Answers – C++ Training

# Q1.1

#include <iostream>

int main()

{

std::cout << "hello C/C++" << std::endl;

system("pause");

return 0;

}

# Q1.2

**Output Directory**

Specifies the directory where tools such as the linker will place all final output files that are created during the build process. Typically, this includes the output of tools such as the linker, librarian, or BSCMake.

**Intermediate Directory**

Specifies the directory where tools such as the compiler will place all intermediate files created during the build process. Typically, this includes the output of tools such as the C/C++ compiler, MIDL, and the resource compiler.

**Extensions to Delete on Clean**

The **Clean** option (**Build** menu) deletes files from the intermediate directory where a project's configuration is built. Files with extensions specified with this property will be deleted when **Clean** is run or when you perform a rebuild. In addition to files of these extensions in the intermediate directory, the build system will also delete any known output of the build regardless of where it is located (including intermediate outputs such as .obj files). Note that you can specify wildcard characters.

**Build Log File**

Allows you to specify a non-default location for the log file that is created whenever you build a project.

**Configuration Type**

There are several configuration types from which to choose:

* **Application (.exe)**, displays linker toolset (C/C++ Compiler, MIDL, Resource Compiler, Linker, BSCMake, XML Web Service Proxy Generator, custom build, prebuild, prelink, postbuild events).
* **Dynamic Library (.dll)**, displays linker toolset, specifies /DLL linker option, and adds the \_WINDLL define to CL.
* **Makefile**, displays makefile toolset (NMake).
* **Static Library (.lib)**, displays librarian toolset (same as linker toolset except substitute librarian for linker and omit XML Web Service Proxy Generator).
* **Utility**, displays utility toolset (MIDL, custom build, prebuild, postbuild events).

# Q1.3

The entry point now has arguments. The variable n contains the number of arguments passed to the program. The variable args is a pointer to pointers of strings. Each string is one of the arguments that was passed to the program.

args[0] is always the path to the .exe used to run the program.

# Q1.4

#include <stdio.h>

#include <Windows.h>

int main()

{

SYSTEMTIME time;

const int MILLISECONDS\_PER\_MINUTE = 1000;

const int FRAME\_RATE = 5;

while (true)

{

GetLocalTime(&time);

printf("%02d:%02d:%02d\n", time.wHour, time.wMinute, time.wSecond);

Sleep(MILLISECONDS\_PER\_MINUTE/FRAME\_RATE);

}

return 0;

}

# Q2.1

In function calculate(), count, value are declared static, count is initialized to 10, value is defaulted to 0.

First line shows 0 10, 0 is the value of the variable value, 10 is the value of the variable count.

Second line shows value of i (value) and 11 (count). Because value and count are static, they are not destroyed after out of calculate().

# Q2.2

* Global variable: using global variable is bad.
* Static variable inside function: there is no way to get the counter from outside.
* Static variable outside function: this is kind of using global variable.

# Q2.3

myVar is accessable inside another.cpp, so his first and second way aren’t working. In the first way, there is no myVar has been declared in main.cpp. In the second way, myVar was declared but it not myVar in another.cpp.

The third way is run wells because that’s what extern means to do.

# Q2.4

#include <stdio.h>

const int xyz = 0;

int main(int n, char\*\* args)

{

printf("%d", xyz);

}

# Q3.1

* char has range of values from -128 to 127, so when we cast i (that has value 140 larger than 127) to char we get a negative number (-116) because the exceed amount start from begin of range.
* unsigned char has range of value from 0 to 255, so cast number smaller than 255 we get same value.
* Same as above.
* int j = (c + 256) % 256;

# Q3.2

The result is an infinite loop cause of while condition is an assignment so it’s always true.

# Q4.1

# Q4.2

First line shows pointer a, value that pointer a points to, address of array. Array is a pointer point to itself so first value and third value is the same. The pointer points to the first element of array – 0.

Second line shows value of pointer points to the second element of the array, value that it points to, first value of array + 1 and address of next free memory.

# Q4.3

# Q5.1

First line, ptr\_a and ptr\_b point to same memory stores value 3.

Second line, ptr\_b now points to new memory stores value 9.

Third line, ptr\_a and ptr\_b point to different memories but store same value 3.

Forth line, the memory that ptr\_a points to was freed (delete ptr\_a) then ptr\_a and ptr\_b point to same memory stores value 3.

Fifth line, ptr\_c now points to ptr\_a so this line shows the value of ptr\_c – the address of ptr\_a and value that ptr\_c points to – the value of ptr\_a.

# Q5.2

Cannot delete a static array, it will be released when goes out of the scope.

Solution: remove first 2 delete statements;

# Q5.3

* Memory which is dynamic allocated is not automatically freed. So we must free it manually using delete.
* The delete operator deallocates memory and calls the destructor for a single object created with new.
* The delete [] operator deallocates memory and calls destructors for an array of objects created with new [].
* Using delete on a pointer returned by new [] or delete [] on a pointer returned by new results in undefined behavior.

# Q5.4

MAX returns min.

sizeof(a) is 40, so Exception thrown: write access violation.

Solution: replace sizeof(a) with sizeof(a)/sizeof(int), change “<” to “>”

# Q6.1

# Q7.1

# Q7.2

* Size of struct is 8
* Size of struct is 12. Data alignment means putting the data at a memory address equal to some multiple of the word size. After dow, a data structure padding is added to fill the word, year fits in next word, day and month in next word so that struct uses 3 words = 12 bytes.
* Size of struct is 7 in both cases because 1 byte alignment is small enough to fit any data.

# Q7.3

# Q8.1

# Q8.2

# Q8.4