

The "C/C++ Programming 2" Pretest

This pretest is not intended to exhaustively test programming ability. Instead, its purpose is to visit some of the basics, to point out some of the "gotchas" and portability issues that permeate C and C++, and to make sure that prospective C/C++ Programming 2 students have at least enough knowledge to grasp the topics covered in the course. It consists of several short code segments and multiple choice questions. You are not expected to immediately know the answers to everything and you may use any reference materials you wish. However, if you don't even understand the requirements/terminology/questions, you should consider taking C/C++ Programming 1 instead.

**PLEASE DO NOT SUBMIT THIS ASSIGNMENT TO THE
"AUTOMATED ASSIGNMENT CHECKER"**

Code Segments:

Your code must not use any magic numbers or external variables and must be completely portable and ANSI/ISO compliant.

1. Write a function-like macro that produces the sum of its two numeric arguments. The macro must be usable as an rvalue in any numeric expression.
2. Write an expression that represents the number of elements in a one-dimensional array named *ABC*, regardless of the size and type of that array.
3. Write a C++ function that has a single type **double** parameter. One of the statements in the function must output the decimal, octal, and hexadecimal representation of the integral part of the function's parameter, all in the same statement. When the function returns it must return that same integral part of the parameter. You may assume that the value of the integral part will not exceed the range of type **long**.
4. Write a C++ function that declares and initializes three type **char** variables to alphabetic characters, then outputs their numeric values.
5. Write a C function to dynamically allocate an array of 350 **floats** as efficiently as possible, where the initial values of the **floats** are unimportant. As is always required, success/failure must be tested.
6. Write a C function to open the file whose name is specified by the function's constant **char** pointer parameter. Open the file in the "binary read and append" mode. As is always required, success/failure must be tested.
7. Write a C function that declares and initializes an array of 250 **doubles** to all 0s in the same statement, then outputs the value of each element starting with element 0. You may use only a compact pointer expression to access the array's elements during outputting.
8. Write a C++ function that contains the definition of a **struct** having the following members: a **private int**, a **protected bool**, the definition of a **public** constant function that has a pointer to an **int** parameter and returns the value of the **private int** member, and the prototype for a **protected** function that has a pointer to a pointer to a **double** parameter and returns **void**.
9. Write a 2-parameter C function that copies the string in its second constant **char** pointer parameter to the character array in its first parameter. Do not call any other functions. You may not use any variables other than the function's two formal parameters.
10. Write the prototypes for 4 C++ single-parameter functions, where all of the functions are legal but only 3 of them can coexist in the same program. Each function must have exactly the same name, the same parameter name, and must return type **int**. The type of the single parameter is up to you and may differ or be the same for each function, as you deem appropriate.

1 **Multiple choice questions:**

2
3 Assume a perfectly implemented ANSI/ISO C or C++ compiler, as appropriate, and realize that how a program
4 runs on your computer only indicates how your computer runs the program, not necessarily how it should run or
5 how portable it is. Therefore, consider all multiple choice questions theoretical since running them as programs
6 can be catastrophically misleading. Below each question is a reference to the location in “C/C++ Notes” where
7 the topic is mentioned or discussed.
8

- 9 1. For C compilers that do not permit “//” style commenting syntax, if variables *x*, *y*, and *z* are properly declared,
10 what is *syntactically* wrong with: `z = y/* division */x;`
11 (Note 1.4)
12 A. Nothing is wrong.
13 B. Everything after the // is a comment so the statement is incomplete.
14 C. It is not portable.
15 D. A comment may not serve as whitespace.
16 E. The value of *y* may be too large.
17
- 18 2. The value of `sizeof('A')` is always:
19 (Note 1.5; Note 2.12)
20 A. the same as the value of `sizeof(char)`.
21 B. the same as `sizeof(int)` in C and the same as `sizeof(char)` in C++.
22 C. 65 if the ASCII character set is used.
23 D. dependent upon the character set being used.
24 E. none of the above.
25
- 26 3. Assuming a 16 bit type `int` and a 32 bit type `long`, the data types of 32767, -32768, 32768, and 2. are:
27 (Note 2.1; Note 2.2)
28 A. **int, int, long, float**
29 B. **int, long, long, float**
30 C. **int, long, long, double**
31 D. implementation dependent
32 E. none of the above
33
- 34 4. Predict the output from `cout << oct << 15 << dec << 15 << hex << 15`
35 (Note 2.6)
36 A. oct 15 dec 15 hex 15
37 B. 017 15 0xf
38 C. 17 15 f
39 D. 1715f
40 E. *Output is implementation dependent.*
41
- 42 5. The values of `-5/4` and `-5%4`, respectively, are:
43 (Note 2.8)
44 A. implementation dependent : `-5/4 == -1` and `-5%4 == -1` or `-5/4 == -2` and `-5%4 == 3`
45 B. -1 and -1
46 C. -2 and 3
47 D. -1 and -2
48 E. none of the above.
49

- 1 6. Assuming `int ax[123]`, the data types of `+(char)65`, `(short)47+(char)65`, `2*6e2L`, and `sizeof(ax)` are:
2 (Note 2.10)
3 A. undefined, undefined, **double**, pointer to **int**
4 B. **char**, **short**, **long**, `size_t`
5 C. **int**, **int**, **long double**, pointer to **int**
6 D. **int**, **int**, **long double**, `size_t`
7 E. implementation dependent
8
- 9 7. What is the value and data type of the expression: `(char)25 < (char)100`
10 (Note 2.10; Note 3.1)
11 A. 1; type **char**
12 B. 1; type **int**
13 C. 75; type **char**
14 D. 0; type **char**
15 E. none of the above
16
- 17 8. Predict what gets printed:
18 (Note 2.14)
19 `const int i;`
20 `for (i = 0; i < 5; ++i)`
21 `cout << i << ' ';`
22 A. 0 1 2 3 4
23 B. 0 1 2 3 4 5
24 C. 1 2 3 4 5
25 D. *It won't compile.*
26 E. *Output is implementation dependent.*
27
- 28 9. Predict what gets printed by: `printf("Goodbye") && printf("Cruel") || printf("World")`
29 (Note 3.2)
30 A. Goodbye
31 B. Goodbye Cruel
32 C. Goodbye Cruel World
33 D. Goodbye World
34 E. *Output is implementation dependent.*
35
- 36 10. For `int x = 1;` predict the value in `x` after: `x = ++x`
37 (Note 3.4)
38 A. 1
39 B. 2
40 C. 3
41 D. undefined
42 E. implementation dependent
43
- 44 11. Predict final value of `i`:
45 (Note 3.10)
46 `for (int i = 0; i < 5; ++i)`
47 `break;`
48 A. 0
49 B. 1
50 C. 2
51 D. 3
52 E. none of the above
53

- 1 12. Predict the value in *x* after: `auto int x = (4, printf("Hello"), sqrt(64.), printf("World"));`
2 (Note 3.11)
3 A. 4
4 B. 5
5 C. 6
6 D. 8
7 E. implementation dependent
8
- 9 13. Predict the output from:
10 (Note 3.15)
11 `if (5 < 4)`
12 `if (6 > 5)`
13 `putchar('1');`
14 `else if (4 > 3)`
15 `putchar('2');`
16 `else`
17 `putchar('3');`
18 `putchar('4');`
19 A. 4
20 B. 2
21 C. 24
22 D. 4 or 24, depending upon the implementation
23 E. Nothing is printed.
24
- 25 14. Predict what gets printed by: `cout << (12 < 5 ? "Hello " : "World")`
26 (Note 3.16)
27 A. Hello
28 B. Hello World
29 C. World
30 D. World Hello
31 E. Output is undefined or implementation dependent.
32
- 33 15. Predict what will happen:
34 (Note 4.3)
35 `char ch;`
36 `while ((ch = cin.get()) != EOF)`
37 `cout.put(ch);`
38 A. A false EOF might be detected or the real EOF might be missed
39 B. It won't compile
40 C. `cin.get()` reads one **int** at a time from input, then its value is printed
41 D. EOF is not defined in C++
42 E. Nothing unwanted happens. It simply reads and prints characters until EOF is reached.
43
- 44 16. Predict what gets printed:
45 (Note 4.6)
46 `cout << __DATE__ << __FILE__ << __LINE__`
47 A. Today's date, the name of the executable file, and the number of lines in the file
48 B. The compilation date, the source file name, and the source file line number containing `__LINE__`
49 C. `cout` cannot use these macros without typecasts
50 D. Output is implementation dependent.
51 E. It won't compile.
52

- 1 17. In C with no prototype, what data types get passed to *fcn* by the call: *fcn((char)23, (short)34, 87, 6.8f)*
 2 (Note 5.5)
 3 A. **char, short, int, float**
 4 B. **char, short, long, float**
 5 C. **int, int, int, float**
 6 D. **int, int, int, double**
 7 E. none of the above or implementation dependent
 8

- 9 18. In C, which statement is true concerning a major problem with the following?
 10 (Note 5.4)

```

11 long double fx(void)
12 {
13     double answer = sum(1.1, 2.2, 3.3);
14     return printf("answer = %f", answer);
15 }
16 double sum(double a, double b, double c)
17 {
18     return(a + b + c);
19 }
    
```

- 20 A. The name *sum* conflicts with a standard ANSI math function.
 21 B. The **return** statement in *fx* returns type **double**.
 22 C. Return statements may not contain an algebraic expression (a + b + c).
 23 D. There is nothing wrong with the program.
 24 E. The call to *sum* assumes that *sum* returns type **int**.
 25

- 26 19. In C++, what gets printed?

```

27 (Note 5.7)
28 void print(int x = 1, int y = 2, int z = 3)
29 {
30     cout << x << y << z;
31 }
32 int main()
33 {
34     print(), print(4), print(5, 6), print(7, 8, 9);
35     return(EXIT_SUCCESS);
36 }
    
```

- 37 A. 123
 38 B. 456789
 39 C. 123456789
 40 D. 123423563789
 41 E. *It won't compile.*
 42

- 43 20. In C++, predict what gets printed?

```

44 (Note 5.8)
45 void print(int x, int y = 2, int z = 3) { cout << x << y << z; }
46 void print(long x, int y = 5, int z = 6) { cout << x << y << z; }
47 int main()
48 {
49     print(4), print(4L);
50     return(EXIT_SUCCESS);
51 }
    
```

- 52 A. 44**L**
 53 B. 423423
 54 C. 423456
 55 D. *Output is implementation dependent.*
 56 E. *It won't compile because the print function definitions are ambiguous.*

- 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.
 18 E. Strings can't be initialized.
- 19
- 20 28. Assuming prototypes and **typedefs** are present for the following C code, what is wrong with it?
 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 9.11)
 34 A. **sizeof(FOO) == sizeof(bar)**
 35 B. **sizeof(FOO) == sizeof(bar.x) + sizeof(bar.y)**
 36 C. **&bar** is numerically equal to **&bar.x**
 37 D. **(char *)&bar + offsetof(FOO, y) == (char *)&bar.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
13

14 The following three questions address topics that are not covered in C/C++ Programming 1, but will be covered in
15 C/C++ Programming 2:
16

- 17 33. On a machine using 1's complement negative integers and 16 bit **ints**, what is the bit pattern for -2?
18 (Note 11.1)
19 A. 1111 1111 1111 1111
20 B. 1111 1111 1111 1110
21 C. 1111 1111 1111 1101
22 D. 1000 0000 0000 0010
23 E. implementation dependent
24
- 25 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*
26 *char uch = 256;* are:
27 (Note 11.2)
28 A. *sch* is 256 and *uch* is 256
29 B. *sch* is implementation defined and *uch* is 256
30 C. *sch* is implementation defined and *uch* is 0
31 D. *sch* is 0 and *uch* is 0
32 E. The results of both are undefined.
33
- 34 35. Assuming a 16 bit **int** 2's complement implementation, predict the value of: *-17 >> 1*
35 (Note 11.7)
36 A. -9 or 0x7FF7, depending upon the implementation
37 B. -8
38 C. 17
39 D. 8
40 E. other implementation dependent values