**Please keep only the 🡺 portions of the document with your answers.** **Kindly delete the instructions and notes.   
 −10% cost if you make me wade through all the stuff you were asked to delete.**

Some find it easier to create a new document (Ctrl+N) and copy the 🡺 portions from here to there.

🡺 Your name: Student No.: UserID: ­­­­\_\_\_\_\_ @mySeneca.ca

**Activity 1** of 3 – **integer overflow**

Imagine a stopwatch timer which counts in hundredths of a second. Start it.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| **DAYS** | **HOURS** | **MINUTES** | **SECONDS** | **hundredths** |

🡺 If the timer value is stored in a signed **long** 32-bit integer,   
how many **days**, to two decimals, will it take until that integer overflows? **(5 points)**

🡺 What are the maximum and minimum values that can be stored in a **short** 16-bit signed integer? (2.5 points)

16-bit signed integer maximum = ????? … minimum = ?????

🡺 Give examples of values that would cause overflow in positive and negative directions when two **short** 16-bit signed integers are added together. **(5 points)**

????? + ????? are two positive **short** values causing overflow when added together.

????? + ????? are two negative **short** values causing overflow when added together.

[ play with the integerOverflow.exe program in the zip file. It requires the .NET virtual machine so ensure Visual Studio IDE has been installed on your computer or launched on the lab PC. ]

Binary Search Bug

FYI Here is an animation of a binary search which walks through the C code:

<https://www.cs.usfca.edu/~galles/visualization/Search.html>   
[ note: numeric keypad input does not work, use the keyboard's top row ]  
[ search: for the value found at index position 20 ]  
[ controls at the bottom allow you to adjust animation speed and stepping ]

The following line of code was used for a long time in many standard programming libraries to find the middle point in a range of array indices for a binary search:

**mid = (low + high) / 2;** // find mid-point in a range of values

**Your task is to identify and fix the bug in that calculation of mid.**

🡺 What is potentially wrong with the **(low + high) / 2** calculation to find the middle point? Under what conditions would the calculation go wrong? **(7.5 points)**For a demonstration of the problem, run **MidBugTest.exe** found in this week's zip file.

🡺 REWRITE the code to prevent overflow **(10 points)***from* **mid = (low + high) / 2;***to* mid = ;

Note: **mid**, **low**, and **high** are all variables of the same numeric data type; an arithmetic operation on two variables of the same type always returns a value of that type. The type is likely **int** and would be declared in the binary\_search() library function where our problematic **mid** calculation is found. The size of an **int** varies depending on the platform's C compiler. Assume that the size of an integer is sufficient for the binary search function. The problem here is not about the data type, its size, or bit width. **Casting is not allowed. The task is to *rewrite the arithmetic* so that regardless of the variables' data type, the calculation *cannot* overflow.**

**Hint**: The range of low to high values may appear to fit within the range of an integer data type.  
0-------------------------low--------------high---------INT\_MAX

Instead of finding the average of **(low + high) / 2** as in the bug version,  
0-------------------------low0------------------------------------------high  
| ^ |  
  
first, find the middle point between low and high.  
0-------------------------low--------------high---------INT\_MAX   
 low--------------high   
 | **^** |

**Casting** is computationally expensive and merely avoids the problem. Casting an intermediate result to a higher capacity data type will work reliably only if you cast from **int** to **long long**. That is the good news. The bad news is you are *avoiding* the bug, not solving it. Your children will be faced with the same problem when a 64-bit sized **int** becomes the default. You do not want them fixing this by casting to **int128\_t** because then your grandchildren will inherit the problem. When will it end?

🡺 Describe the steps you used to develop and test your solution to the binary search bug. **For the full 20 points**, what were the details of your process from problem analysis to solution implementation? (This is like the reflection component in your C course workshop.)

A C program to test your new formula, **MidBugTest.c** is found in this week's zip file. The code demonstrates the bug. Change the **mid =** line of code to your new formula, compile, and run.

When you are done (or just done in), read this: <https://research.googleblog.com/2006/06/extra-extra-read-all-about-it-nearly.html> “Nearly All Binary Searches and Mergesorts are Broken” –– *only one of the suggested solutions in this article works in* ***all*** *cases*. Your code should always be portable across platforms meaning it must work in all cases.

**Activity 2** of 3 – **Boolean logic (25 points)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Relational operators** | **C language expression** |  | **Logical Operators** | **C language** |
| **Equal to** | a **==** b |  | [negation](https://en.wikipedia.org/wiki/Negation) – NOT TRUE | **!(** a == b**)** |
| **Not equal to** | a **!=** b |  | [negation](https://en.wikipedia.org/wiki/Negation) – NOT | a != b |
| **Greater than** | a **>** b |  | [conjunction](https://en.wikipedia.org/wiki/Logical_conjunction) – AND *both expressions are true* | a > b **&&** b > c a *must be >* c |
| **Less than** | a **<** b |  | [disjunction – OR](https://en.wikipedia.org/wiki/Logical_disjunction)  *either expression is true* | a > b **||** b > c a *vs* c *is unknown* |
| **Greater than or equal to** | a **>=** b |  | *order of operations | operator precedence ( implied parentheses shown )*  *(* comparison AND comparison *)*  *(*comparison AND comparison*)* OR *(*comparison AND comparison*)* | |
| **Less than or equal to** | a **<=** b |  |

Note: **A relational expression in programming always compares one single thing to another single thing.** It does not allow testing a range or a list, e.g.   
(startDate <= today <= endDate) works only for mathematics,   
(today == startDate to endDate) and (today != Saturday,Sunday) works for humans, but neither works for C programming.

The most common form of the expression compares a variable to a standard | literal | reference. E.g. today

**🡺 For any given date, what is the Boolean logic to decide if you have to attend school during the current term?**

The answer includes checking the following criteria and the links for details:

* Is today’s date within the [Academic date](http://www.senecacollege.ca/registrar/dates/) range for the first and last day of classes?
* Exceptions: there are no classes during study week,   
  and the college is closed on [holidays](https://www.statutoryholidays.com/ontario.php) within the term.
* Does your timetable have classes on this day of the week? Use today’s day-of-the-week (**DoW**) matched to your timetable’s number of classes on that day-of-the-week.
* You *must* attend during the “Final assessment period” (AKA exam week) which may vary from your regular timetable. See the Academic data link.
* You may have other reasons to be or not to be on campus, e.g. project group meeting or illness.

Use C language [comparison](https://en.wikipedia.org/wiki/Operators_in_C_and_C%2B%2B#Comparison_operators/relational_operators) and [logical](https://en.wikipedia.org/wiki/Operators_in_C_and_C%2B%2B#Logical_operators) operators and write your logic in C syntax.

Assume the following variables contain values for the current day-of-the-week, today's date, and term dates:

* **int DoW =** today's Day of Week number: 1 is Monday, 2 is Tuesday, 3 is Wednesday, 4 is Thursday, 5 is Friday, 6 is Saturday, 7 is Sunday.This is the international standard [ISO 8601](https://en.wikipedia.org/wiki/Determination_of_the_day_of_the_week) where Monday is the first day of the week.
* **int classesDoW[7] =** array containing the number of classes on each Day of Week.
  + Using ISO 8601 **DoW** as an array index is inconvenient for C programming where the first array element starts at zero. [Why zero](https://developerinsider.co/why-does-the-indexing-of-array-start-with-zero-in-c/)?
* **int DoW\_index = DoW - 1;** // ISO 8601 day of week number adapted for C array index, e.g. **classesDoW[DoW\_index]**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Day of Week | Monday | Tuesday | Wednesday | Thursday | Friday | Saturday | Sunday |
| **DoW\_index** | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| **classesDoW** | *n* | *n* | *n* | *n* | *n* | 0 | 0 |

* **long today =** today's calendar date YYYYMMDD
* **long startClasses =** first day of classes for this term YYYYMMDD
* **long endClasses =** last day of classes for this term YYYYMMDD
* **long startStudyWeek =** first day of study week for this term YYYYMMDD
* **long endStudyWeek =** last day of study week for this term YYYYMMDD
* **long startExamWeek =** first day of exam week for this term YYYYMMDD
* **long endExamWeek =** last day of exam week for this term YYYYMMDD
* Winter term holiday dates
  + **long familyDay =** YYYYMMDD // third Monday in February
  + **long goodFriday =** YYYYMMDD // depends on lunar cycle  
    Easter is the first Sunday following the full moon following the Spring equinox.
* Summer term holiday dates
  + **long goodFriday =** YYYYMMDD // depends on lunar cycle
  + **long victoriaDay =** YYYYMMDD // third Monday in May
  + **long canadaDay =** YYYY0701 // first day of July
  + **long civicDay =** YYYYMMDD // first Monday in August
* Fall term holiday dates
  + **long thanksgivingDay =** YYYYMMDD // second Monday in October

There is no need to write a C program. Although some students like to do so and it is always interesting to read a fully programed solution, *there is no need to write a full C program*.

The solution can be written as one long Boolean statement or can be separated into a stepwise progression. Criteria can be tested one at a time and stored in Boolean variables which are named beginning as if they were a question or an affirmative statement with a True or False response. E.g.   
Is today a holiday? isHoliday Do I have classes today? hasClass There is [no perfect naming method](https://dev.to/michi/tips-on-naming-boolean-variables-cleaner-code-35ig) but anything is better than flag1, check2, test3, …

To use the Boolean constants true, false, and data type bool,   
C programs require #include <stdbool.h>

bool isHoliday = false, hasClass = false, *...* ; // declare and init logical flags

if (*decision logic is true*) isHoliday = true;

if (*decision logic is true*) hasClass = true;  
 ...  
 if (hasClass && !isHoliday && ...) printf("GOTO SCHOOL");  
 else printf("stay home");

There is no need to show the progressive resolution of your Boolean expressions to TRUE or FALSE as shown in this week's slides. The solution for this exercise is not the same as presented in in this week's slides.

**Activity 3a and 3b** of 3 – **Numbering Systems and Conversions (15 + 10 points)**

HTML colour codes are expressed in three hexadecimal values like this #0088FF which can represent 16,777,216 colours. **#** indicates the start of hex digit pairs which read as # 00 88 FF pronounced "hex, zero-zero, eight-eight, Fox-Fox." (three hex values)

Hex digits are 0 – 9, Able, Baker, Charlie, Dog, Easy, Fox for 0 – 9,10,11,12,13,14,15 decimal values in a single hex digit.

All of these are equivalent:  
16×16 × 16×16 × 16×16 (6 hex digits or three hex values, #FFFFFF)  
or 256 × 256 × 256 (three byte values)  
or 28  × 28  × 28  (three 8 bit values)  
or 224 (one 24 bit value)  
or 16,777,216 possible decimal values.

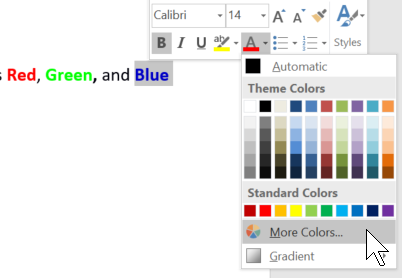
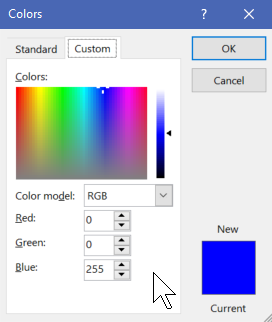
Each hex digit pair represents a range of three colours: **Red #FF0000**, **Green #00FF00**, and **Blue #0000FF** known as RGB values.

See <https://www.rapidtables.com/convert/color/index.html> to convert colour codes between HEX and RGB. To further explore colours, go to <https://www.hexcolortool.com/>. The Windows calculator ( "calc" *o*r + R "calc") has a very handy Programmer function available from the hamburger button.

3a 🡺 What is the hex value for these colours? **(15 points)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Colour** | **Red value** | **Green value** | **Blue value** | **6 digit Hex code** |
| White | 255 | 255 | 255 | # |
| **Grey** | 128 | 128 | 128 | # |
| **Black** | 0 | 0 | 0 | # |
|  | 0 | 211 | 023 | # |
|  | 103 | 110 | 160 | # |

In Microsoft Word, you can edit a font’s colour by adjusting its **Red**, **Green,** and **Blue** values. However, Word uses decimal instead of hex values where each colour value can be any decimal number from 0 to 255 (same range as a hex pair 00 – FF). Using the following Hexadecimal values, determine what colour would be produced in Word.  
  
First convert each HEX pair to decimal and then use Font Color / More Colors…Custom to adjust the RGB decimal values:

3b 🡺 Fill in this chart as per the column headings **(10 points)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **6 digit Hex code** | **Red decimal value (0-255)** | **Green decimal value (0-255)** | **Blue decimal value (0-255)** | **Describe the Final Colour *and* change the cell's background colour, i.e. R-click and see MS Word 'Shading', to match the values for RGB** |
| #FB6905 |  |  |  |  |
| #0AA2C8 |  |  |  |  |
| #E160B7 |  |  |  |  |
| #0B6A0C |  |  |  |  |