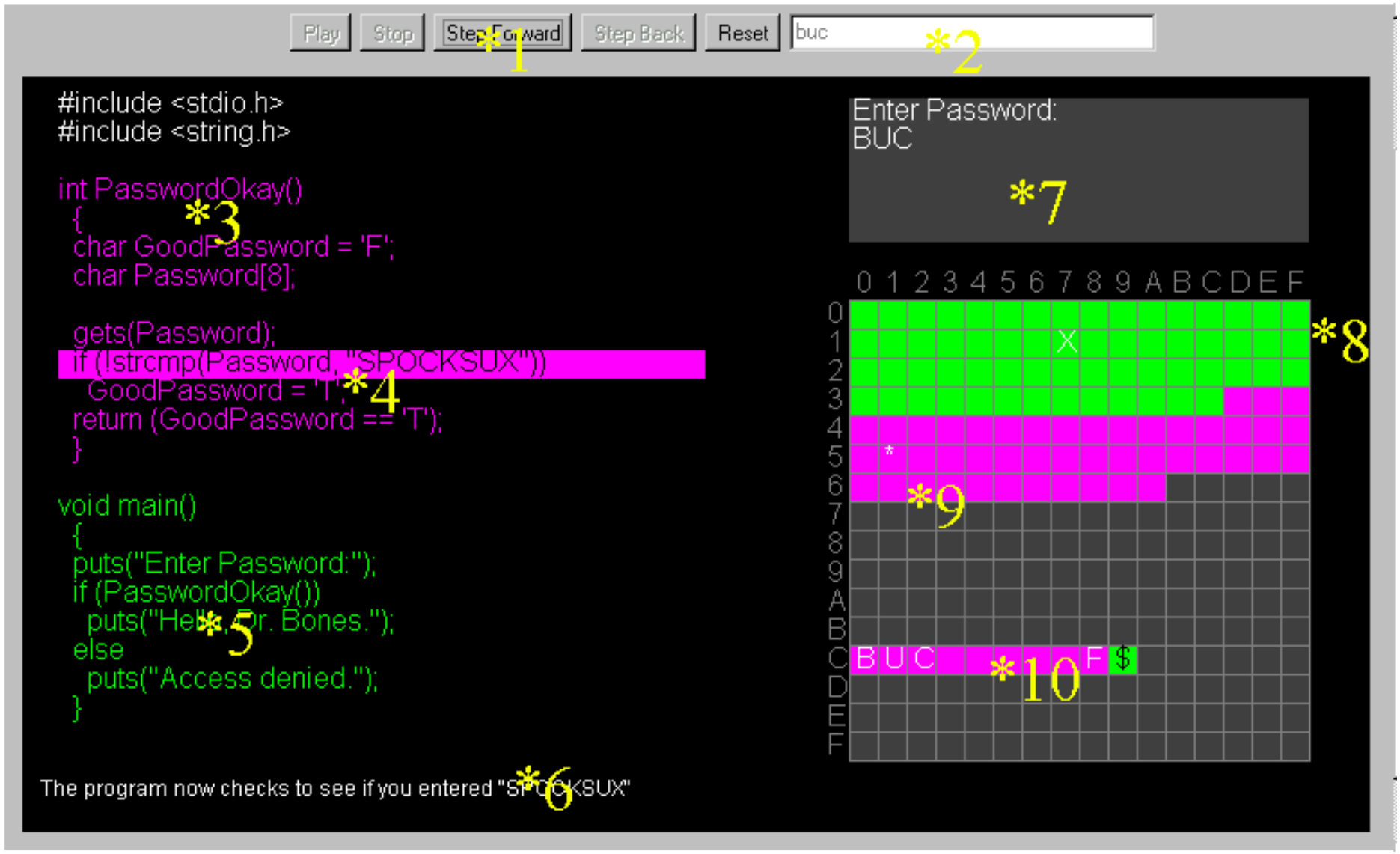
Play
Stop
Step Forward
Step Back
Reset
buc
*1
*2
#include <stdio.h>
#include <string.h>
int PasswordOkay()
{
*3
char GoodPassword = 'F';
char Password[8];
gets(Password);
if (!strcmp(Password, "SPOCKSUX"))
GoodPassword = 'T';
*4
return (GoodPassword == 'T');
}
void main()
{
puts("Enter Password:");
if (PasswordOkay())
puts("Hello, Dr. Bones.");
*5
else
puts("Access denied.");
}
The program now checks to see if you entered "SPOCKSUX"
*6
Enter Password:
BUC
*7
0 1 2 3 4 5 6 7 8 9 A B C D E F
X
*8
*
*9
B U C
F
$
*10

1 - buttons

- Click “Step Forward” to *execute* one line of the program.
- Click “Reset” to start over.
- “Play,” “Stop,” and “Step Back” are disabled in this version of the applet.

2 – input text box

- The input text box is enabled whenever a **gets()** function is being executed (library functions are not treated as separate subroutines in this applet). When it is enabled you may activate it and give input to the program by typing into the input box. The text box will ignore all non-alphabetic characters (‘1’, ‘?’, ‘~’, etc.) and will covert all lower case characters to upper case before passing the input on to the program.

3 – source code

- This is the C *source code* for the program that is being executed. You should understand several things even if you are not a C programmer: 1. **PasswordOkay()** will call the **PasswordOkay()** subroutine, 2. **gets()** reads a string from the user and **puts()** prints a string, 3. **strcmp()** compares two strings to see if they are the same, and 4. A statement like **char A[5] = "HELLO"** will allocate a five byte buffer called A and initialize it to "HELLO". **char Num = 'R'** allocates one byte called Num and sets it equal to 'R'.

4 – highlighted line

- The highlighted line of code is the one currently being executed. This lets you watch the path that the program execution takes.

5 – more source code

- Notice that the source code for different subroutines is also different colors.
- Also notice how an **if** statement looks:

if (*something is true*)

do one thing;

else

do something else;

6 – notes

- Every line of source code has a note that is shown on the bottom of the screen as it is executed. These notes might be hints or explanations of what is going on.

7- I/O area

- The Input/Output area is much like the *terminal* of a DOS or UNIX machine. The program can print things out for you to see and it can ask you to type things in (using the input text box) for it to store. Remember that you have to activate the input text box by clicking on it to give the program any input, and that you can only do this when the input text box is enabled during a **gets()** statement.

8 – memory

- There are 256 bytes of *memory*. Memory is laid out in *hexadecimal* notation (0x??) from left to right, top to bottom like you would read a book. 0x00 is the the first *byte* in memory, 0xFF is the last. 0xC9 is the '$'. It's just like looking something up in a table.

9 – code text

- Code text is the program *compiled* into a format that the computer can understand. It is just a series of numbers like anything else in memory, except that each number represents a unique computer *instruction*.
- Note that a '*' will show the current position of the *program counter* (how the computer keeps track of where it is) and an 'X' will be placed over an instruction to signify that that instruction called another subroutine and will be returned to.
- The color of the C code for a subroutine is the same as the color of the code text for that subroutine in memory.

10 - stack

- The stack stores return pointers, temporary variables and buffers, and sometimes other things.
- When something stored on the stack is no longer needed it is destroyed. You can watch the stack be built and destroyed as you execute the program.
- Notice that a ‘$’ is a return pointer to the subroutine with the same color as the box the ‘$’ is in.
- A ‘?’ is a *canary* which only StackGuard uses.

Glossary

- Execute – To execute the program means to "run" it. The two terms can be used interchangeably.
- Source code – this is the code that the programmer writes in a high-level language like Java, BASIC, or, in this case, C. A high-level language is one that humans can easily read and understand.
- Terminal – A terminal is a combination of a screen and a keyboard where the user of a program can type input into the program and receive output back.
- Memory – Memory is where the computer stores the code text for a program, the variables, the stack, and everything else that the program needs.
- Hexadecimal – The decimal system uses a base of 10, where 2863 has a 2 in the 1000's place, an 8 in the 100's place, a 6 in the 10's place, and a 3 in the 1's place. Hexadecimal uses 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F instead of just 0 through 9, where A = 10, B = 11 … F = 15. The base of hexadecimal is 16, so it has a 1's place, a 16's place, a 256's place, a 4096's place, etc. 0 through 255 can be represented in hexadecimal as 0x00 through 0xFF. 0x?? is a common convention for distinguishing hexadecimal numbers from decimal numbers.
- Compiled – A compiler takes the source code a programmer has written and turns it into code text that a computer can easily execute. Code text is a sequence of instructions.
- Instruction – An instruction tells the computer what to do, but it is very low-level. Unlike source code, it works at the computer level telling the computer what elementary steps to take to execute the program.
- Program counter – The instructions are stored as a sequence of numbers in the computer just like anything else. The computer usually executes one instruction after another unless it is told to *jump* somewhere else. The program counter keeps track of what instruction in memory is being executed at any given time.
- Jump – A jump occurs when the computer is told to start executing instructions somewhere else besides the next instruction after the current one being executed.
- Canary – A canary is put in memory and if it is overwritten then something is wrong. It is analogous to the canaries used in coal mines to detect poisonous gas.