**Derived data types in C++**

The coursework-2 is based on derived data types in C++. This document contains all the tasks to be done along with the basic concepts required for completing this assignment. The student is required to go through the materials of the week3 and week4 as well before attempting this. For your convenience only, some of the fundamental concepts of arrays, pointers, functions and structures are given as a revision in the beginning but make sure you go through the lectures and tutorials of the week3 and week4 learning materials as well as reference URLs mentioned in the handout notes.

The marking scheme for this session is as follows:

* Task 1: 50 marks
* Task 2: 20 marks
* Task 3: 30 marks
* Task 4: 0 marks

**Arrays in C++**

An array is a derived data type. It is an ordered sequence of finite data items of the same data type that shares a common name. The common name is the **array name** and each individual data item is known as an **element** of the array. The elements of the array are stored in the subsequent memory locations starting from the memory location given by the array name.

A one-dimensional array can be used to represent a list of data items and it is also known as a **vector**. A two-dimensional array can be used to represent a table of data items consisting of rows and columns and is also known as a **matrix**.

**How are arrays declared?**

One-dimensional arrays are declared with a pair of square brackets, two-dimensional arrays with 2 pairs of square brackets, and so on. For example,

int x[10]; /\* One-dimensional array \*/

float a[5][5]; /\*Two-dimensional array \*/

int p[6][4][3]; /\* Three-dimensional array \*/

**What is array initialization?**

If the elements of an array are initialized in the declaration itself, it is known as array initialization.

The values are enclosed within { } and all the rows are enclosed within { and }. For example,

int a[4] = { 3, 5, -8, 10 };

int x[3][2] = {{20, 5}, {-8, 5), {5, 7}};

char t[] = {'a', 'e', 'i', 'o', 'u', '\0'};

or

char t[] = “aeiou”;

The elements that are not initialized are set to zero.

**How are the two-dimensional array elements stored in memory?**

Two-dimensional arrays follow row major order storage representation. The elements are stored in row by row in the subsequent memory locations.

NOTE:

Array elements are accessed through index numbers [Index numbers are integers and always starts from 0].

NOTE: How many bytes of memory will the following arrays need?

(a) char s[80]; /\*80 \*/

(b) char s[80][10]; /\*800 \*/

(c) int d[10]; /\* 10\*sizeof(int) \*/

(d) float d[10][5]; /\* 50 \* sizeof(float) \*/

**Pointers**

**What is a pointer value?**

A pointer value is a data object that refers to a memory location.

**What is an address?**

Each memory location is numbered in the memory. The number attached to a memory location is called the address of that location.

**What is a pointer variable?**

A pointer variable is a variable that may contain the address of another variable or any valid address in the memory.

**How is a pointer variable declared?**

Format:

data\_type \*varl, \*var2, ..., \*varN;

For example,

int \*pl, \*p2;

Here, p 1 and p2 are pointer variables.

**Which are the operators exclusively used with pointers?**

Address of operator ( & ) and indirection operator ( \* ). The operator & returns the address of its operand. The operator \* returns the value pointed by its operand.

**Give the syntax for using an address of operator.**

The address of operator returns the address of its operand. The operand must be a named region of storage like int variable, float variable, etc. for which a value may be assigned. It cannot be a constant or an expression or a register type variable.

Format : ptr variable = &named\_region

For example,

int num, \*p;

p = &num;

**Give the syntax for using an indirection operator.**

The operand must be a pointer expression. The value returned is an lvalue.

Format : \*ptr expression

For example,

int \*p, num = 10;

p = &num;

cout << \*p ; /\* Prints the value 10 \*/

**Difference between pointers and arrays.**

|  |  |  |
| --- | --- | --- |
|  | **Array** | **Pointer** |
| 1 | Array allocates space automatically | Explicitly assigned to point to an allocated space. |
| 2 | It cannot be resized. | It can be resized using realloc( ). |
| 3 | It cannot be reassigned. | Pointer can be reassigned. |
| 4 | sizeof(arrayname) gives the number of bytes occupied by the array. | sizeof(p) returns number of bytes used to store the pointer variable p. |

**C++ Functions**

C++ functions are a group of statements in a single logical unit to perform some specific task.

Along with the main function, a program can have multiple functions that are designed for different operation.

The results of functions can be used throughout the program without concern about the process and the mechanism of the function.

In POP (Procedural Oriented Programming) language like C, programs are divided into

different **functions** but in OOP (Object Oriented Programming) approach program is divided into objects where functions are the components of the object.

Generally, C++ function has three parts:

• Function Prototype

• Function Definition

• Function Call

**C++ Function Prototype**

While writing a program, we can’t use a function without specifying its type or without telling the compiler about it. So before calling a function, we must either declare or define a function.

Thus, declaring a function before calling a function is called **function declaration or prototype** which tells the compiler that at some point of the program we will use the function of the name specified in the prototype.

**Syntax**

return\_type function\_name(parameter\_list);

**Note:**

function prototype must end with a semicolon.

Here, return\_type is the type of value that the function will return. It can be int, float or any user-defined data type.

function\_name means a valid identifier or name that we give to the function.

Finally, parameter\_list contains a total number of arguments that need to be passed to the function.

**C++ Function Call**

**Function call** means calling the function with a statement. When the program encounters the function call statement the specific function is invoked.

**Syntax**

function\_name( argument\_list );

Here, function\_name is the name of the called function and argument\_list is the comma-separated list of expressions that constitute the arguments.

The syntax is similar to that of prototype except that return\_type is not used.

**C++ functions definition**

**Function definition** is a part where we define the operation of a function. It consists of the declarator followed by the function body.

**Syntax**

return\_type function\_name( parameter\_list )

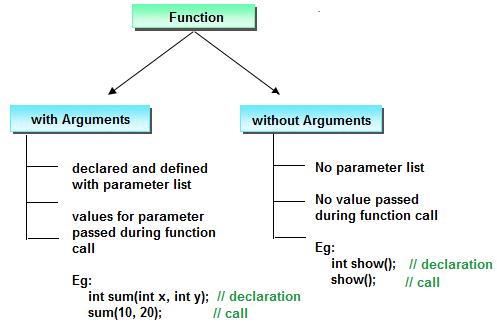
{

function body;

}

**Call by Value and Call by Reference in C++**

On the basis of arguments there are two types of function are available in C++ language, they are;



* With argument
* Without argument

If a function takes any arguments, it must declare variables that accept the values as an arguments. These variables are called the formal parameters of the function. There are two ways to pass value or data to function in C++ language which is given below;

* call by value
* call by reference

For further reading go through the learning materials.

**Structures in C++:**

It is a user defined data type.

Unlike arrays that can have elements of homogeneous data types only, a structure can have elements of both homogeneous as well as heterogeneous data types.

It can be declared globally outside of main() function or locally inside main() function.

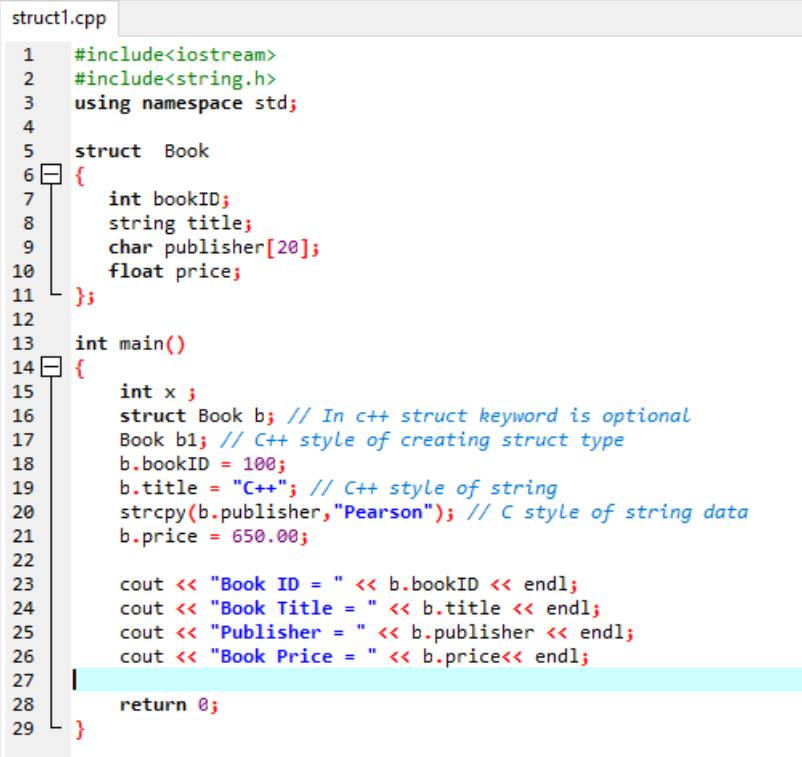
Syntax :

Struct <structure name>

{  
 data members….

};

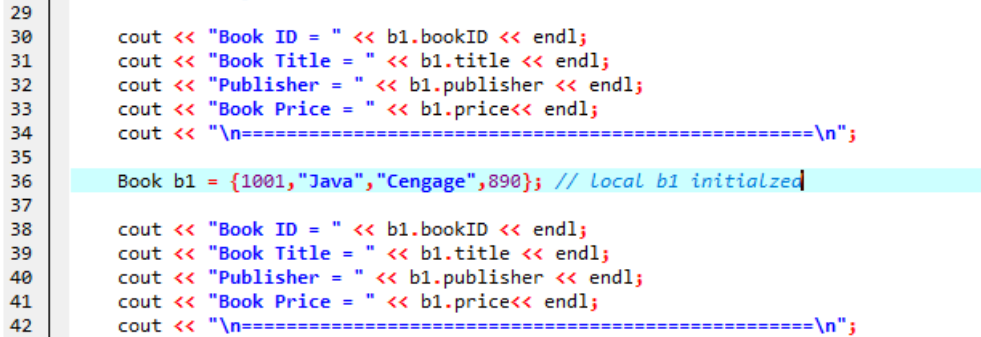
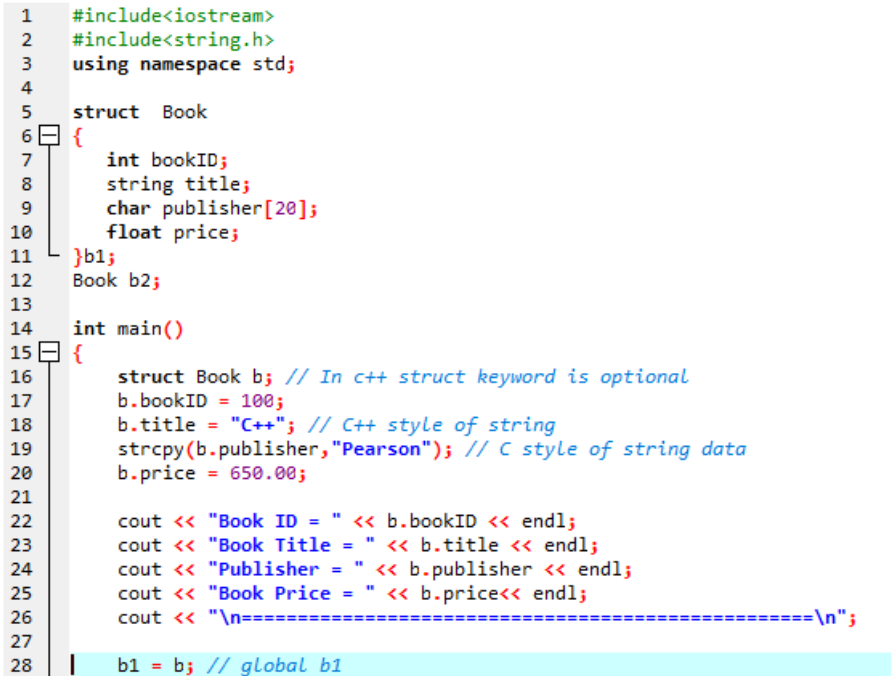
Example:

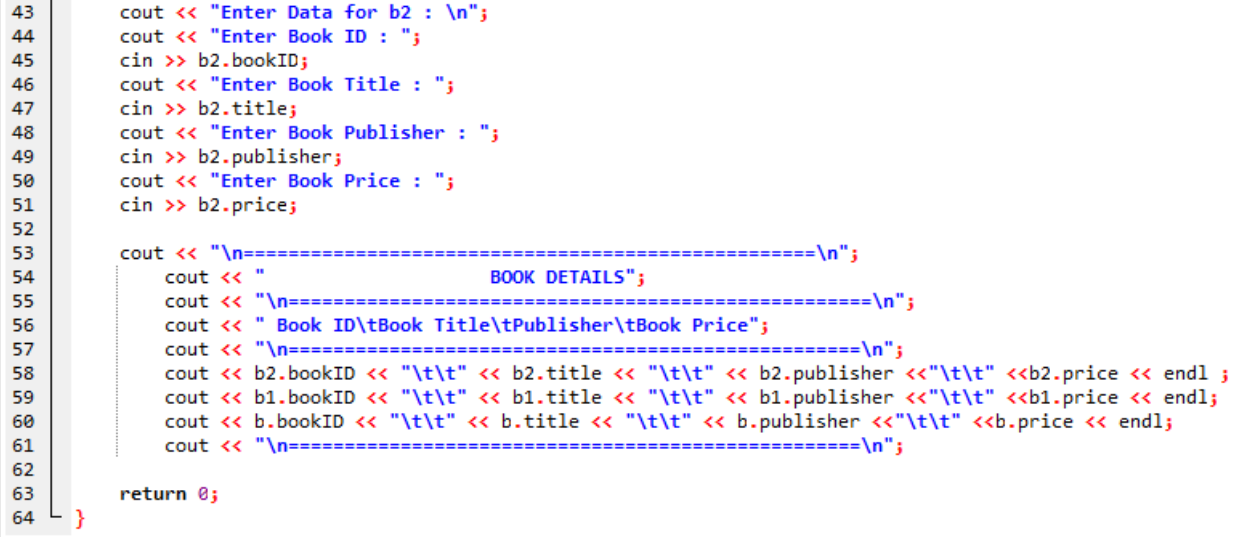


Declaring structure type variable:

We can declare structure types globally with or outside the syntax of the structure or locally inside any function.

We can copy the data of one structure variable into other structure variable .

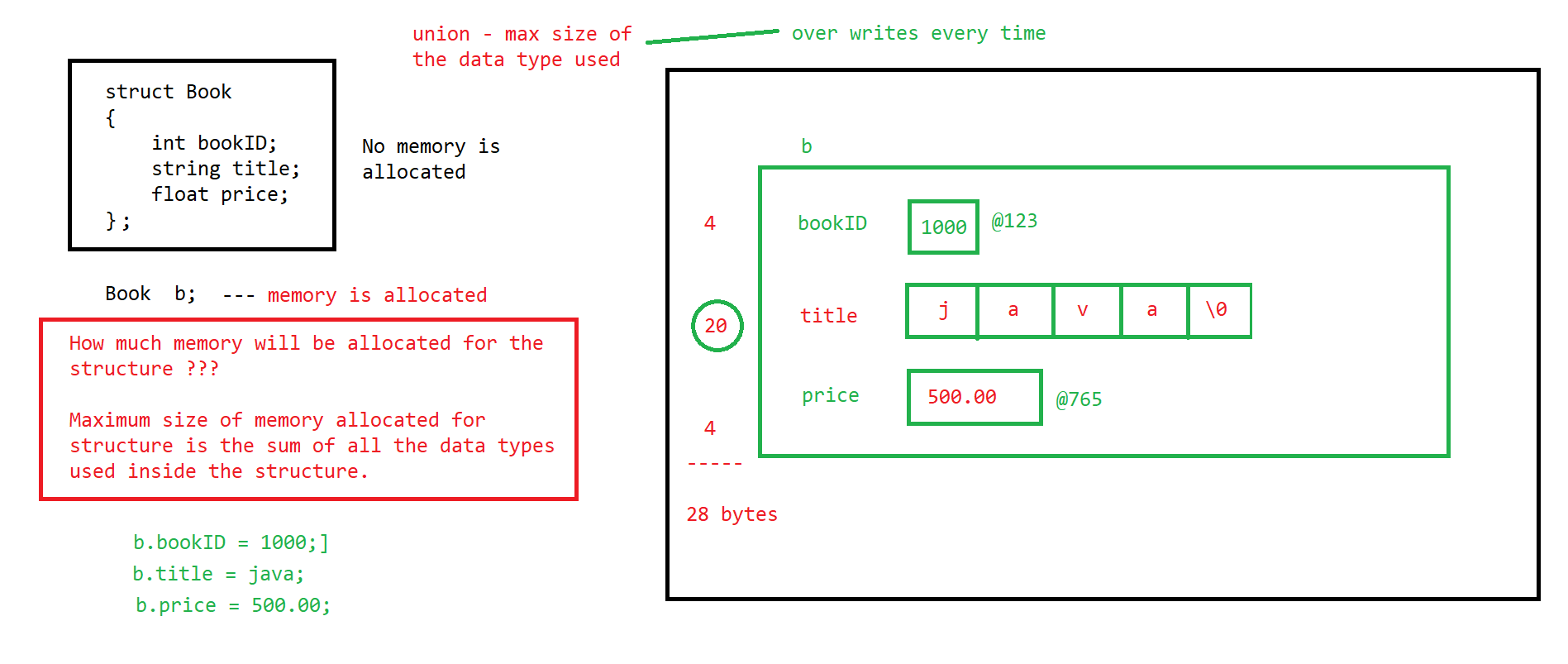




Generally no memory is allocated when we declare a structure .

Memory is allocated when we declare a variable of structure and initialize it .

The maximum size of memory allocated is equal to the total size of data types of all the data members of the structure.



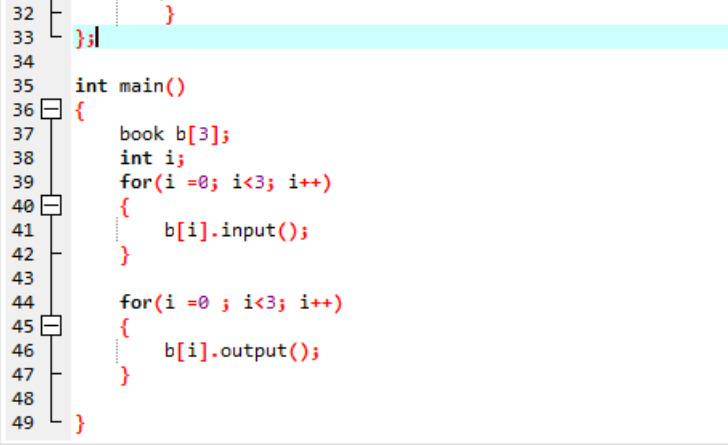
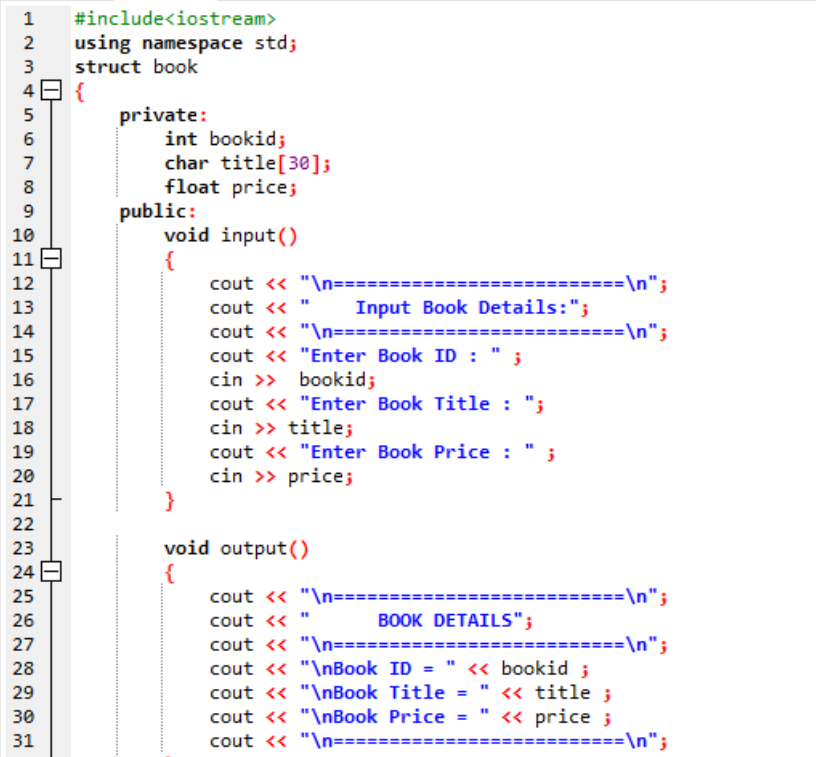
Structures in C++ can have data members and member functions .

Every member in a structure is by **default public**. It means these members can be accessed outside of structure. In C++ we can secure the members with private access specifiers.

We generally declare the data members as private and the member functions as public.

NOTE : Since structures are derived data types, **we also can create arrays of structure type**.

Example :



**NOTE: We also can have structures inside a structure.**

The following program will prompt the user to enter details of a store it in a structure, pass the structure as an argument to a

function, and display the resultant values on the screen.

//Write a program to pass a structure as an argument to a function

1

#include 2

using namespace std; 3

struct emp 4

{ 5

int empcode; 6

char name[20]; 7

ﬂoat cl; 8

ﬂoat pl; 9

ﬂoat sl; 10

}; 11

void pass(struct emp e1); 13

int main() 14

{ 15

struct emp e1; 16

cout <<”Enter Name of employee”<17

cin>>e1.name; 18

cout <<”Enter employee code”<19

cin>>e1.empcode; 20

cout <<”Enter casual leaves taken by employee”<21

cin>>e1.cl; 22

cout <<”Enter paid leaves taken by employee”<23

cin>>e1.pl; 24

cout <<”Enter sick leaves taken by employee”<25

cin>>e1.sl; 26

pass(e1); 27

return 0; 28

} 29

void pass(struct emp e1) 30

{ 31

ﬂoat cl=7; 32

ﬂoat pl=30; 33

ﬂoat sl=10; 34

cl=cl-e1.cl; 35

pl=pl-e1.pl; 36

sl=sl-e1.sl; 37

cout <<”Balance leaves of “ <“ ( “

<“<38

cout <<”Casual leaves” <<’\t’<<” - “ <39

cout <<”Paid leaves” <<’\t’<<” - “ <40

cout <<”Sick leaves” <<’\t’<<” - “ <41

} 42

**Explanation**

Lines 4 to 11

**struct emp**

**{**

**int empcode;**

**char name[20];**

**ﬂoat cl;**

**ﬂoat pl;**

**ﬂoat sl;**

**};** Lines 4 to 11 show the declaration of the structure Here, the structure is declared globally. Therefore, it can be used anywhere in the program.

Line 16

**struct emp e1;**

In this line, the variable **e1** is declared as an object of the structure type

Line 18

**cin>>e1.name;**

This line is used to assign the name entered by the user to the structure member **name** of the object

Line 20, 22, 24, and 26

These lines are similar to line 18.

Line 27 **pass(e1);**

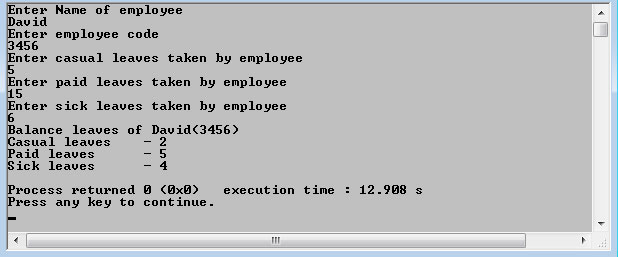
In this line, a function call is made to the function **pass( )** and the structure object **e1** is passed as an argument.

Line 30

**void pass(struct emp e1)**

After the execution of line 27, the control will reach this line (line 30). The information stored in the structure object **e1** (which is passed as an argument) is used to calculate balance leaves of the employee using the function

The output of the above example is displayed in the following figure.



**NOTE:** Revise the week-3 and week-4 learning materials to complete the tasks.

**Submission Details in the VLE:**

Submit your solution of worksheet-1[named as “yourid\_task\_1.docx”] along with a text file containing URL of your GIT REPO zipped in file named “YourID.zip” in the link provided on the VLE of UWE.

**Tasks**

**Task 1 : Basics of Derived types**

**Answer the following questions [1-5] with proper descriptions. [25 marks]**

1. Array is used to represent

(a) a list of data items of integer data type

(b) a list of data items of real data type

(c) a list of data items of different data type

(d) a list of data items of same data type  
An array is a collection of data items that are stored in a contiguous block of memory and all have the same data type. It is used to store a list of data items of the same data type, such as integers, floats, or characters. Arrays are useful for storing and organizing large amounts of data and for performing operations on data in a systematic way.

2. To initialize a 5 element array all having value 0 is given by

(a) int num[5] = {0};

(b) int num[5] = {0, 0, 0, 0, 0,};

(c) options a and b

(d) int num[5] = (1);

This will initialize an array of integers called "num" with 5 elements, all having a value of 0. Option (b) is also a valid way to initialize an array with 5 elements all having a value of 0, but option (c) is not correct because it lists both options (a) and (b) as correct answers. Option (d) is not correct because it attempts to initialize the array with a single element having a value of 1, rather than 5 elements all having a value of 0.

3. Consider the following declarations:

char \*a;

char a[5];

Which of the following statements is/are true?

1. both are of pointer type

2. memory allocation is different in both the cases

3. the value of a in declaration char a[5] can be changed later

(a) (2) only (b) (1) only (c) (1) and (2) (d) (2) and (3)

Statement 1: Both declarations are of pointer type.

Statement 2: Memory allocation is different in both cases. In the first declaration, "char \*a" is a pointer to a character, and memory is dynamically allocated for the character at runtime. In the second declaration, "char a[5]" is an array of characters with a fixed size of 5, and memory is statically allocated for the array at compile time.

Statement 3: The value of "a" in the declaration "char a[5]" can be changed later, as it is an array of characters and the elements of an array can be modified. However, the size of the array cannot be changed once it has been declared.

4. Given char \*p = “ANSI C”; identify the expression returning the value C.

(a) p[5] (b) \*( “ANSI C” + 5 ) (c) “ANSI C”[5] (d) all the above

All of the expressions listed will return the value 'C' when applied to the given char pointer "p" and the string "ANSI C".

In the expression "p[5]", the square brackets "[ ]" indicate that we want to access the element at the 5th index of the array pointed to by "p". Since the string "ANSI C" is stored in an array of characters, and the character 'C' is at the 5th index of this array (indexing starts at 0), this expression will return the value 'C'.

In the expression "( "ANSI C" + 5 )", the parentheses "( )" and the asterisk "" indicate that we want to dereference the pointer to the character at the 5th index of the array "ANSI C". This is equivalent to "ANSI C"[5] or p[5], and will also return the value 'C'.

Finally, in the expression "ANSI C"[5], the square brackets "[ ]" indicate that we want to access the element at the 5th index of the array "ANSI C". This is equivalent to "\*( "ANSI C" + 5 )" or p[5], and will also return the value 'C'.

5. Given char \*t[10]; identify the correct statement.

(a) strcpy(t[0], “BASIC”); (b) t[0] = “JAVA”;

(c) both options a and b (d) none of the above

The array t is an array of character pointers, and the statement (b) t[0] = “JAVA” is not correct because it attempts to assign a string literal to a character pointer. String literals are immutable, so they cannot be modified.

The function strcpy can be used to copy a string, including a string literal, into an array of characters or a character pointer. In this case, the string "BASIC" is being copied into the first element of the array t, which is a character pointer.

Therefore, the correct statement is (a) strcpy(t[0], “BASIC”).

6. Match the following. [10 marks]

|  |  |  |
| --- | --- | --- |
| 1 | data\_type (\*name[SIZE])() ; | Array of pointers |
| 2 | data\_type \*arrayname[SIZE]; | Pointer to a function |
| 3 | data\_type (\*name)(); | Array of pointer to a function |
| 4 | data type (\*arrayname)[SIZE]; | Pointer to pointer to pointer |
| 5 | char \*\*\*c; | Pointer to an array |

ANSWER:

|  |  |  |
| --- | --- | --- |
| 1 | data\_type (\*name[SIZE])() ; | Array of pointers |
| 2 | data\_type \*arrayname[SIZE]; | Pointer to an array |
| 3 | data\_type (\*name)(); | Pointer to a function |
| 4 | data type (\*arrayname)[SIZE]; | Pointer to pointer to pointer |
| 5 | char \*\*\*c; | Array of pointer to a function |

7. What is the output of the program? [10 marks]

main()

{

int i = 2;

twice (2);

cout << I << endl;

}

twice (int i)

{

i = i\*2;

}

Give reasons for the output.

The output of the program will be 2.

The function twice takes an integer argument and assigns the value of the argument multiplied by 2 to the local variable i within the function. However, this assignment does not affect the value of the global variable i in the main function. The global variable i is initialized to 2 at the beginning of the main function, and it retains this value throughout the program.

Therefore, the output of the program will be 2

8. struct car{

int speed;

char type[10];

} vehicle;

struct car \*ptr;

ptr = &vehicle;

Referring to the sample code above, which of the following will make the speed equal to 200? **[5 marks]**

(a) (\*ptr).speed = 200; (b) (\*ptr)–>speed = 200;

(c) \*ptr.speed = 200; (d) &ptr.speed = 200;

Answer must contain correct option along with description.

The correct option is (a) (\*ptr).speed = 200.

The struct car type has two members: an integer named speed and an array of characters named type. The variable vehicle is an instance of the struct car type. The variable ptr is a pointer to a struct car type and is initialized to point to the address of the variable vehicle.

To access the members of a struct through a pointer, you can use the arrow operator (->) or the dereference operator (\*) combined with the dot operator (.).

In this case, the correct syntax for accessing the member speed through the pointer ptr is either (\*ptr).speed or ptr->speed.

Option (b) (\*ptr)–>speed = 200 is incorrect because it has a typographical error (a hyphen instead of the greater than symbol).

Option (c) ptr.speed = 200 is incorrect because the dot operator (.) has higher precedence than the dereference operator (). This means that the expression is interpreted as \*(ptr.speed), which attempts to dereference the member speed of ptr, rather than as (\*ptr).speed, which accesses the member speed through the pointer ptr.

Option (d) &ptr.speed = 200 is incorrect because the dot operator (.) has higher precedence than the address of operator (&). This means that the expression is interpreted as &(ptr.speed), which takes the address of the member speed of ptr, rather than as &(ptr).speed, which takes the address of the entire struct pointed to by ptr. In any case, it is not possible to assign a value to an address.

Therefore, the correct option is (a) (\*ptr).speed = 200. This expression correctly accesses the member speed through the pointer ptr and assigns the value 200 to it.

**Task 2 : Debugging Exercises : 20 Marks**

Identify the compiler errors in Problems 1 and 2 and state what is wrong with the code**. [10 marks]**

1.

#include <iostream>

using namespace std;

char [] FillArray();

int main()

{

int values[75];

values = FillArray();

return 0;

}

**The compiler error in this code is in the function declaration for FillArray. The correct syntax for declaring a function that returns an array of characters is:**

**char\* FillArray();**

**The declaration char [] FillArray(); is not correct because it is missing the type of the array elements.**

2.

#include <iostream>

using namespace std;

void SortArray(int values[]);

int main()

{

int values[75];

//assume values becomes filled with data

SortArray(values[75]);

return 0;

}

void SortArray(int values)

{

for(i = 0; i < 75; ++i)

{

if(values[i] < values[i-1]);

values[i] = values[i-1];

}

}

The compiler error in this code is in the function call to SortArray. The correct syntax for passing an array to a function is:

SortArray(values);

The function call SortArray(values[75]); passes the 75th element of the array values to the function, which is not what is intended. This is not a valid syntax because the index of an array must be a nonnegative integer.

Identify the runtime errors in Problems 3 & 4 and describe why the problems occured.

**[10 marks]**

3. Specification: Ask the user for a character array and then reverse the characters.

If the user entered Hello World, the new string would read dlroW olleH.

#include <iostream>

using namespace std;

int main()

{

char saying[50],revSaying[50];

int i;

cout << "\nEnter a saying.";

cin.getline(saying,50);

for(i = 0; i< 50; ++i)

{

revSaying[i] = saying[50-i];

}

return 0;

}

There are no runtime errors in this code. However, the code does not achieve the desired result of reversing the characters in the saying.

To reverse the characters in the string, the loop should iterate from the end of the string towards the beginning. One way to do this is to use the strlen function to determine the length of the string and use that value as the upper bound of the loop. Additionally, the revSaying array should be null-terminated to be a valid string. Here is a version of the code that correctly reverses the characters in the string:

#include <iostream>

#include <cstring> // for strlen

using namespace std;

int main()

{

char saying[50], revSaying[50];

int i;

cout << "\nEnter a saying: ";

cin.getline(saying, 50);

int len = strlen(saying);

for (i = 0; i < len; ++i)

{

revSaying[i] = saying[len - i - 1];

}

revSaying[len] = '\0'; // null-terminate the string

cout << "The reversed saying is: " << revSaying << endl;

return 0;

}

4. Specification: Fill a character array with the uppercase alphabet (place A in

[0], B in [1], etc.).

#include <iostream>

using namespace std;

int main()

{

char alphabet[26];

int i;

for(i = 0; i< 26; ++i)

{

alphabet[i] = i;

}

return 0;

}

There is a runtime error in this code in the for loop of the main function. The loop variable i is being used to index the alphabet array and to store the ASCII code of the uppercase alphabet characters. However, the ASCII code for the uppercase alphabet characters is not equal to their index in the alphabet.

To fill the alphabet array with the uppercase alphabet, you should assign the ASCII code for each character to the corresponding element of the array. The ASCII code for 'A' is 65, for 'B' is 66, and so on.

Here is a version of the code that correctly fills the alphabet array with the uppercase alphabet:

#include <iostream>

using namespace std;

int main()

{

char alphabet[26];

int i;

for (i = 0; i < 26; ++i)

{

alphabet[i] = 'A' + i;

}

cout << "The alphabet is: ";

for (i = 0; i < 26; ++i)

{

cout << alphabet[i] << " ";

}

cout << endl;

return 0;

}

**Task 3 : Programming Exercises: [15X2 = 30 marks]**

1. Write a program that uses the random number generator functions to generate a random sequence of letters from A to Z. The GenerateALetter function should return a letter based on the ASCII table; that is, A = 65 and Z = 90. In main, above the “do again” loop, ask the user to enter a seed value and call srand() passing it the seed value. Write out fifteen randomly-generated characters on one line (separated by commas).

*Note*: You will need to place the call to GenerateALetter inside a for loop and write one letter at a time.

Go through the following example to get an idea how to analyze a problem based on functions.

IsItPrime is Program. It asks the user to enter a positive integer value.

The program checks that the value is positive, and then determines if the number is prime.

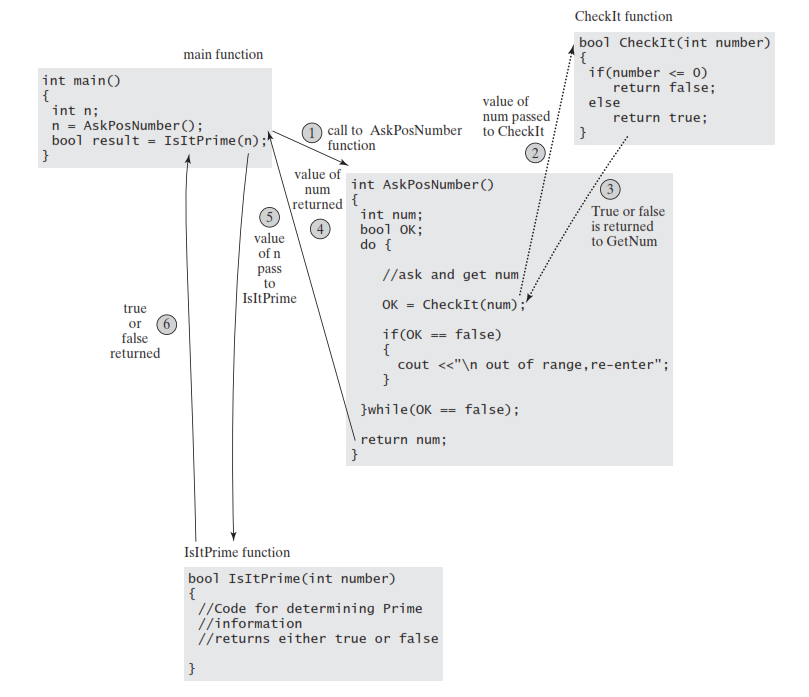
The program flow is illustrated in the figure below with the numbered arrows indicating the function call order and values that are passed between functions. This program calls the function AskPosNumber to get a value from the user. In the AskPosNumber function, it calls the function CheckIt. This function checks to see if the value is positive (using a boolean return value, true or false).

If it is zero or negative, AskPosNumber asks the user to re-enter the value. Once

a positive integer is returned to main, we call the IsItPrime function, which determines if the number is prime. The IsItPrime function returns a true or false to main.

The circled numbers indicate the order of the function calls and returned values.

From the diagram you will have some idea how functions are declared and called. Now write a C++ programs for the following question.

****

IsItPrime Program Flow.

#include <iostream>

#include <cstdlib> // for srand, rand

#include <ctime> // for time

using namespace std;

// Generates a random letter from A to Z

char GenerateALetter()

{

int letter = rand() % 26 + 'A'; // generate a random number between 65 and 90

return static\_cast<char>(letter);

}

int main()

{

int seed;

cout << "Enter a seed value: ";

cin >> seed;

srand(seed);

cout << "The randomly-generated letters are: ";

for (int i = 0; i < 15; ++i)

{

cout << GenerateALetter() << ", ";

}

cout << endl;

return 0;

}

1. Create a structure named Dog with a string field for the Dog’s name. Create a structure named Cat with a string field for the Cat’s name. Write a program that declares one Dog and one Cat, and assign names to them. Write two overloaded functions named speak().

If you pass the Dog to speak(), the speak()function should display the Dog’s name and

a description of how dogs speak (for example, “Spot says woof” ). If you pass the Cat to the version of speak()that accepts a Cat parameter, then it should display the Cat’s name and a description of how cats speak (for example, “Tiger says meow”). Save the file as **CatDog.cpp.**

#include <iostream>

#include <string>

using namespace std;

struct Dog

{

string name;

};

struct Cat

{

string name;

};

// Overloaded function to make a dog speak

void speak(Dog d)

{

cout << d.name << " says woof" << endl;

}

// Overloaded function to make a cat speak

void speak(Cat c)

{

cout << c.name << " says meow" << endl;

}

int main()

{

Dog d;

d.name = "Spot";

Cat c;

c.name = "Tiger";

speak(d); // make the dog speak

speak(c); // make the cat speak

return 0;

}

**Task 4**

* Check and commit all your solutions.
* This task carries no marks but it is mandatory. Ensure that your solution is visible to us.