CS100 Spring 2025

Quiz 1

Mar 19, 2025

1. (15 points) Name: _____; No.: _____; Email: _____@shanghaitech.edu.cn

2. (25 points) [**C**] Select the pieces of code that have undefined behaviors.

   A.
   ```c
   #include <stdio.h>
   int main(void) {
     int *ptr = NULL;
     printf("%d\n", *ptr);
   }
   ```

   B.
   ```c
   int main(void) {
     int a[10];
     for (int i = 0; i <= 10; ++i)
       a[i] = 0;
   }
   ```

   C.
   ```c
   #include <stdio.h>
   int* foo(void) {
     static int a[10];
     return a;
   }
   int main(void) {
     int *ptr = foo();
     printf("%d\n", *ptr);
   }
   ```

   D.
   ```c
   int main(void) {
     int x = 1;
     x += (x+=2) + (++x);
   }
   ```

   E.
   ```c
   int main(void) {
     int cnt = 0;
     for (int i = 1; i <= 10; ++i)
       for (int i = 1; i <= 10; ++i)
         ++cnt;
   }
   ```

   F.
   ```c
   #include <stdio.h>
   #include <stdlib.h>
   int a[] = {1, 2, 3, 4, 5, 6};
   int main(void) {
     printf("%d\n", a[0]);
     free(a);
   }
   ```

3. (30 points) [**C**] The following code is to allocate $n \times m$ integers memory into 2-dimensional array form. Please fill the blank corresponding to the comments in the code, each blank should be filled with one statement.

```c
#include <stdlib.h>
int main(void) {
  int n, m;
  scanf("%d%d", &n, &m);

  int **ptr = malloc(/* (a) Allocate memory for an array of pointers to row */);

  for (int i = 0; i < n; ++i)
    /* (b) Allocate memory for each row */

  for (int i = 0; i < n; ++i)
    for (int j = 0; j < m; ++j)
      ptr[i][j] = i * n + j;

  for (int i = 0; i < n; ++i)
    /* (c) Free memory for each row */

  free(ptr);
}
```

(a) `sizeof(int*) * n`

(b) `ptr[i] = malloc(sizeof(int) * m)`

(c) `free(ptr[i])`

4. (15 points) [**C**] The following function is intended to remove the first `cnt` characters from a string and shift the remaining characters to the front. Does it implement this behavior correctly? If not, explain what is wrong.

```c
#include <string.h>
#include <stddef.h>

/*  @brief Removes the first `cnt` characters from the given string and shifts the remaining
 *         characters to the front.  If `cnt` is greater than or equal to the length of the
 *         string, the string will be set to an empty string. The behavior is undefined if
 *         `str` does not point to a null-terminated string.
 *  @param str A pointer to a null-terminated byte string that will be modified.
 *  @param cnt The number of characters to be removed from the beginning of the string.
 */
void pop(char *str, size_t cnt) {
  if (cnt >= strlen(str)) {
    *str = '\0';
    return;
  }
  while (*(str + cnt) != '\0') {
    *str = *(str + cnt);
    ++str;
  }
}
```

For those who are **unfamiliar** with the *C standard library function* `strlen`, the following summary (adapted from `en.cppreference.com`) provides a clear explanation:

> The function `strlen` is defined in the header `<string.h>`:
>
> ```
> size_t strlen( const char* str );
> ```
>
> `strlen` returns the length of the given null-terminated byte string, that is, the number of characters in a character array whose first element is pointed to by `str` up to and not including the first null character. The behavior is **undefined** if `str` is not a pointer to a null-terminated byte string.

Solution: Incorrect. The function shifts the characters but forgets to add a null terminator (‘\0‘) at the end of the string.

To fix this, add a null terminator after the loop:

```
while (*(str + cnt) != '\0') {
  *str = *(str + cnt);
  ++str;
}
*str = '\0';  // Add the null terminator at the end
```

5. (15 points) [**C**] Read the following code. Write the output of the code. If the code contains a compile error or undefined behavior, please write ‘CE‘ or ‘UB‘ in the blank.

```
#include <stdio.h>
#define SIZEOF_UINT 32

void trans(unsigned x, char **s) {
  if (x > 1) {
    trans(x >> 1, s);    // Recursive call to process the higher bits
    (*s)++;              // Move the pointer to the next character
  }
  **s = (x & 1u) + '0';  // Store the bit as a character
}

int main(void) {
  // Initialize string filled with '\0's (null characters)
  char str[SIZEOF_UINT + 1] = {'\0'};

  char *ptr = str;

  trans(148, &ptr);

  printf("%s", str);  // Write down the output of this printf statement below

  return 0;
}
```

Solution: The function `trans` recursively converts the integer `x` into its binary representation.

- It works by continuously dividing the integer `x` by 2 (through right-shifting `x` by 1 bit, `x >> 1`) and recursively processing the higher-order bits until the integer becomes less than 2.
- During the recursion, the pointer `*s` is moved (incremented) after each recursive call to store each bit in the string `s`.
- The function stores each bit of `x` starting from the least significant bit (LSB) into the string `s`, converting each bit to its corresponding character ('0' or '1') by adding `'0'` to the result of `x & 1`.

When the function `trans` is called with `x = 148`:

- The binary representation of 148 is `10010100`.
- The function processes each bit, starting from the least significant bit (rightmost), and stores it in the string `s`.
- The pointer `ptr` is incremented after storing each bit, ensuring that the binary digits are written sequentially into the string.
- Once all bits are processed, the string `s` contains the binary form of 148, which is `"10010100"`.

Therefore, the output of the program is the binary representation of the integer 148: `10010100`.