HW2.9. Common C Pitfalls

For each of the following code samples (all of which compile), determine if the code works as intended on a **32-bit system**, or if there is an error. If there is an error, choose an error type from the following list:

Probable Segfault: The code attempts to access memory not directly allocated by the program. For the purposes of this question, we also assume that accesses past the end of a stack- or heap-allocated array yield probable segfaults. (In practice, stack and heap allocations tend to be part of a larger block of allocated memory, so the system fails to catch the invalid memory access).

Incorrect free: The function free() is called on a pointer which cannot be freed.

Logic error: The code does not work as intended at least sometimes.

Memory Leak: The code runs to completion without any of the previous errors, but allocates memory which cannot be freed later.

Assume that all necessary libraries have been included, and that the system is 32-bit.

Q1.1:

```
#define PASSWORD "correct horse battery staple"
char *check_permissions (char *user_guess) {
     if (user_guess = PASSWORD) {
          return "access granted";
     }
     return "access denied";
}
```

○ (a) Probable Segfault

○ (b) Incorrect Free

○ (c) Logic Error

○ (d) Memory Leak

○ (e) No Error

Q1.2:

```
#define PASSWORD "hunter2"
char *check_permissions (char *user_guess) {
     if (user_guess == PASSWORD) {
          return "access granted";
     }
     return "access denied";
}
```

○ (a) Probable Segfault

○ (b) Incorrect Free

○ (c) Logic Error

○ (d) Memory Leak

○ (e) No Error

Q1.3:

```
int main () {
     char *x = malloc(0xFFFFFFFF * sizeof(char));
     x[0] = '!';
}
```

○ (a) Probable Segfault

○ (b) Incorrect Free

○ (c) Logic Error

○ (d) Memory Leak

Homework 2

Assessment overview

Total points: 75/100

Score: 75%

Question

Value: 10

History:

Awarded points: 0/10

Report an error in this question

Previous question

Next question

Attached files

No attached files

Attach a file

Attach text

○ (e) No Error

Q1.4:

```
int foo () {
     int *x = malloc(20);
     x[0] = x[1] = 1;
     x[2] = x[0] + x[1];
     x[3] = 99;
     return x[3];
}
```

○ (a) Probable Segfault

○ (b) Incorrect Free

○ (c) Logic Error

○ (d) Memory Leak

○ (e) No Error

Q1.5:

```
int main () {
  char *x = "patrick";
  printf("%s", x);
  free(x); // tidy up
}
```

○ (a) Probable Segfault

○ (b) Incorrect Free

○ (c) Logic Error

○ (d) Memory Leak

○ (e) No Error

Q1.6:

```
/*Copies up to maxlength characters from source into destination.
It is assumed that destination points to a buffer sufficiently long to hold
the copied string, and that source is a valid string.
You can assume that the maxlength input argument corresponds to the length of
the source string*/
void strncpy(char* destination, char* source, uint32_t maxlength) {
  int i = 0;
  while(i < maxlength && source[i]) {
    destination[i]=source[i];
    i++;
  }
  destination[strlen(destination)]='\0';
}
```

○ (a) Probable Segfault

○ (b) Incorrect Free

○ (c) Logic Error

○ (d) Memory Leak

○ (e) No Error

Q1.7:

```
#DEFINE MAXSTRINGLENGTH 64 //Assume that the string input is at most 64
characters long
char* copystringtoheap(char* string) {
  char* c = malloc(sizeof(char)*(MAXSTRINGLENGTH+2));
  char* cclone = c;
  do {
   *(c++) = *string;
  } while (*(string++));
  free(c);
  return cclone;
}
```

○ (a) Probable Segfault

○ (b) Incorrect Free

○ (c) Logic Error

○ (d) Memory Leak

○ (e) No Error

Save & Grade *20 attempts left* Save only *Additional attempts available with new variants* ?