7/20/2020

Contents

[1. Chapter 1. Getting Started 1](#_Toc62891754)

[1.1. Writing a Simple C++ Program 1](#_Toc62891755)

[1.2. A First Look at Input/Output 1](#_Toc62891756)

# Chapter 1. Getting Started

## Writing a Simple C++ Program

Q: What is the difference between g++ and gcc.

A:

**Exercise 1.1**. Exercise 1.1: Review the documentation for your compiler and determine

what file naming convention it uses. Compile and run the main program from page 2.

≫ g++ -o prog\_01\_01 prog\_01\_01.cc

Running prog\_01\_01.exe gives



**Exercise 1.2**: Change the program to return -1. A return value of -1 is

often treated as an indicator that the program failed. Recompile and rerun

your program to see how your system treats a failure indicator from main.

Answer: The system shows failure indicator as -1 when using echo



## A First Look at Input/Output

7/12/2020

Question: MATLAB has printf for output. C also has input output statements. Similarly C++ has a statement for I/O.

Yes

No

Q: The term stream in iostream library and types ‘istream’ and ‘ostream’, stands for which of the following:

1. A channel for reading input and output
2. Characters are generated, or ingested, sequentially over time.
3. All characters are available at the same time

Correction Answer: B

Exercise 1.6

Answer: Remove the semicolons, when the statement is not termiated.

Excercies 1.9

Sum from 50 to 100 = (100)(101)/2 – (49)(50)/2 = 3825

7/12/2020

Excerise 1.12, Execerise 1.13

Exercise 1.14

Compare and contrast the loops that used a for with those

using a while. Are there advantages or disadvantages to using either form?

Answer:

A **while loop** will always evaluate the condition first.

while (condition) {

//gets executed after condition is checked

}

The main difference between the for's and the while's is a matter of pragmatics: we usually use for when there is a known number of iterations, and use while constructs when the number of iterations in not known in advance. The while vs do ... while issue is also of pragmatics, the second executes the instructions once at start, and afterwards it behaves just like the simple while.

For

void fun\_for()

{

double x=0;

int maxI = pow(2,31) - 1;

for (int i=0; i<maxI; i++)

{

x++;

}

}

void fun\_while()

{

double x=0,i=0;

int maxI = pow(2,31) - 1;

while(i<maxI)

{

i++;

}

}

The run-time was random, 4s or 5s for either For or While loop.

Exercise 1.16

Familiarize yourself with common errors of the compiler.

7/18/20

Exercise 2.1 What is the difference between int, long, long long, short

|  |  |  |
| --- | --- | --- |
| Type | Length on my PC | Max Value |
| int | 4 bytes = 32 bits |  |
| long | 4 bytes = 32 bits |  |
| Long long | 8 bytes = 64 bits |  |
| short | 2 bytes = 16 bits |  |
| Float | 4 bytes = 32 bits | 7 significant digits  IEEE 754 Representation  ±1.18 x 10-38 to ±3.4 x 1038 |
| Double | 8 bytes = 64 bits | 16 significant digits. |

1 char is 1 byte = 8 bits = 2^8 – 1 = 256 – 1.

Exercise 2.2. To calculate a mortgage payment, what types would you use

for the rate, principal, and payment? Explain why you selected each type.

Key Insight: Rounding errors occur when a number can’t be stored precisely. This can happen even with simple numbers, like 0.1. Therefore, rounding errors can, and do, happen all the time. Rounding errors aren’t the exception -- they’re the rule. Never assume your floating point numbers are exact.

A corollary of this rule is: be wary of using floating point numbers for financial or currency data.

Rate: 12.5 or 3.25. Usually 2 significant digits. Float is more than enough.

Principal. 100,000.21, Use double. The cost of double is negligible. Double operations are faster than single.

Principal: Use double because it is used to compute the payment

Payment: Only 2 significant digits are required, so you can get away with float, if memory is really a concern.

When do you use float and when do you use double?

The default choice for a floating-point type should be double. This is also the type that you get with floating-point literals without a suffix or (in C) standard functions that operate on floating point numbers (e.g. exp, sin, etc.).

float should only be used if you need to operate on a lot of floating-point numbers (think in the order of thousands or more) and analysis of the algorithm has shown that the reduced range and accuracy don't pose a problem.

long double can be used if you need more range or accuracy than double, and if it provides this on your target platform.

You can check the famous article by D. Goldberg ("What Every Computer Scientist Should Know About Floating-Point Arithmetic"). You should think twice before using floating-point arithmetic. There is a pretty big chance they are not needed at all in your particular situation.

<http://perso.ens-lyon.fr/jean-michel.muller/goldberg.pdf>

In summary, float and long double should be reserved for use by the specialists, with double for "every-day" use.

Exercise 2.3

What output will the following code produce?

unsigned u = 10, u2 = 42;

std::cout << u2 - u << std::endl; (Answer: 42 – 10 = 32)… Correct

std::cout << u - u2 << std::endl;

Answer:

10 – 42 = -32 as an ‘unsigned’. So wrap around. 256-32 = 224.Wrong

**Correction**:

=pow(2, 32) -32 = 4294967296 – 32 = 4,294,967,264

int i = 10, i2 = 42;

std::cout << i2 - i << std::endl; (Answer = 42 – 10 = 32) Correct

std::cout << i - i2 << std::endl; (Answer = 10 – 42 = -32) Correct

std::cout << i - u << std::endl;

10 – 42 = -32 converted to unsigned = 224 (Wrong)

Correction:

10 – 10 = 0. OK for unssigned

std::cout << u - i << std::endl;

i is converted to unsigned, 10 – 10 = 0. OK. (Correct)

7/20/20

Exercise 2.5

Explain the value of each literal

1. ‘a’ is a character. L’a’ is a wide character and type is wchar\_t. “a” is a string, L”a” is a wide character string of type wchar\_t
2. 10 is of type integer which is 4 bytes , 10u is unsigned integer, 10L is long integer with 8 bytes, 10uL is unsigned long integer 8 bytes, 012 is octal, 0xC is hexadecimal format.
3. 3.14 is a floating point literal of type double, 3.14f is a floating point literal, 3.14L is a long double

Here are some examples of legal and illegal floating point literals

3.14159 //Legal

314159E-5 //Legal

510E //Illegal because of incomplete exponent

210f //Illegal because of the missing decimal point

.e55 //Illegal because of missing integer or fraction

Exersise 2.6. What if any are the differences between:

int month = 9 is the number 9, day = 7 is the decimal number 7;

int month = 09 is the octal9=which is illegal, day = 07 is the octal 7;

Exercise 2.7

What values do these literals represent? What type does each

have?

(a) "Who goes with F\145rgus?\012". Answer: \145 ocatal is character = e. \012 is octal 12 value which is linefeed or newline. What does with Fergus? \n.

(b) 3.14e1L. Answer= 31.4

(c) 1024f. Answer = floating point 1024.0

(d) 3.14L. Answer = 3.14 represented by long double i.e. 8 bytes

7/21/2020

**Exercise 2.9:** Explain the following definitions. For those that are illegal,

explain what’s wrong and how to correct it.

**(a)** std::cin >> int input\_value; Answer: Not OK. Int cannot appear in c

**(b)** int i = { 3.14 }; Answer: Not OK. Trying to initialize a float or double to int. Complier will complain and wrong value will get stored.

**(c)** double salary = wage = 9999.99; Answer: Not OK. You cannot initialize

**(d)** int i = 3.14; Answer: Not OK. Trying to initialize a float or double to int. Complier might complain and wrong value will get stored.

**Exercise 2.10:** What are the initial values, if any, of each of the following

variables?

std::string global\_str;

int global\_int;

int main()

{

int local\_int;

std::string local\_str;

}

Answer:

global\_int will be initialized to zero since it is outside the loop

local int will also be initialized to zero since it is not inside a function

local\_str will be empty string

7/22/20

**Exercise 2.11:** Explain whether each of the following is a declaration or a

definition:

**(a)** extern int ix = 1024; Answer: definition

**(b)** int iy; Answer: definition…..it allocates memory and initializes to zero.

**(c)** extern int iz; Answer: declares but does not define iz

Exercise 2.12: Which, if any, of the following names are invalid?

(a) int double = 3.14; Answer: Valid

(b) int \_; Answer: ~~invalid because \_ cannot be a variable~~. OK to have \_ as variable

(c) int catch-22; Answer: invalid because hyphen is not allowed

(d) int 1\_or\_2 = 1; Answer: Valid because identifiers can be letters, numbers, underscore

(e) double Double = 3.14; Answer: Invalid because Double is a reserved word

Exercise 2.13: What is the value of j in the following program?

int i = 42;

int main()

{

int i = 100;

int j = i;

}

Answer: j=100. (Correct)

Exercise 2.14: Is the following program legal? If so, what values are printed?

int i = 100, sum = 0;

for (int i = 0; i != 10; ++i)

sum += i;

std::cout << i << " " << sum << std::endl;

Answer:

Yes. Scope of i=0 is local. Runs from 0 to 9 with sum = n\*(n+1)/2 = 9\*10/2 = 45.

7/23/20

Exercise 2.15

Which of the following definitions, if any, are invalid? Why?

(a) int ival = 1.01; Answer: Well, it is invalid because ival will store value = 1.

(b) int &rval1 = 1.01; Answer: invalid. Referenced variable cannot store an int value.

(c) int &rval2 = ival; Answer: valid. Rval2 will store the value 1

(d) int &rval3; Answer: invalid. declared as reference but not initialized.

Exercise 2.16

Which, if any, of the following assignments are invalid? If they are valid, explain what they do.

int i = 0, &r1 = i; double d = 0, &r2 = d;

(a) r2 = 3.14159; Answer: Valid. The value of d is set to 3.14159

(b) r2 = r1; Answer: Valid, r2 which is alias for d is set to value of I, d=0;

(c) i = r2; Answer: Valid. The ‘i’ will which is indirectly referring to do ‘d’ is set to 0. i=0.

(d) r1 = d; Answer: Invalid because different types are set. Value of i is set to d = 0;

Exercise 2.17 What does the following code print?

int i, &ri = i;

i = 5; ri = 10;

std::cout << i << " " << ri << std::endl;

Answer: 10 10. i is set to 10 because ri indirectly sets it to 10. ri also referes to the same value of 10.

7/24/20

Exercise 2.19 Differences between pointers and references

|  |  |
| --- | --- |
| References | Pointer |
| Not an object | Object in its own right |
| Must be initialized | Need not be initialized |
| Once defined, no way to make it refer to a different object |  |
| When we use a reference, we always get the object to  which the reference was initially bound | There is no such identity between a pointer and the address that it holds |
|  |  |
|  |  |

Exercise 2.20: What does the following program do?

int i = 42;

int \*p1 = &i;

\*p1 = \*p1 \* \*p1;

Answer: Value of i = 42\*42 = 1764

Exercise 2.21: Explain each of the following definitions. Indicate whether any are illegal and, if so, why.

int i = 0;

(a) double\* dp = &i; //Answer: illegal because types differ

(b) int \*ip = i; // Answer: illegal because ip should hold address. i is not an address.

(c) int \*p = &i; // Answer: Legal

Exercise 2.22: Assuming p is a pointer to int, explain the following code:

if (p) // ...

if (\*p) // ...

Answer: When p is a pointer to integer, it is just a declaration. Unless the pointer is initialized to an object, its value is undefined and p holds any random memory location.

Multiple run of this program



Now, close the cmd line. Reopen…. Points to same location in the memory.

Exercise 2.23: Given a pointer p, can you determine whether p points to a valid object? If so, how? If not, why not?

No, you can't. Why? Because it would be expensive to maintain meta data about what constitutes a valid pointer and what doesn't, and in C++ you don't pay for what you don't want.

And you don't *want* to check whether a pointer is valid, because you *know* where a pointer comes from, either because it's a private part of your code that you control, or because you specified it in your external-facing contracts.

Exercise 2.24: Why is the initialization of p legal but that of lp illegal?

int *i* = 42; void \*p = &i; long \*lp = &i;

Answer:

Void can point to any object type. But lp is long type and int and long type don’t mix. The type void \* is a special pointer type that holds the address of any object. Type of object is unknown.

**Generally, we use void \* pointer to deal with memory as memory, rather than using the pointer to access the object.**

Exercise 2.25: Determine the types and values of each of the following

variables.

(a) int\* ip, &r = ip;

(b) int i, \*ip = 0;

(c) int\* ip, ip2;

Answer:

1. ip is a pointer to integer. &r is a an alias to the pointer.
2. i if of type integer an ip is a pointer to integer and null pointer
3. ip is a pointer to integer and ip is an integer

7-27-2020

Exercises Section 2.4

Exercise 2.26: Which of the following are legal? For those that are illegal,

explain why.

(a) const int buf; Answer: Illegal. Not initialized

(b) int cnt = 0; Answer: Legal.

(c) const int sz = cnt; Answer: Legal.

(d) ++cnt; ++sz; Answer:++cnt is legal. ++sz is illegal because sz is read-only variable.

8/11/20

**Exercise 2.27:** Which of the following initializations are legal? Explain why.

**(a)** int i = -1, &r = 0; Answer: i is legal. 2nd one is illegal because cannot bind non-constant value of type int& to 0, which is of type const int.

**(b)** int \*const p2 = &i2; Answer: Illegal because i2 is not defined.

**(c)** const int i = -1, &r = 0; Answer: Legal. Legal because 0 is of type const int and r is of type.

**(d)** const int \*const p3 = &i2; Answer: Legal because p3 is const type pointer that can hold address of i2. But i2 is not defined

**(e)** const int \*p1 = &i2; Legal. But i2 is not defined

**(f)** const int &const r2; Illegal.

**(g)** const int i2 = i, &r = i; Answer: Legal, Legal.