**ITT-310 Design Report Template**

|  |  |
| --- | --- |
| **Topic:** | Milestone 4: Application Release 3 |
| **Date:** | *28th October, 2021* |
| **Revision:** | Application Release 3 |
| **Milestone Summary:** | |  |  |  | | --- | --- | --- | | **User Story / Task** | **Hours Worked** | **Hours Remaining** | | **Overview**  In this milestone, students will design and develop the third release of their C programming application. From previous assignments and activities, students should be proficient with the following skills:   1. Writing C programs that use structures and unions   **Execution**  Execute this assignment according to the following guidelines:   1. Update your C application to demonstrate the following features:    1. Use of structures and unions 2. All code should be fully commented with single-line comments for all logic in code statements and multi-line comments for the all functions as well as code modules. 3. Complete test cases using the “Test Case Template,” located in the Course Materials that validates all functionality of the program. Positive, negative, and abuse test cases MUST be included in the assignment. 4. Update the technical design report with the required design documentation. 5. Complete a screencast, using Loom or a similar free application, that contains a technical walk through of all code and a functional demonstration of the code running on your system. |  |  | |  |  |  | |  |  |  | |
| **GIT URL:** | [masoncota1/Milestone-4-Application-Release-3: I designed and developed he third release of their C programming application. (github.com)](https://github.com/masoncota1/Milestone-4-Application-Release-3) |

**Design Documentation**

# Milestone 4: Application Release 3

**Overview**

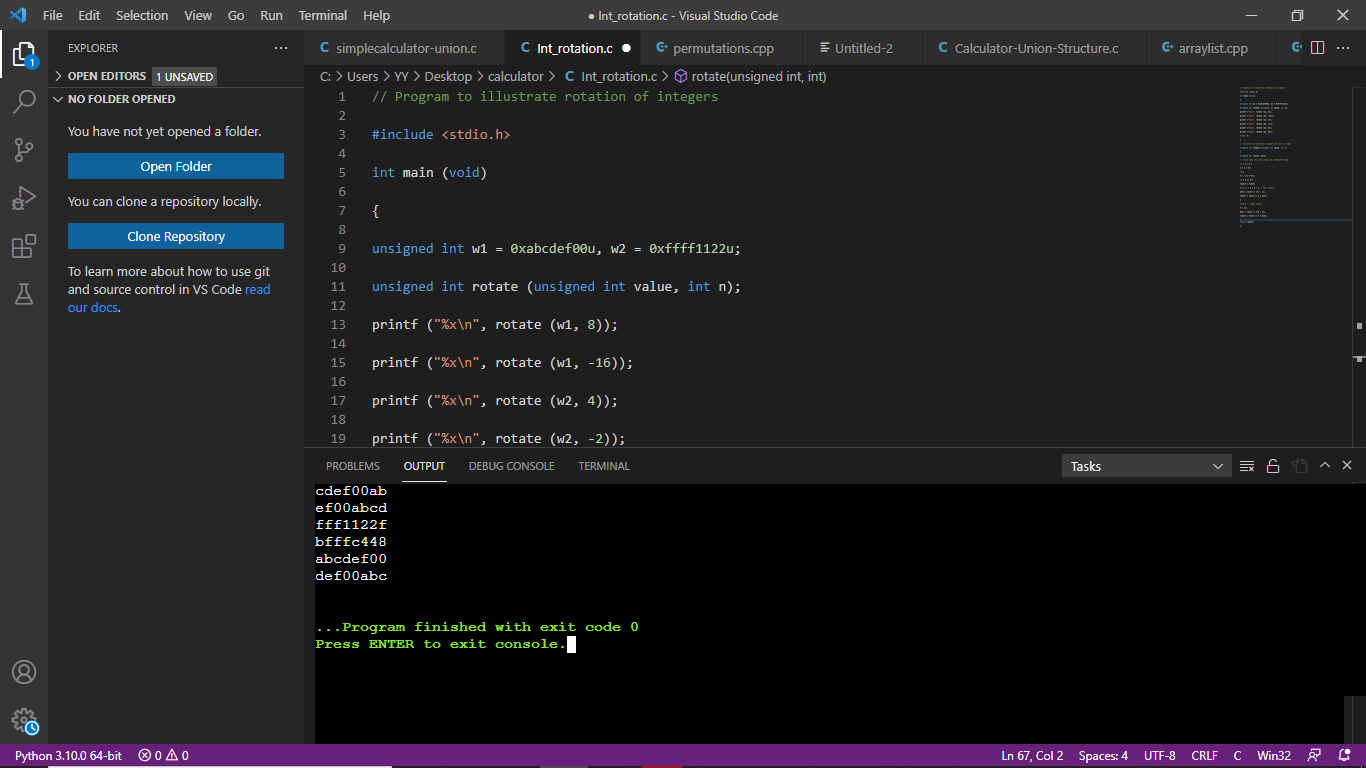
In this milestone, students will design and develop the third release of their C programming application. From previous assignments and activities, students should be proficient with the following skills:

1. Writing C programs that use structures and unions

**Execution**

Execute this assignment according to the following guidelines:

1. Update your C application to demonstrate the following features:
   1. Use of structures and unions

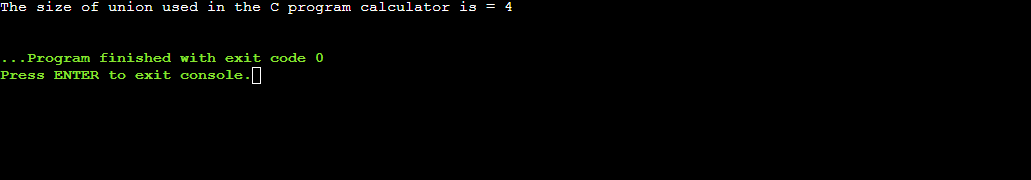


**Union Code Run Separately**

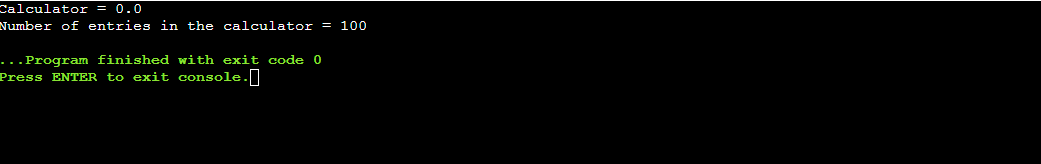
1. All code should be fully commented with single-line comments for all logic in code statements and multi-line comments for the all functions as well as code modules.
2. Complete test cases using the “Test Case Template,” located in the Course Materials that validates all functionality of the program. Positive, negative, and abuse test cases MUST be included in the assignment.

**Test case 1**

**Union and Structure Code run together**

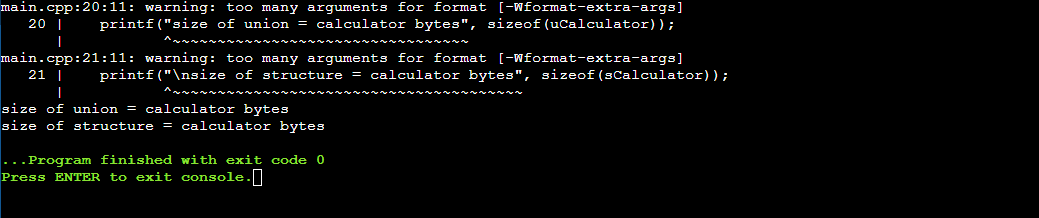


Union Case run separately



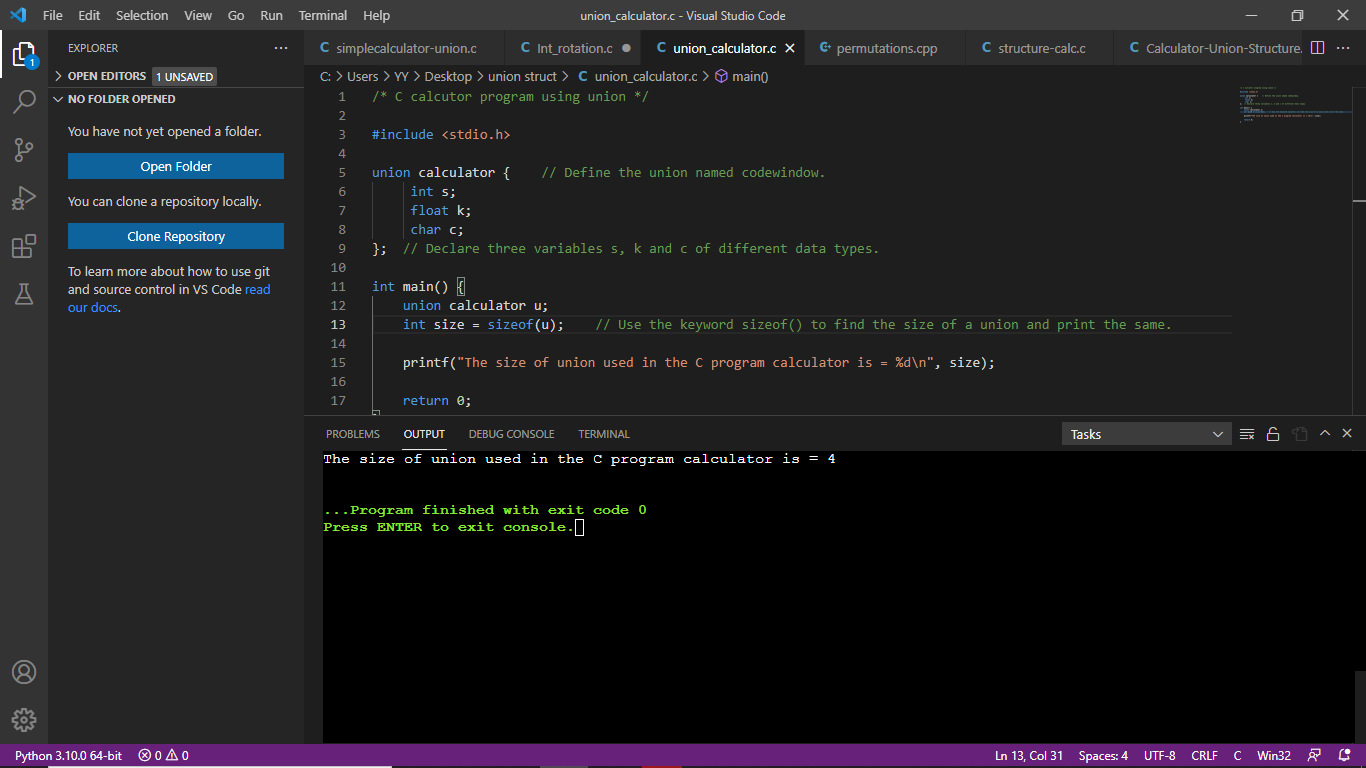
**Test case 2 (Negative)**

**Code displays root errors when running**

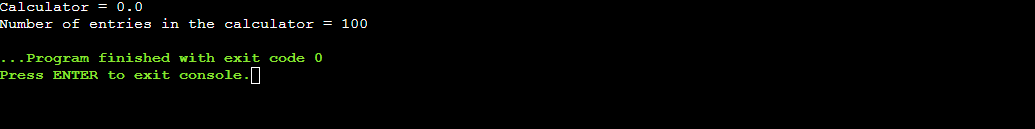
****

**Test case 3**

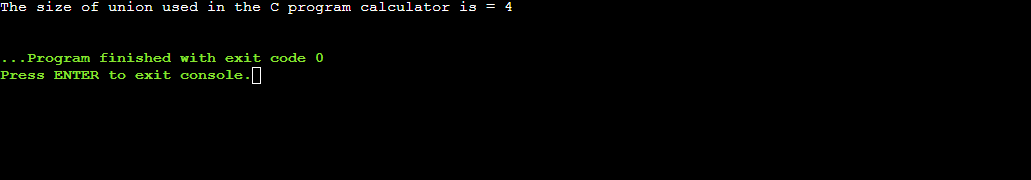
***Union Code Run Separately***

******

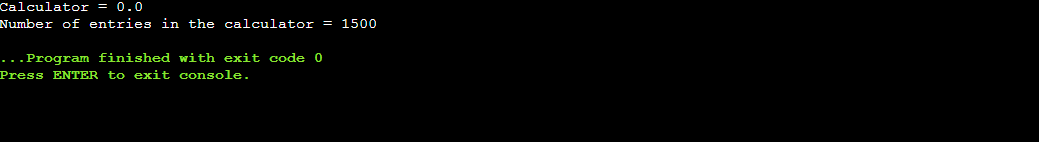
***Output***

******

***Number of Assigned value is changed. The code runs successfully***



***Output***

****

1. Update the technical design report with the required design documentation.

Union is a data type in C programming that allows different data types to be stored in the same memory locations. Union provides an efficient way of reusing the memory location, as only one of its members can be accessed at a time. A union is used almost in the same way you would declare and use a structure.

#### HOW THE STRUCTURE AND UNION HAVE BEEN USED IN C:

|  |  |
| --- | --- |
| **C Structure** | **C Union** |
| Structure allocates storage space for all its members separately. | Union allocates one common storage space for all its members. Union finds that which of its member needs high storage space over other members and allocates that much space |
| Structure occupies higher memory space. | Union occupies lower memory space over structure. |
| We can access all members of structure at a time. | We can access only one member of union at a time. |
| Structure example: struct student { int mark; char name[6]; double average; }; | Union example: union student { int mark; char name[6]; double average; }; |
| For above structure, memory allocation will be like below. int mark – 2B char name[6] – 6B double average – 8B  Total memory allocation = 2+6+8 = 16 Bytes | For above union, only 8 bytes of memory will be allocated since double data type will occupy maximum space of memory over other data types.  Total memory allocation = 8 Bytes |

1. Complete a screencast, using Loom or a similar free application,that contains a technical walk through of all code and a functional demonstration of the code running on your system.

**Code demonstration of Structures**

#include <stdio.h>

struct calculator {    // Define the structure named calculator.

     int s;

     float k;

     char c;

};  // Declare three variables s, k and c of different data types.

int main() {

    struct calculator u;

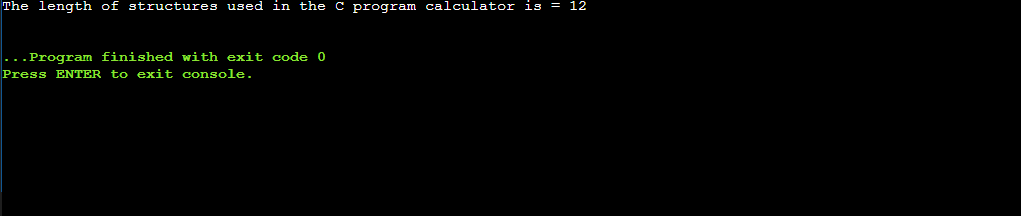
    int size = sizeof(u);    // Use the keyword sizeof() to find the size of a structure and print the same.

    printf("The length of structures used in the C program calculator is = %d\n", size);

    return 0;

}

**Output**



**NOTES TO EXPLAIN THE CODE IN THE LOOM VIDEO**

**How the union and structure variables and functions work in the code**

Here, the size of sCalculator is 40 bytes because

* the size of name[32] is 32 bytes
* the size of calculator is 4 bytes
* the size of CalculatorNo is 4

However, the size of uCalculator is 32 bytes. It's because the size of a union variable will always be the size of its largest element. In the above example, the size of its largest element, (name[32]), is 32 bytes.

**Code demonstration of Union**

#include <stdio.h>

union unionCalculator

{

   //defining a union

   char name[32];

   float calculator;

   int calculatorNo;

} uCalculator;

struct structCalculator

{

    //defining a Structure

   char name[32];

   float calculator;

   int calculatorNoNo;

} sCalculator;

int main()

{

   printf("size of union = %d calculator bytes", sizeof(uCalculator));

   printf("\nsize of structure = %d calculator bytes", sizeof(sCalculator));

   return 0;

}

***Output***

