

The "C/C++ Programming 2" Pretest

For all questions assume a perfectly implemented ANSI/ISO C/C++ compiler. How a program runs on your computer only indicates how your computer runs the program, not necessarily how it should run or how portable it is. Therefore, consider all of these questions theoretical since running them as programs can be catastrophically misleading. Below each question is a reference to the location in "C/C++ Notes" where the topic is mentioned or discussed. This assignment will not be collected or graded and the answers are provided on the last page.

1. In C, if variables x , y , and z are properly declared, what is *syntactically* wrong with: $z = y/*\textit{division}*/x$;
(Note 1.4)
 - A. Nothing is wrong.
 - B. Everything after the `//` is a comment so the statement is incomplete.
 - C. It is not portable.
 - D. A comment may not serve as whitespace.
 - E. The value of y may be too large.
2. The value of `sizeof('A')` is always:
(Note 1.5; Note 2.12)
 - A. the same as the value of `sizeof(char)`.
 - B. the same as `sizeof(int)` in C and the same as `sizeof(char)` in C++.
 - C. 65 if the ASCII character set is used.
 - D. dependent upon the character set being used.
 - E. none of the above.
3. Assuming a 16 bit type `int` and a 32 bit type `long`, the data types of 32767, -32768, 32768, and 2. are:
(Note 2.1; Note 2.2)
 - A. `int, int, long, float`
 - B. `int, long, long, float`
 - C. `int, long, long, double`
 - D. implementation dependent
 - E. none of the above
4. Predict the output from `cout << oct << 15 << dec << 15 << hex << 15`
(Note 2.6)
 - A. oct 15 dec 15 hex 15
 - B. 017 15 0xf
 - C. 17 15 f
 - D. 1715f
 - E. *Output is implementation dependent.*
5. The values of `-5/4` and `-5%4`, respectively, are:
(Note 2.8)
 - A. implementation dependent : `-5/4 == -1` and `-5%4 == -1` or `-5/4 == -2` and `-5%4 == 3`
 - B. -1 and -1
 - C. -2 and 3
 - D. -1 and -2
 - E. none of the above.
6. Assuming `int ax[123]`, the data types of `+(char)65`, `(short)47+(char)65`, `2*6e2L`, and `sizeof(ax)` are:
(Note 2.10)
 - A. undefined, undefined, `double`, pointer to `int`
 - B. `char, short, long, size_t`
 - C. `int, int, long double, pointer to int`
 - D. `int, int, long double, size_t`
 - E. implementation dependent

- 1 7. What is the value and data type of the expression: `(char)25 < (char)100`
2 (Note 2.10; Note 3.1)
3 A. 1; type **char**
4 B. 1; type **int**
5 C. 75; type **char**
6 D. 0; type **char**
7 E. none of the above
8
- 9 8. Predict what gets printed:
10 (Note 2.14)
11 **const int i;**
12 **for (i = 0; i < 5; ++i)**
13 **cout << i << ' ';**
14 A. 0 1 2 3 4
15 B. 0 1 2 3 4 5
16 C. 1 2 3 4 5
17 D. *It won't compile.*
18 E. *Output is implementation dependent.*
19
- 20 9. Predict what gets printed by: `printf("Goodbye") && printf(" Cruel") || printf(" World")`
21 (Note 3.2)
22 A. Goodbye
23 B. Goodbye Cruel
24 C. Goodbye Cruel World
25 D. Goodbye World
26 E. *Output is implementation dependent.*
27
- 28 10. For `int x = 1;` predict the value in `x` after: `x = ++x`
29 (Note 3.4)
30 A. 1
31 B. 2
32 C. 3
33 D. undefined
34 E. implementation dependent
35
- 36 11. Predict final value of `i`:
37 (Note 3.10)
38 **for (int i = 0; i < 5; ++i)**
39 **break;**
40 A. 0
41 B. 1
42 C. 2
43 D. 3
44 E. none of the above
45
- 46 12. Predict the value in `x` after: `auto int x = (4, printf("Hello"), sqrt(64.), printf("World"));`
47 (Note 3.11)
48 A. 4
49 B. 5
50 C. 6
51 D. 8
52 E. implementation dependent
53

- 1 13. Predict the output from:
 2 (Note 3.15)
 3 **if** (5 < 4)
 4 **if** (6 > 5)
 5 putchar('1');
 6 **else if** (4 > 3)
 7 putchar('2');
 8 **else**
 9 putchar('3');
 10 putchar('4');
- 11 A. 4
 12 B. 2
 13 C. 24
 14 D. 4 or 24, depending upon the implementation
 15 E. Nothing is printed.
- 16
 17 14. Predict what gets printed by: `cout << (12 < 5 ? "Hello " : "World")`
 18 (Note 3.16)
 19 A. Hello
 20 B. Hello World
 21 C. World
 22 D. World Hello
 23 E. Output is undefined or implementation dependent.
- 24
 25 15. Predict what will happen:
 26 (Note 4.3)
 27 **char** ch;
 28 **while** ((ch = cin.get()) != EOF)
 29 cout.put(ch);
- 30 A. A false EOF might be detected or the real EOF might be missed
 31 B. It won't compile
 32 C. cin.get() reads one **int** at a time from input, then its value is printed
 33 D. EOF is not defined in C++
 34 E. Nothing unwanted happens. It simply reads and prints characters until EOF is reached.
- 35
 36 16. Predict what gets printed:
 37 (Note 4.6)
 38 cout << __DATE__ << __FILE__ << __LINE__
- 39 A. Today's date, the name of the executable file, and the number of lines in the file
 40 B. The compilation date, the source file name, and the source file line number containing __LINE__
 41 C. cout cannot use these macros without typecasts
 42 D. Output is implementation dependent.
 43 E. It won't compile.
- 44
 45 17. In C with no prototype, what data types get passed to *fcn* by the call: `fcn((char)23, (short)34, 87, 6.8f)`
 46 (Note 5.5)
 47 A. **char, short, int, float**
 48 B. **char, short, long, float**
 49 C. **int, int, int, float**
 50 D. **int, int, int, double**
 51 E. none of the above or implementation dependent
 52

1 18. In C, which statement is true concerning a major problem with the following?

2 (Note 5.4)

```
3     long double fx(void)
4     {
5         double answer = sum(1.1, 2.2, 3.3);
6         return printf("answer = %f", answer);
7     }
8     double sum(double a, double b, double c)
9     {
10        return(a + b + c);
11    }
```

- 12 A. The name *sum* conflicts with a standard ANSI math function.
 13 B. The **return** statement in *fx* returns type **double**.
 14 C. Return statements may not contain an algebraic expression ($a + b + c$).
 15 D. There is nothing wrong with the program.
 16 E. The call to *sum* assumes that *sum* returns type **int**.

17
 18 19. In C++, what gets printed?

19 (Note 5.7)

```
20     void print(int x = 1, int y = 2, int z = 3)
21     {
22         cout << x << y << z;
23     }
24     int main()
25     {
26         print(), print(4), print(5, 6), print(7, 8, 9);
27         return(EXIT_SUCCESS);
28     }
```

- 29 A. 123
 30 B. 456789
 31 C. 123456789
 32 D. 123423563789
 33 E. *It won't compile.*

34
 35 20. In C++, predict what gets printed?

36 (Note 5.8)

```
37     void print(int x, int y = 2, int z = 3) { cout << x << y << z; }
38     void print(long x, int y = 5, int z = 6) { cout << x << y << z; }
39     int main()
40     {
41         print(4), print(4L);
42         return(EXIT_SUCCESS);
43     }
```

- 44 A. 44L
 45 B. 423423
 46 C. 423456
 47 D. *Output is implementation dependent.*
 48 E. *It won't compile because the print function definitions are ambiguous.*
 49

- 1 21. What is the main problem with the following:
 2 (Note 5.11)
 3 `int *ip;`
 4 `for (*ip = 0; *ip < 5; *ip++)`
 5 `;`
 6 A. Nothing is wrong.
 7 B. It dereferences an uninitialized pointer.
 8 C. It does nothing useful.
 9 D. It contains a magic number.
 10 E. It contains implementation dependent problem(s).
- 11
 12 22. Assuming `#define sum(a, b) a + b` predict the value of: `5 * sum(3 + 1, 2)`
 13 (Note 5.18)
 14 A. 30
 15 B. 18
 16 C. 22
 17 D. *none of the above*
 18 E. *implementation dependent*
- 19
 20 23. In C++, what gets printed? (Assume a pointer is printed as a standard integer.)
 21 (Note 6.9)
 22 `void print(int &x, int y, int *z) { x = 10; y = 20; z = (int *)30; }`
 23 `int main()`
 24 `{`
 25 `int a = 1, b = 2, *c = (int *)3;`
 26 `print(a, b, c);`
 27 `cout << a << b << c;`
 28 `return(EXIT_SUCCESS);`
 29 `}`
 30 A. 123
 31 B. 102030
 32 C. 10230
 33 D. 102garbage
 34 E. 1023
- 35
 36 24. If a prototype for `fx` (below) is present, predict the output from: `printf("%d", *fx)`
 37 (Note 6.12)
 38 `int *fx(void)`
 39 `{`
 40 `int x = 5;`
 41 `return(&x);`
 42 `}`
 43 A. 5
 44 B. *garbage*
 45 C. *the address of the variable x*
 46 D. *A compiler error occurs.*
 47 E. *none of the above or implementation dependent*
- 48
 49 25. If `chars` are 8 bits, `ints` are 16 bits, and `int *ip = (int *)20`, predict the value of `++ip`
 50 (Note 6.14)
 51 A. 20
 52 B. 21
 53 C. 22
 54 D. 23
 55 E. *none of the above or implementation dependent*
 56

- 1 26. What is the main problem with the following:
 2 (Note 6.17)
 3 `int ip[] = {6, 7, 2, 4, -5};`
 4 `for (int i = 0; i < 5; ++i, ++ip)`
 5 `cout << *ip;`
 6 A. Nothing is wrong.
 7 B. An uninitialized pointer is being dereferenced.
 8 C. An attempt is being made to modify the name of an array, a constant.
 9 D. It contains a magic number, which is illegal in some compilers.
 10 E. An out of bounds array access occurs.
- 11
 12 27. What is wrong with the following string initialization? `char s[] = {'H', 'E', 'L', 'L', 'O', NULL};`
 13 (Note 7.1)
 14 A. Nothing is wrong.
 15 B. The syntax is incorrect.
 16 C. A character array can't hold a string.
 17 D. `NULL` may be of the wrong type and its value is not necessarily even equal to 0.
 18 E. Strings can't be initialized.
- 19
 20 28. Assuming prototypes and **typedefs** are present, what is wrong with the following?
 21 (Note 8.4; Note 10.3)
 22 `char *cp = malloc(256);`
 23 `FILE *fp = fopen("hello", "a+");`
 24 `fprintf(fp, "Message\n");`
 25 `cp[0] = 'A';`
 26 A. Nothing is wrong.
 27 B. The syntax is incorrect.
 28 C. `cp` is not an array so the form `cp[0]` is not valid.
 29 D. `malloc` and `fopen` are not portable.
 30 E. All file opens and dynamic allocations must be checked before being used.
- 31
 32 29. For `typedef struct {char x; int y;} FOO; FOO bar;` which of the following may be false?
 33 (Note 9.12)
 34 A. `sizeof(FOO) == sizeof(bar)`
 35 B. `sizeof(FOO) == sizeof(x) + sizeof(y)`
 36 C. `&bar` is numerically equal to `&bar.x`
 37 D. `(char *)&bar + offsetof(FOO, y) == (char *)&y`
 38 E. they can all be false, depending upon implementation
- 39
 40 30. What is wrong with the following? (Note 9.12)
 41 `struct Svalues {char x; int y;} s1 = { 25, 30 };`
 42 `class Cvalues {char x; int y;} c1 = { 25, 30 };`
 43 A. Members of the class and the structure have the same names.
 44 B. Public members of a structure are being accessed by an initializer list.
 45 C. Private members of a structure are being accessed by other than a member/friend function.
 46 D. Private members of a class are being accessed by other than a member/friend function.
 47 E. Nothing is wrong.
- 48
 49 31. A file must never be opened in the text mode if:
 50 (Note 10.2)
 51 A. it will be used for binary data.
 52 B. `fprintf` or `cout` will be used to write to the file.
 53 C. the data to be written contains the newline character.
 54 D. the C++ `fstream` methods are to be used.
 55 E. compatibility with modern compilers is desired.
 56

- 1 32. What's wrong with the following:
 2 (Note 10.4)
 3 ofstream fout("file1");
 4 ifstream fin("file2");
 5 fstream fio("file3");
 6 A. The syntax is incorrect for all 3 opens.
 7 B. "file1" may not exist
 8 C. *ifstream*, *ofstream*, and *fstream* opens require two arguments.
 9 D. *fstream* opens require two arguments
 10 E. It's not portable
- 11
- 12 33. On a machine using 1's complement negative integers and 16 bit **ints**, what is the bit pattern for -2?
 13 (Note 11.1)
 14 A. 1111 1111 1111 1111
 15 B. 1111 1111 1111 1110
 16 C. 1111 1111 1111 1101
 17 D. 1000 0000 0000 0010
 18 E. implementation dependent
- 19
- 20 34. If an **int** is 16 bits and a **char** is 8 bits, the values in *sch* and *uch* after *signed char sch = 256;* and *unsigned*
 21 *char uch = 256;* are:
 22 (Note 11.2)
 23 A. *sch* is 256 and *uch* is 256
 24 B. *sch* is implementation defined and *uch* is 256
 25 C. *sch* is implementation defined and *uch* is 0
 26 D. *sch* is 0 and *uch* is 0
 27 E. The results of both are undefined.
- 28
- 29 35. Assuming a 16 bit **int** 2's complement implementation, predict the value of: *-17 >> 1*
 30 (Note 11.7)
 31 A. -9 or 0x7FF7, depending upon the implementation
 32 B. -8
 33 C. 17
 34 D. 8
 35 E. other implementation dependent values
- 36
 37
 38
 39
 40
 41
 42
 43
 44
 45
 46
 47
 48
 49

Pretest Answers

- 50
 51
 52 1A, 2B, 3C, 4D, 5A, 6D, 7B, 8D, 9B, 10D, 11A, 12B, 13A, 14C, 15A, 16B, 17D, 18E, 19D, 20C, 21B, 22B, 23E,
 53 24B, 25C, 26C, 27D, 28E, 29B, 30D, 31A, 32D, 33C, 34C, 35A