Welcome to DrRacket, version 8.15 [cs].
Language: racket, with debugging; memory limit: 128 MB.

A C program, and how information is placed on the run-time stack at three moments in time.

The purpose is to help understand how the run-time stack holds the current function's local variables (incl. its parameters), as well as a vital piece of book-keeping: what program-instruction to resume at, when the current helper function finishes.

This is important cybersecurity, since it explains how an attacker, if they can get a stack overflow to both (a) place malicious code onto the stack, *and* (b) overwrite the return-instruction-pointer so that the program 'returns' to that malicious code rather than the real return-site, then the attacker has achieved "running of arbitrary code".
It's important to programmers, since knowing which data are on the heap vs the stack explains why local variables are local, and what overhead function-calls might incur.

Notes:
- "%rip" is a local system variable for "return instruction pointer" -- where to resume the program at,
  when finishing the current helper function.
- "%rsp" is a local system variable for "return stack pointer" -- where to adjust the top-of-stack to,
  when finishing the current helper function.

"The sample C program:"

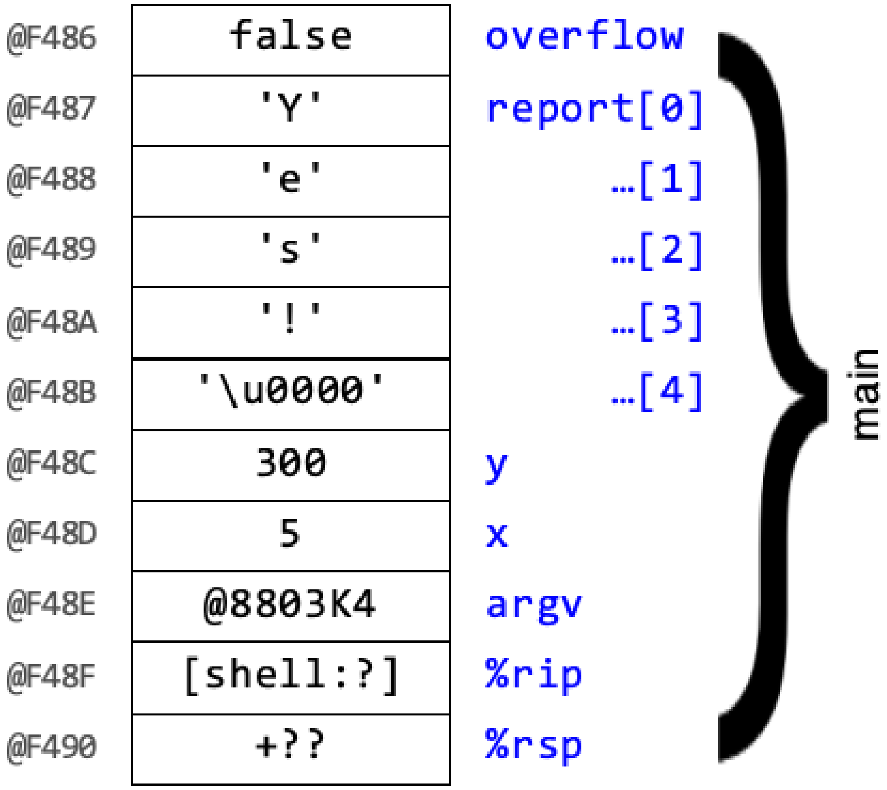
```
#include <stdio.h>
#include <stdbool.h>
```

```
// Return x*y.
//
int multiply (int x, int y) {
  return x*y;
}


// Return a^b.
//
int power(int a, unsigned int b) {
  int product = 1;
  while (b!=0) {
    product = multiply(product, a);
    b--;
  }
  return product;
}


int main( int argc, char* argv ) {
  printf("This program verifies whether 5 to the 300th
power is bigger than 0.\n");
  unsigned int x = 5;
  unsigned int y = 300;
  char report[5] = "Yes!";
  bool overflow;

  if (power(x,y) >= 0) {
    printf("%s\n",report);
    overflow = false;
  }
  else {
    printf( "It's not!  Hmmm; overflow?\n" );
    overflow = true;
  }
  return overflow; // indicate an error, to the shell /
caller.
}
""
"The stack, as main first calls power first and it in
turn calls multiply the first time:"
```



@F486	false	overflow
@F487	'Y'	report[0]
@F488	'e'	…[1]
@F489	's'	…[2]
@F48A	'!'	…[3]
@F48B	'\u0000'	…[4]
@F48C	300	y
@F48D	5	x
@F48E	@8803K4	argv
@F48F	[shell:?]	%rip
@F490	+??	%rsp

(list

@F481	1	product	power
@F482	300	b	
@F483	5	a	
@F484	[main:4]	%rip	
@F485	+11	%rsp	
@F486	false	overflow	main
@F487	'Y'	report[0]	
@F488	'e'	…[1]	
@F489	's'	…[2]	
@F48A	'!'	…[3]	
@F48B	'\u0000'	…[4]	
@F48C	300	y	
@F48D	5	x	
@F48E	@8803K4	argv	
@F48F	[shell:?]	%rip	
@F490	+??	%rsp	

@F47D	5	y	multiply
@F47E	1	x	
@F47F	[power:3]	%rip	
@F480	+5	%rsp	
@F481	1	product	power
@F482	300	b	
@F483	5	a	
@F484	[main:4]	%rip	
@F485	+11	%rsp	
@F486	false	overflow	main
@F487	'Y'	report[0]	
@F488	'e'	…[1]	
@F489	's'	…[2]	
@F48A	'!'	…[3]	
@F48B	'\u0000'	…[4]	
@F48C	300	y	
@F48D	5	x	
@F48E	@8803K4	argv	
@F48F	[shell:?]	%rip	
@F490	+??	%rsp	

)

>