CSSE2310

•••

Reptest and Mid-Sem Style Questions

Week 3, Sem 1, 2020

Repository Testing Tools

- To run the public tests on the files in your repository, you can use `reptest*.sh`
- To practice using `reptest`, do the following:

1. Svn checkout
   `https://source.eait.uq.edu.au/svn/csse2310-s???????/trunk/playground`
2. Commit fixed `calc.c` and `Makefile`. If you don't yet have the files, run:
   `cp ~uqjfenw1/public/test_tool/src/* .`
3. Run `reptesttest.sh`

Midsem style questions

...

No compiler allowed!

Q1.

```
int x = 5;
int y = 3;
float z = 4 + x / y;
printf("%e", z);
```

Q1.

```
int x = 5;
int y = 3;
float z = 4 + x / y;
printf("%e", z);
```

- Integer division
  - `4 + x / y`
    `= 4 + (5 / 3)`
    `= 4 + 1`
    `= 5`
- Exponential format specifier (%e)
  - `5.00e+00`

Q2.

```
int x = 5 % 3;
printf("%d", x);
```

Q2.

```
int x = 5 % 3;
printf("%d", x);
```

- Modulo operator (%)
  - Returns the remainder after division
- `5/3 = (1*3)+2 so 5%3 == 2`

Q3.

```
char* s = "abcde";
char* t = s + 2;
printf("%c", t[1]);
```

Q3.

```
char* s = "abcde";
char* t = s + 2;
printf("%c", t[1]);
```

- Strings are just null-terminated blocks of memory
  - You can move forward and backward through memory using pointer arithmetic
  - `s` points to the start of the string “abcde” in memory
  - `t` points two ‘spaces’ past the memory pointed to by `s`, so it points to ‘c’ in memory
  - `t[1]` accesses the memory one ‘space’ after `t` and therefore points at `'d'`

Q4.

```
int i = 5;
for (int i = 0; i < 3; ++i) {
    printf("%d,", i * 2);
}
printf("%d", i);
```

Q4.

```
int i = 5;
for (int i = 0; i < 3; ++i) {
    printf("%d,", i * 2);
}
printf("%d", i);
```

- The loop iterates three times
  - `i = 0,1,2`
  - After the `i = 2` loop:
    - `i` increments to 3
    - The condition `(i < 3)` is false and we break out of the loop
  - We print the value i * 2 for each iteration:
    - `0,2,4,`
- The initialisation of `i` in the loop body masks `i` in the outer scope
  - `i` in the outer scope is unchanged by the loop, so we print `5`

Q5.

```
int x = 50;
while (x > 0) {
    if (x % 2 == 0) {
        x--;
    } else if (x % 3 == 0) {
        x -= 2;
    } else {
        printf("%d", x);
    }
}
```

Q5.

```
int x = 50;
while (x > 0) {
    if (x % 2 == 0) {
        x--;
    } else if (x % 3 == 0) {
        x -= 2;
    } else {
        printf("%d", x);
    }
}
```

- This is an infinite loop
  - We enter with x = 50. So x % 2 == 0 is true and x becomes 49
  - In the next iteration:
    - x % 2 == 1 and x % 3 == 1
    - So we print x (49)
  - x remains unchanged in each successive iteration, so we never break out of the loop and keep printing 49

Q6.

```
int x = 19;
int y = 9;
printf("%d", x^y^y);
```

Q6.

```
int x = 19;
int y = 9;
printf("%d", x ^ y ^ y);
```

- ^ is the XOR operator
  - Something XOR'd with itself is 0
  - Something XOR'd with 0 is unchanged
  - `x^y^y = x^0 = x`
  - So we print the value of x (19)

Q7.

```
char x = 'G';
int d = 'b' - 'B';
printf("%c", (char)(x + d));
```

Q7.

```
char x = 'G';
int d = 'b' - 'B';
printf("%c", (char)(x + d));
```

- We can add and subtract chars
  - Characters are just 8-bit values and we can perform arithmetic on them
  - You can convert from upper to lowercase or vice versa by adding or subtracting respectively
    - E.g. `'a' = 97` and `'A' = 65`
  - Here, we are adding the difference between the upper and lowercase letters to the character ‘G’, so we are effectively converting uppercase ‘G’ to lowercase ‘g’
  - So we print `'g'`

Q8.

```
int x = 5;
int y = 3;
int z = 6;
if (x < y)
    x += 2;
    y--;
if (z > y)
    z++;
printf("%d %d %d", x, y, z);
```

Q8.

```
int x = 5;
int y = 3;
int z = 6;
if (x < y)
    x += 2;
    y--;
if (z > y)
    z++;
printf("%d %d %d", x, y, z);
```

- If statements only affect the following statement
  - If we omit the braces, the if statement body only consists of the statement directly following it
  - `y--` is not part of the first if statement
  - This is also true for loops and other construct bodies
- Execution path:
  - `x < y (5 < 3)` is false (don't enter if)
  - Decrement y `(y == 2)`
  - `z > y (6 > 2)` is true, increment z
  - Print values: `5, 2, 7`

Q9.

```
for (int i = 0; i < 10; ++i) {
    for (int j = 0; j < 3; ++j) {
        if (i > j) {
            i++;
        }
    }
    printf("%d %d", i, j);
}
```

Q9.

```
for (int i = 0; i < 10; ++i) {
    for (int j = 0; j < 3; ++j) {
        if (i > j) {
            i++;
        }
    }
    printf("%d %d", i, j);
}
```

- We cannot print variables which are out of scope
  - `j` only exists within the inner loop body
  - We don't know if there exists a `j` in the outer scope
  - We don't know what the output will be. This may not even compile.

Q10.

```
for (int i=0; i<5; ++i); {
    printf("Hello\n");
}
```

Q10.

```
for (int i=0; i<5; ++i); {
    printf("Hello\n");
}
```

- The body of a loop construct stops at the end of the next statement
  - We have an additional semicolon after the loop, so the loop body is empty
  - The curly braces have no effect in this piece of code
  - We print `“Hello\n”` once

Q11.

```
void f(char* buffer, const char* s) {
    char buff2[2];
    buff2[1] = '\0';
    if ((*s == 'a') || (*s == 'e') ||
            (*s == 'i') || (*s == 'o') ||
            (*s == 'u') || (*s == 'w')) {
        buff2[0] = *s - ('a' - 'A');
    } else {
        buff2[0] = *s;
    }
    if (buff2[0] != 0) {
        strcat(buffer, buff2);
        f(buffer, s + 1);
    }
}
```

```
int main(int argc, char** argv) {
    char res[100];
    res[0] = '\0';
    f(res, "Hello world");
    printf("%s\n", res);
    return 0;
}
```

Today

Activities:

- `testtest.sh` and `reptesttest.sh`
- Implement `join2.c` from Thursday's lecture

Assignment 1:

- Need to have finished C and Linux tutorials / exercises
- Current spec version is 1.0
- Due 6pm Monday March 23
- testa1.sh / reptesta1.sh / style.sh