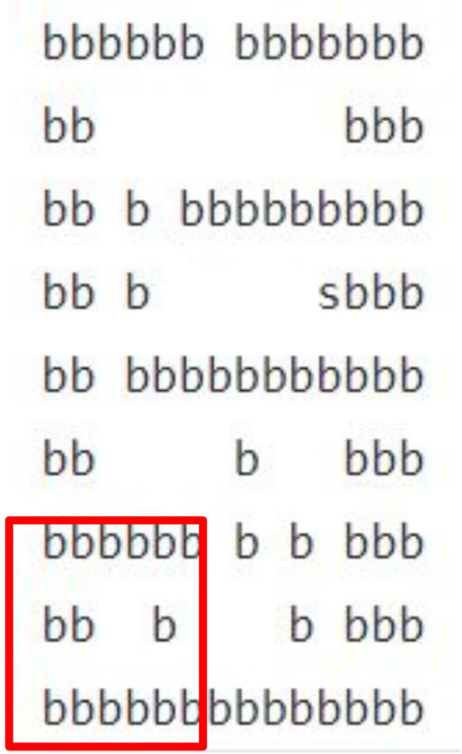
yunglu@purdue.edu



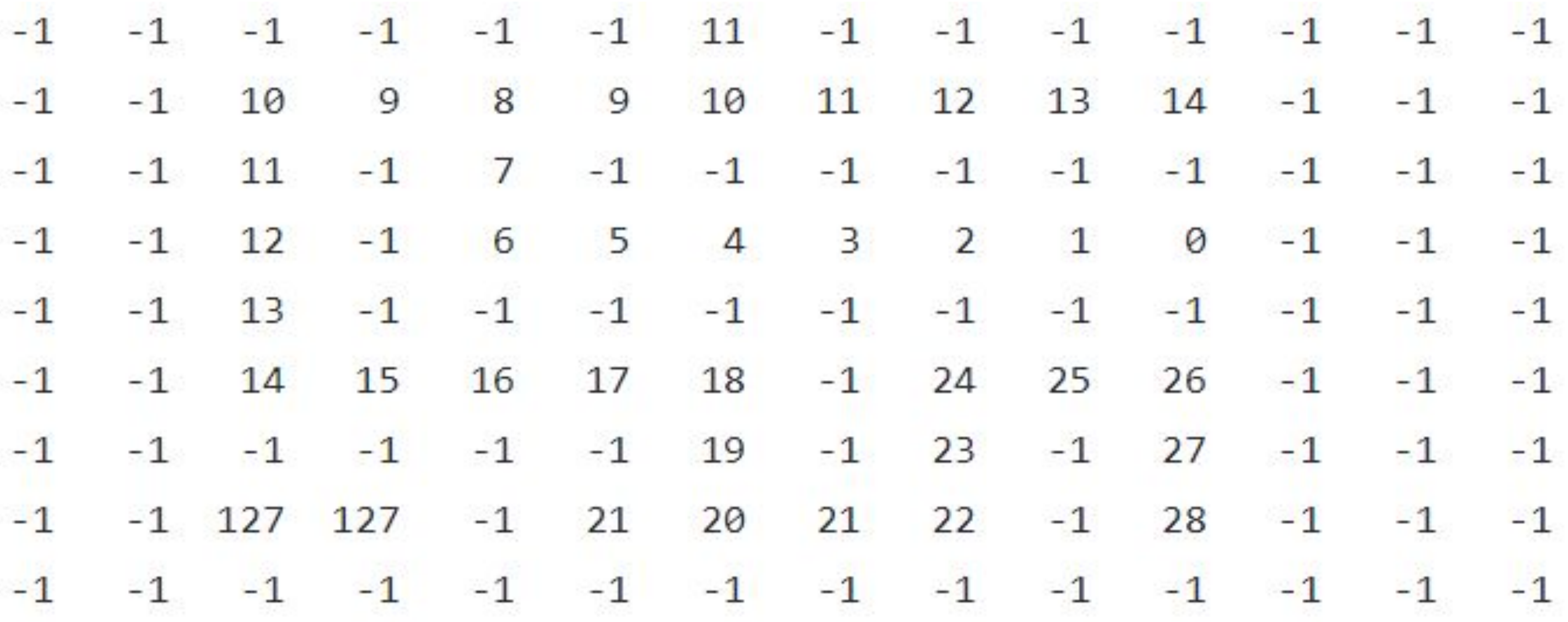
```
bbbbbb bbbbbbb
bb         bbb
bb b bbbbbbbbb
bb b      sbbb
bb bbbbbbbbbbb
bb     b   bbb
bbbbbb b b bbb
bb  b    b bbb
bbbbbbbbbbbbbb
```



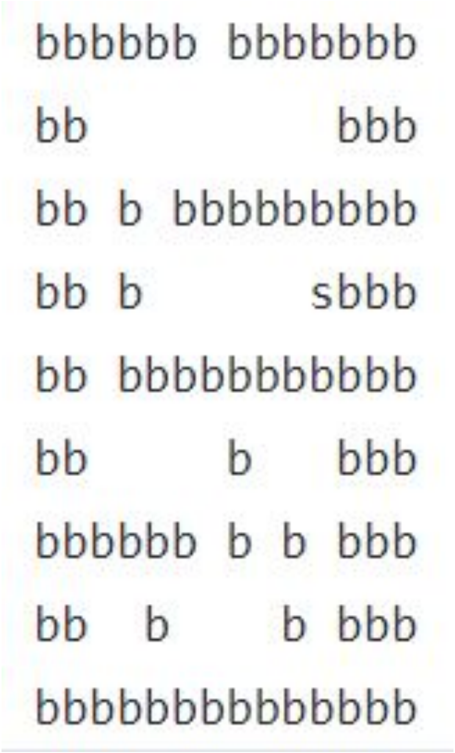
```
   -1   -1   -1   -1   -1   -1   11   -1   -1   -1   -1   -1   -1   -1
   -1   -1   10    9    8    9   10   11   12   13   14   -1   -1   -1
   -1   -1   11   -1    7   -1   -1   -1   -1   -1   -1   -1   -1   -1
   -1   -1   12   -1    6    5    4    3    2    1    0   -1   -1   -1
   -1   -1   13   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1
   -1   -1   14   15   16   17   18   -1   24   25   26   -1   -1   -1
   -1   -1   -1   -1   -1   -1   19   -1   23   -1   27   -1   -1   -1
   -1   -1  127  127   -1   21   20   21   22   -1   28   -1   -1   -1
   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1
```



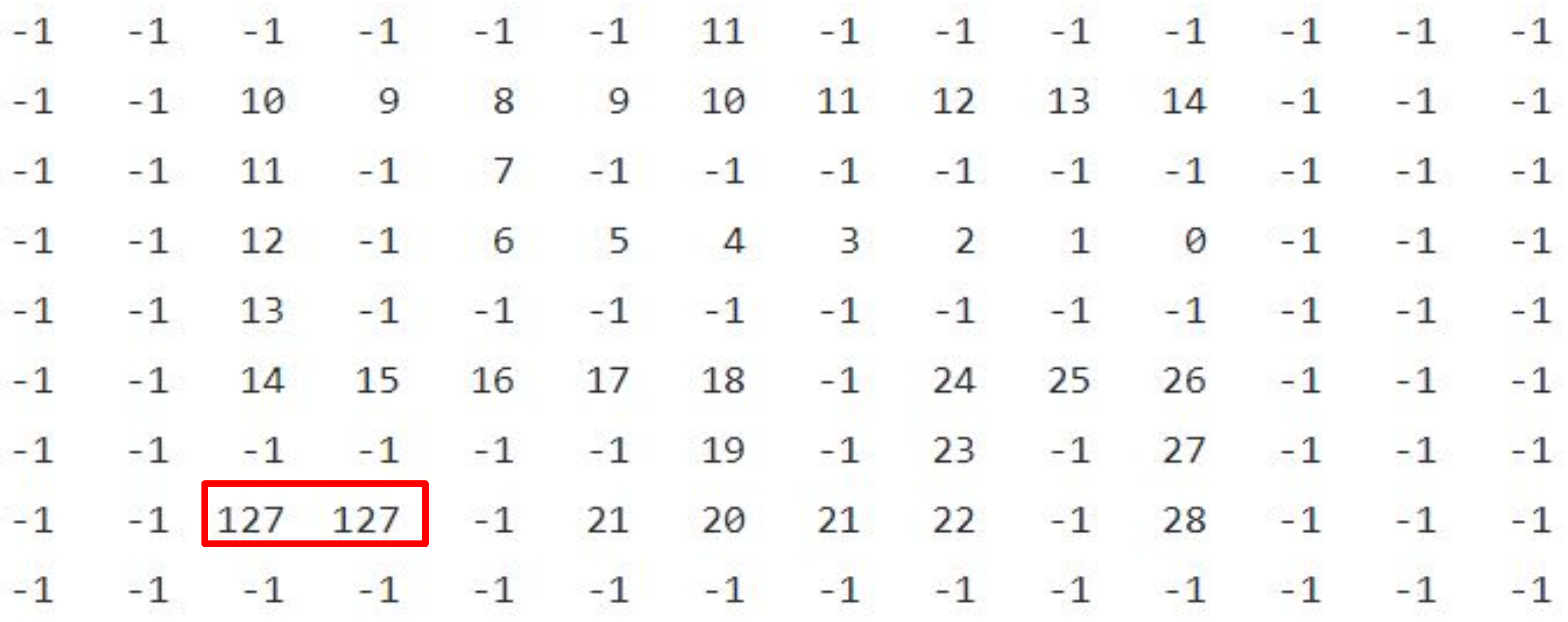
```
bbbbbb bbbbbbb
bb         bbb
bb b bbbbbbbbb
bb b      sbbb
bb bbbbbbbbbbb
bb     b   bbb
bbbbbb b b bbb
bb  b    b bbb
bbbbbbbbbbbbbb
```



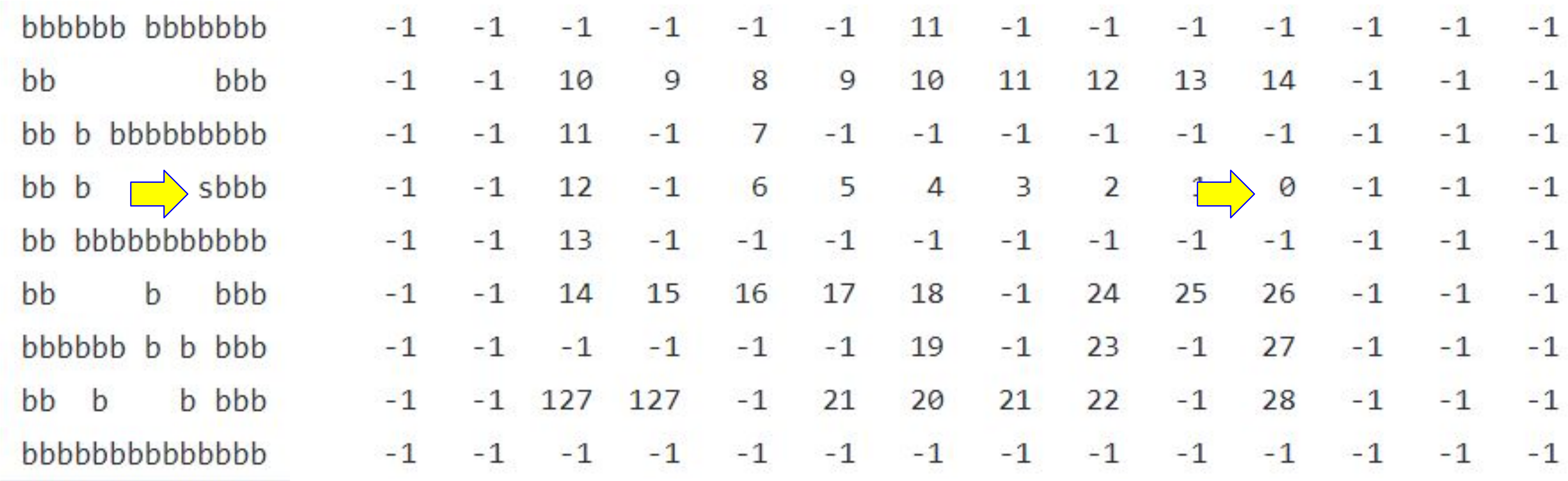
```
 -1   -1   -1   -1   -1   -1   11   -1   -1   -1   -1   -1   -1   -1
 -1   -1   10    9    8    9   10   11   12   13   14   -1   -1   -1
 -1   -1   11   -1    7   -1   -1   -1   -1   -1   -1   -1   -1   -1
 -1   -1   12   -1    6    5    4    3    2    1    0   -1   -1   -1
 -1   -1   13   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1
 -1   -1   14   15   16   17   18   -1   24   25   26   -1   -1   -1
 -1   -1   -1   -1   -1   -1   19   -1   23   -1   27   -1   -1   -1
 -1   -1  127  127   -1   21   20   21   22   -1   28   -1   -1   -1
 -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1
```



```
bbbbbb bbbbbbb
bb         bbb
bb b bbbbbbbbb
bb b      sbbb
bb bbbbbbbbbbb
bb     b   bbb
bbbbbb b b bbb
bb  b    b bbb
bbbbbbbbbbbbbb
```

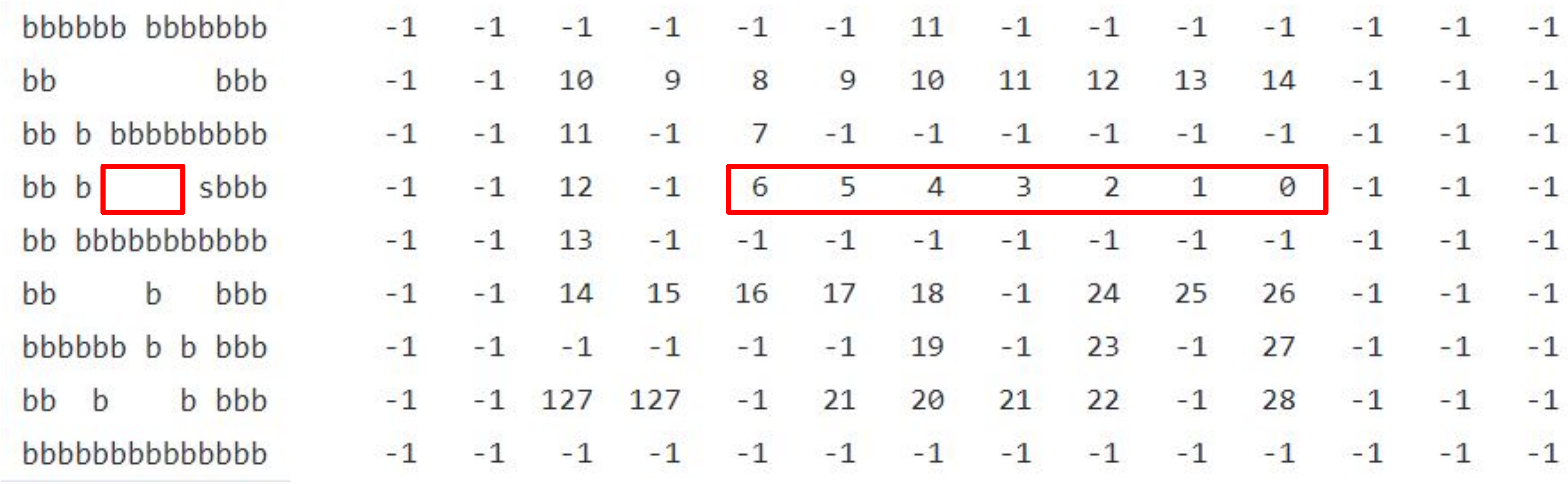


```
 -1   -1   -1   -1   -1   -1   11   -1   -1   -1   -1   -1   -1   -1
 -1   -1   10    9    8    9   10   11   12   13   14   -1   -1   -1
 -1   -1   11   -1    7   -1   -1   -1   -1   -1   -1   -1   -1   -1
 -1   -1   12   -1    6    5    4    3    2    1    0   -1   -1   -1
 -1   -1   13   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1
 -1   -1   14   15   16   17   18   -1   24   25   26   -1   -1   -1
 -1   -1   -1   -1   -1   -1   19   -1   23   -1   27   -1   -1   -1
 -1   -1  127  127   -1   21   20   21   22   -1   28   -1   -1   -1
 -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1
```

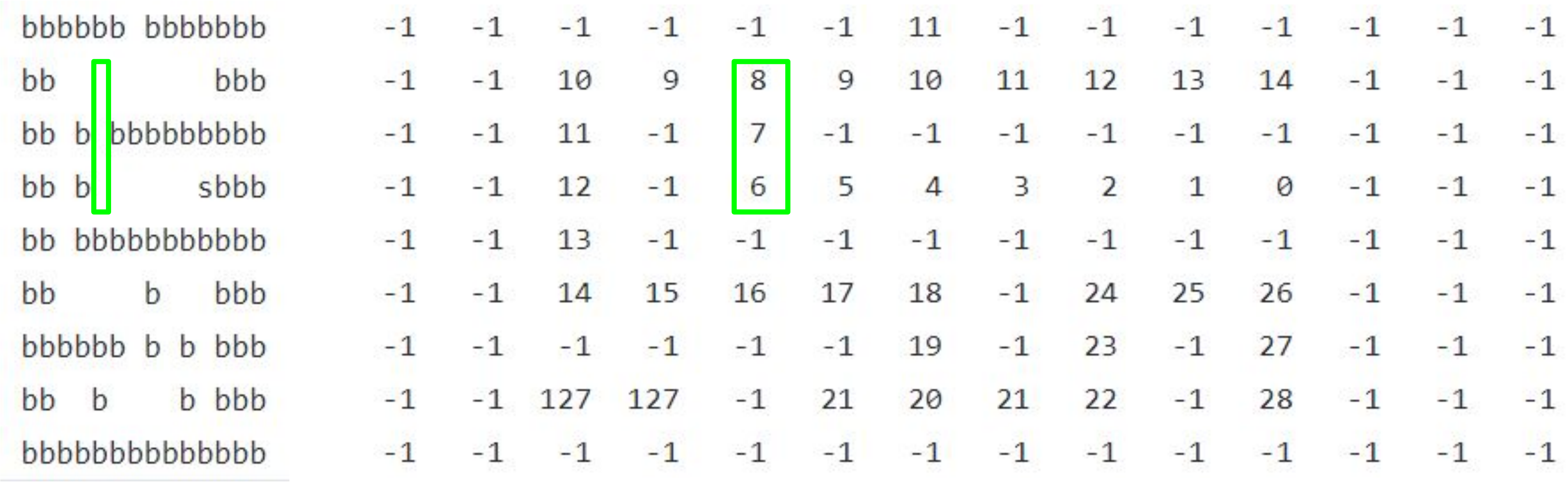


```
bbbbbb bbbbbbb
bb         bbb
bb b bbbbbbbbb
bb b      sbbb
bb bbbbbbbbbbb
bb     b   bbb
bbbbbb b b bbb
bb  b    b bbb
bbbbbbbbbbbbbb
```

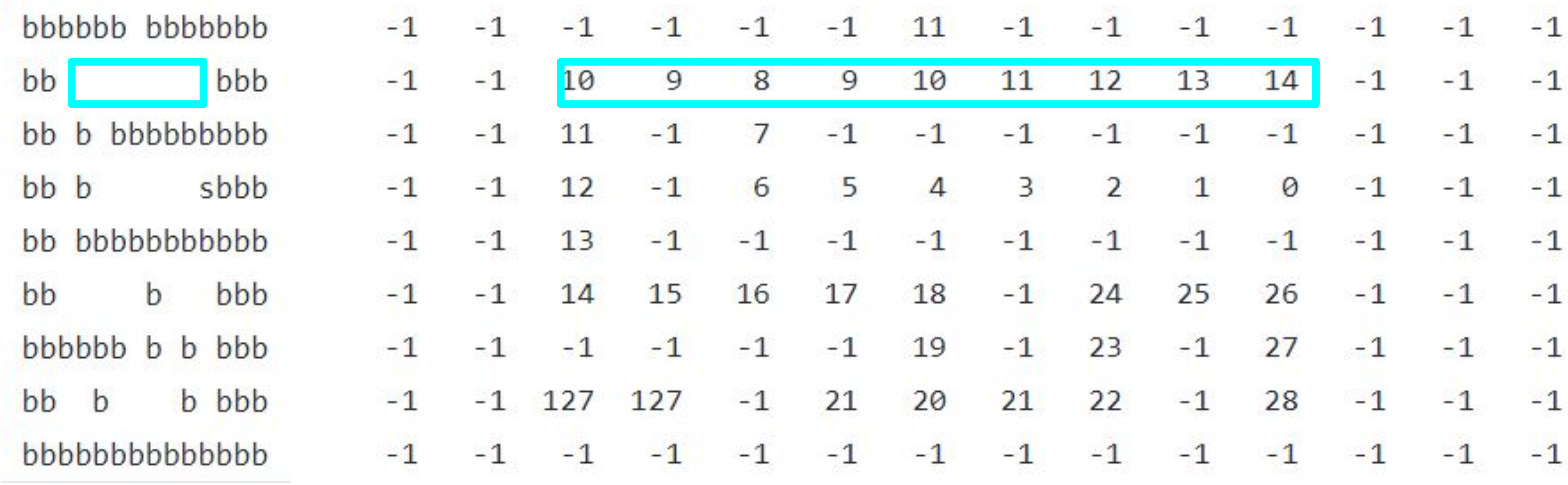
```
  -1  -1  -1  -1  -1  -1  11  -1  -1  -1  -1  -1  -1  -1
  -1  -1  10   9   8   9  10  11  12  13  14  -1  -1  -1
  -1  -1  11  -1   7  -1  -1  -1  -1  -1  -1  -1  -1  -1
  -1  -1  12  -1   6   5   4   3   2   1   0  -1  -1  -1
  -1  -1  13  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1
  -1  -1  14  15  16  17  18  -1  24  25  26  -1  -1  -1
  -1  -1  -1  -1  -1  -1  19  -1  23  -1  27  -1  -1  -1
  -1  -1 127 127  -1  21  20  21  22  -1  28  -1  -1  -1
  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1
```



```
bbbbbb bbbbbbb     -1  -1  -1  -1  -1  -1  11  -1  -1  -1  -1  -1  -1  -1
bb         bbb     -1  -1  10   9   8   9  10  11  12  13  14  -1  -1  -1
bb b bbbbbbbbb     -1  -1  11  -1   7  -1  -1  -1  -1  -1  -1  -1  -1  -1
bb b      sbbb     -1  -1  12  -1   6   5   4   3   2   1   0  -1  -1  -1
bb bbbbbbbbbbb     -1  -1  13  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1
bb     b   bbb     -1  -1  14  15  16  17  18  -1  24  25  26  -1  -1  -1
bbbbbb b b bbb     -1  -1  -1  -1  -1  -1  19  -1  23  -1  27  -1  -1  -1
bb  b    b bbb     -1  -1 127 127  -1  21  20  21  22  -1  28  -1  -1  -1
bbbbbbbbbbbbbb     -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1  -1
```

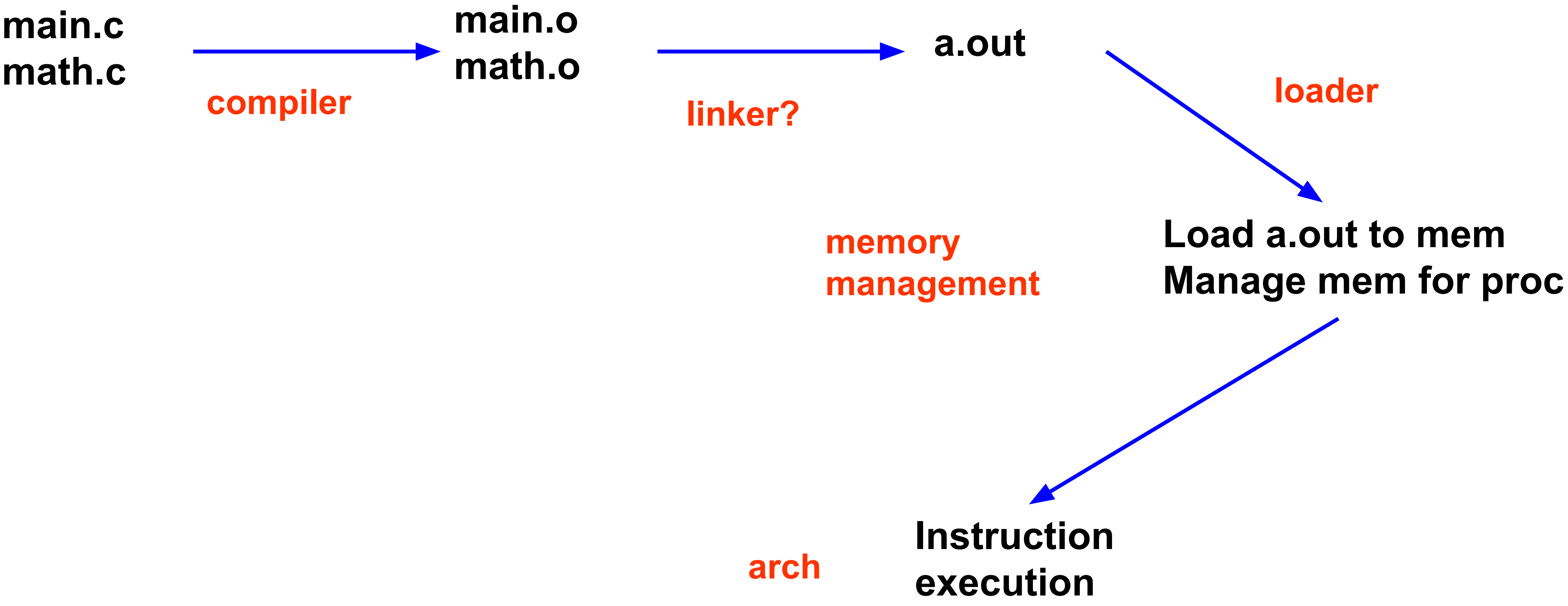


```
bbbbbb bbbbbbb     -1   -1   -1   -1   -1   -1   11   -1   -1   -1   -1   -1   -1   -1
bb         bbb     -1   -1   10    9    8    9   10   11   12   13   14   -1   -1   -1
bb b bbbbbbbbb     -1   -1   11   -1    7   -1   -1   -1   -1   -1   -1   -1   -1   -1
bb b      sbbb     -1   -1   12   -1    6    5    4    3    2    1    0   -1   -1   -1
bb bbbbbbbbbbb     -1   -1   13   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1
bb     b   bbb     -1   -1   14   15   16   17   18   -1   24   25   26   -1   -1   -1
bbbbbb b b bbb     -1   -1   -1   -1   -1   -1   19   -1   23   -1   27   -1   -1   -1
bb  b    b bbb     -1   -1  127  127   -1   21   20   21   22   -1   28   -1   -1   -1
bbbbbbbbbbbbbb     -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1
```



```
bbbbbb bbbbbbb    -1   -1   -1   -1   -1   -1   11   -1   -1   -1   -1   -1   -1   -1
bb         bbb    -1   -1   10    9    8    9   10   11   12   13   14   -1   -1   -1
bb b bbbbbbbbb    -1   -1   11   -1    7   -1   -1   -1   -1   -1   -1   -1   -1   -1
bb b      sbbb    -1   -1   12   -1    6    5    4    3    2    1    0   -1   -1   -1
bb bbbbbbbbbbb    -1   -1   13   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1
bb     b   bbb    -1   -1   14   15   16   17   18   -1   24   25   26   -1   -1   -1
bbbbbb b b bbb    -1   -1   -1   -1   -1   -1   19   -1   23   -1   27   -1   -1   -1
bb  b    b bbb    -1   -1  127  127   -1   21   20   21   22   -1   28   -1   -1   -1
bbbbbbbbbbbbbb    -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1   -1
```

Compilation and Linking



main.c
math.c

compiler

main.o
math.o

linker?

a.out

loader

memory
management

Load a.out to mem
Manage mem for proc

arch

Instruction
execution

Example

Main.c:

```
extern float sin( );
main( )
{
  static float x, val;


  printf(“Type number: ”);
  scanf(“%f”, &x);
  val = sin(x);
  printf(“Sine is %f”, val);
}
```

Math.c:

```
float sin(float x)
{
  static float temp1, temp2, result;

  – Calculate Sine –

  return result;
}
```

Example (cont)

- Main.c uses externally defined sin() and C library function calls
  - printf()
  - scanf()

- How does this program get compiled and linked?

Compiler

- Compiler: generates object file
  - Information is incomplete
  - Each file may refer to symbols defined in other files

Components of Object File

- Header

- Two segments
  - Code segment and data segment
  - OS adds empty heap/stack segment while loading

- Size and address of each segment
  - Address of a segment is the address where the segment begins.

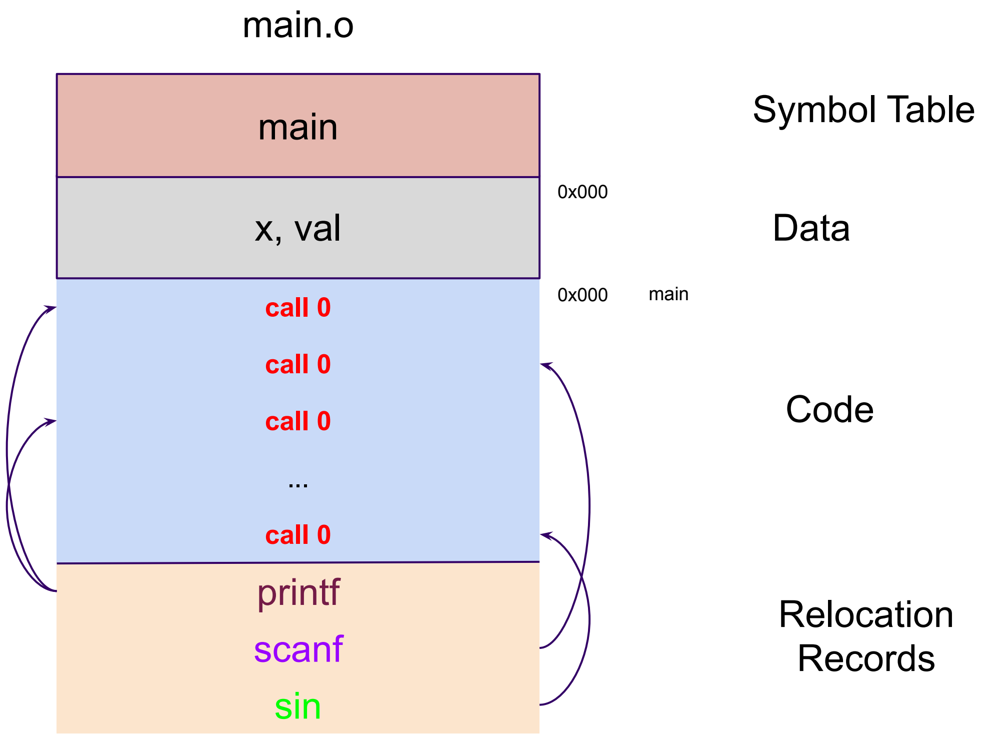
Components of Object File (cont)

- Symbol table
  - Information about stuff defined in this module
  - Used for getting from the name of a thing (subroutine/variable) to the thing itself
- **Relocation information**
  - Information about addresses in this module linker should fix
    - External references (e.g. lib call)
    - Internal references (e.g. absolute jumps)
- Additional information for debugger

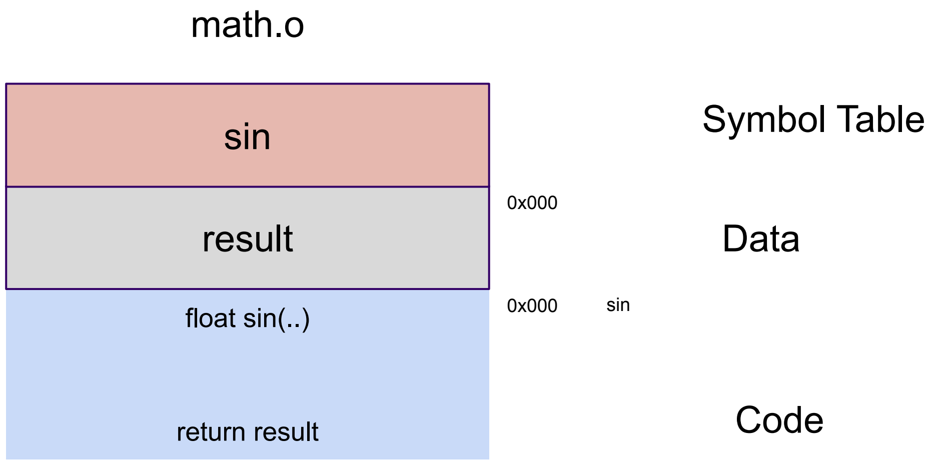
What could the compiler not do?

- Compiler does not know final memory layout
  - It assumes everything in .o starts at address zero
  - For each .o file, compiler puts information in the symbol table to tell the linker how to rearrange outside references safely/efficiently
    - For exported functions, absolute jumps, etc

Compiler: main.c



Compiler: math.c



Linker functionality

- Three functions of a linker
  - Collect all the pieces of a program
  - Figure out new memory organization
    - Combine like segments
    - Does the ordering matter? (spatial locality for cache)
  - Touch-up addresses

- The result is a runnable object file (e.g. a.out)

Linker – a closer look

- Linker can shuffle segments around at will, but cannot rearrange information within a segment

Linker requires at least two passes

- Pass 1: decide how to arrange memory
- Pass 2: address touch-up

Pass 1 – Segment Relocation

- Pass 1 assigns input segment locations to fill-up output segments
  - Read and adjust symbol table information
  - Read relocation info to see what additional stuff from libraries is required

Pass 2 – Address translation

- In pass 2, linker reads segment and relocation information from files, fixes up addresses, and writes a new object file

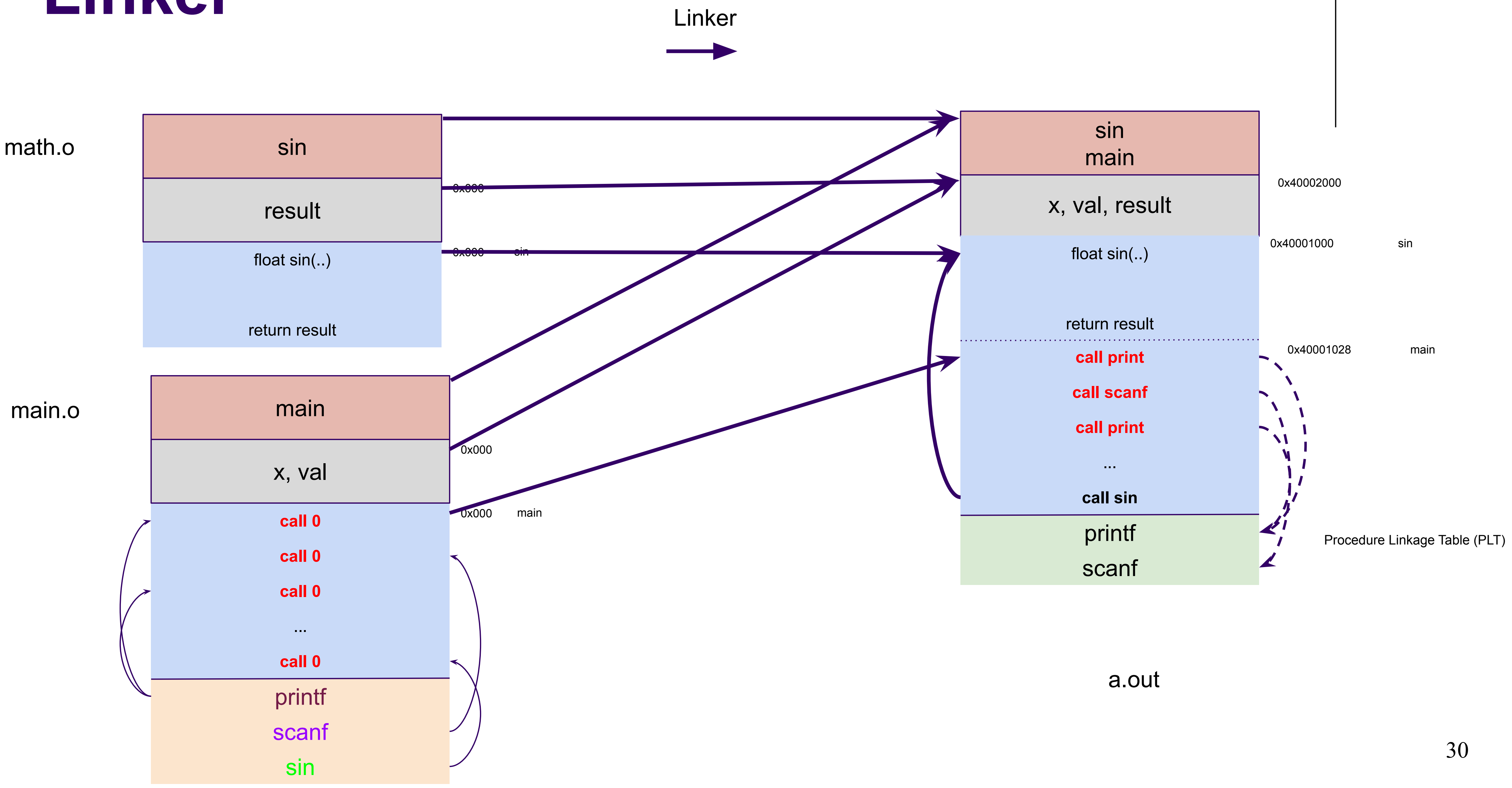
- Relocation information is crucial for this part

Putting It Together

- Pass 1:
  - Read symbol table, relocation table
  - Rearrange segments, adjust symbol table

- Pass 2:
  - Read segments and relocation information
  - Touch-up addresses
  - Write new object file

Linker



Linker

math.o

sin

result

float sin(..)

return result

0x000

0x000 sin

main.o

main

x, val

call 0

call 0

call 0

...

call 0

printf

scanf

sin

0x000

0x000 main

sin
main

x, val, result

float sin(..)

return result

call print

call scanf

call print

...

call sin

printf

scanf

0x40002000

0x40001000 sin

0x40001028 main

Procedure Linkage Table (PLT)

a.out