Demonstration of safe string concatenation in C using `strncat()`

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

int main() {
   char s1[] = "security";
   char s2[15] = "software";
   char s3[10] = "";

   printf("Before strncat operations...\n");
   printf("s1: \"%s\" size: %zu length: %zu\n", s1, sizeof(s1), strlen(s1));
   printf("s2: \"%s\" size: %zu length: %zu\n", s2, sizeof(s2), strlen(s2));
   printf("s3: \"%s\" size: %zu length: %zu\n", s3, sizeof(s3), strlen(s3));

   strncat(s2,s1,sizeof(s2)-strlen(s2)-1);       // append s1 to s2
   strncat(s3,s1,sizeof(s3)-strlen(s3)-1);       // append s1 to s3
   strncat(s3,s2,sizeof(s3)-strlen(s3)-1);       // append s2 to s3

   printf("After strncat operations...\n");
   printf("s1: \"%s\" size: %zu length: %zu\n", s1, sizeof(s1), strlen(s1));
   printf("s2: \"%s\" size: %zu length: %zu\n", s2, sizeof(s2), strlen(s2));
   printf("s3: \"%s\" size: %zu length: %zu\n", s3, sizeof(s3), strlen(s3));
}
```

Let's assume the storage for arrays s1, s2, and s3 is allocated in the order of declarations in the source code (high memory address to low memory address).

Step 1: Show the stack frame contents BEFORE call to `strncat` function (left table on next page)

Step 2: Use the before call stack frame contents to determine the output produced by the first three print statements.

```
s1: "security" size: 9 length: 8
s2: "software" size: 15 length: 8
s3: "" size: 10 length: 0
```

Step 3: Show the stack frame contents AFTER call to `strncat` function (right table on next page)

Step 4: Use the after call stack frame contents to determine the output produced by the last three print statements.

```
s1: "security" size: 9 length: 8
s2: "softwaresecuri" size: 15 length: 14
s3: "securitys" size: 10 length: 9
```

Stack frame (before call to `strncat`)

High address	RA (4 bytes)
CFP →	PFP (4 bytes)
s1[8]	'\0'
	'y'
	't'
	'i'
	'r'
	'u'
	'c'
	'e'
s1[0]	's'
s2[14]	
	'\0'
	'e'
	'r'
	'a'
	'w'
	't'
	'f'
	'o'
s2[0]	's'
s3[9]	
Low address s3[0]	'\0'

Stack frame (after call to `strncat`)

High address	RA (4 bytes)
CFP →	PFP (4 bytes)
s1[8]	'\0'
	'y'
	't'
	'i'
	'r'
	'u'
	'c'
	'e'
s1[0]	's'
s2[14]	'\0'
	'i'
	'r'
	'u'
	'c'
	'e'
	~~'\0'~~ 's'
	'e'
	'r'
	'a'
	'w'
	't'
	'f'
	'o'
s2[0]	's'
s3[9]	'\0'
	~~'\0'~~ 's'
	'y'
	't'
	'i'
	'r'
	'u'
	'c'
	'e'
Low address s3[0]	~~'\0'~~ 's'