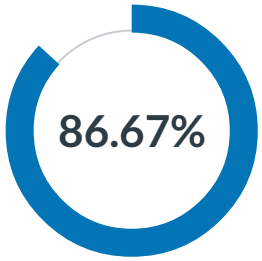
Results

ALVIN ZHENG



13
Out of 15 points

25:32
Time for this attempt

Your Answers:

1 / 1 point

Select all statements that are true of C, regardless of architecture.

- [x] sizeof(char) == 1
- [ ] sizeof(int) == 4
- [ ] sizeof(short int) < sizeof(int)
- [x] sizeof(signed int) == sizeof(unsigned int)
- [x] sizeof(short int) < sizeof(long int)

- [ ] sizeof(int) == sizeof(float)

2 1 / 1 point

Consider this code fragment:

```
int n = 0;
int *p = &n;

*p = 10;

printf("%d\n", n);
```

What will be printed?

- [ ] 0
- [x] 10
- [ ] Nothing: this code contains a type error
- [ ] Undetermined: this code contains a memory error

3 1 / 1 point

We with to allocate an array of integers of length L, where L is a variable whose value will not be known until runtime. Which of the following will allocate such an array?

- [ ] int A[] = malloc(L * sizeof(int));
- [ ] int A[] = malloc(sizeof(L * int));
- [x] int *A = malloc(L * sizeof(int));
- [ ]

sizeof(A) = L * sizeof(int);

None of the above

4 1 / 1 point

In C, which of the following should NOT be put in a header file?

Function prototypes

Macro definitions

Type definitions

✓ Function definitions

None of the above

5 1 / 1 point

Assume the following type definition:

```
struct item {
        char name[32];
        int value;
        int weight;
};
```

Assume also the following initialized local variable:

```
struct item *items = malloc(itemCount * sizeof(struct item));
```

Which of the following statements would be allowed (i.e., would not produce an error from the compiler)?

items[3].name = "foo";

items[3]->weight = 10;

items.weight[3] = 10;

items[3] = malloc(sizeof(struct item));

✓ None of the above

6 1 / 1 point

We want a macro in our code such that LENGTH will be replaced by some constant, such as 256. What keyword will we use?

✓ #define

#macro

#begin

#const

None of the above

7 1 / 1 point

Assume these initialized local variables:

```
int x = 5;
int *p = &x;
```

Which of the statements below does NOT have the same effect as the others?

x = 6;

*p = 6;

++x;

✓ ++p;

None of the above (all statements have the same effect)

8 0 / 1 point

Assuming these initialized local variables:

```
int *p = malloc(20 * sizeof(int));
int *q = p + 10;
```

What will free(q) do?

Deallocate the array that p points to

Deallocate the integer p[10]

Shrink the array that p points to to contain 10 integers

✗ Deallocate the local variable q

Correct Answer: **None of the above**

None of the above

9 1 / 1 point

Consider the following type definition and partial function definition:

```
struct node {
        int data;
        struct node *next;
};

void insert_head(int n, struct node **p)
{
        assert(p != NULL);
```

```
        struct node *new = malloc(sizeof(struct node));
        new->data = data;
        new->next = *p;
        /* HERE */
}
```

What makes the most sense to replace the comment /* HERE */?

- p = &new;
- ✓ *p = new;
- **p = *new;
- memcpy(*p, new, sizeof(struct node));
- None of the above

10 1 / 1 point

The function points_to_positive() takes a pointer to an integer and tests whether the integer is positive. We need the function signature to guarantee that the pointed-to integer is not modified.

```
int points_to_positive(/* ARGUMENT */)
{
    return *p > 0;
}
```

Which possible replacement for /* ARGUMENT */ best expresses this?

- int *p
- ✓ const int *p
- int * const p
- int *p const

None of the above

11 0 / 1 point

Assume a file "foo" exists in the working directory and contains at least 100 bytes. Consider this code fragment:

```
int d, r;
char b[32];
d = open("foo", O_RDONLY);
r = read(d, b, 20);
```

Assuming that open() and read() do not report errors (d != -1 and r != -1), what is the **minimum** number of bytes written to b?

0

1

2

× 20

Correct Answer: **1**

None of the above

12 1 / 1 point

Assume we have (successfully) opened a file:

```
int f = open("some file", O_RDONLY);
```

and used dup() to (successfully) create a second file descriptor:

```
int d = dup(f);
```

What would we need to do to ensure that this file is closed?

close(f)

close(d)

✓ Both close(f) and close(d)

Either close(f) or close(d), but not both

None of the above

13 1 / 1 point

Process A uses fork() to create a child process, B. Under what circumstances will process B become a zombie process?

Process A terminates without calling wait()

✓ Process B terminates, but process A has not yet called wait()

Process B terminates after process A calls wait()

Process B enters an infinite loop

None of the above

14 1 / 1 point

Process A calls fork(), creating process B, and then calls wait(). Process B calls fork(), creating process C, and terminates. When process C terminates, what will be its status?

C will be an orphan process

C will be a zombie process

✓ C will be an orphan zombie process

- It depends on whether process A calls wait() a second time
- None of the above

15 1 / 1 point

Assume the following code fragment:

```
execl("foo", "foo", NULL);
puts("hello");
```

Under what circumstances will this program print "hello"?

- It will print "hello" after "foo" terminates
- It will print "hello" while "foo" is executing
- It will never print "hello"
- ✓ It will print "hello" if exec() fails
- None of the above