**Blood Bank Management System in C**

# Introduction

The Blood Bank Management System is crucial for maintaining blood inventory and ensuring blood availability for patients. A programe of c language is built for managing this system using different type of user define function. This report details the functionality of each function implemented in this code for a blood bank management system.

# Functions overview

This system consist of some user define function to add, delete, update, and display blood group with quantity and to create blood group package. A structure type of array named bloodGroup used to store and represent blood groups with relevant information.

After running the program there will be appear several instructions into choice to manage this system. Entering the choice any of them user will be able to manage effecientely.

**Structures**

**Struct BloodGroup**

* This structure defines a blueprint for storing information about a blood group.
* It contains two members:
  1. **bloodType:** A character array to store the blood type (e.g., "A+", "O-").
  2. **quantity:** An integer to store the available quantity of that blood type.

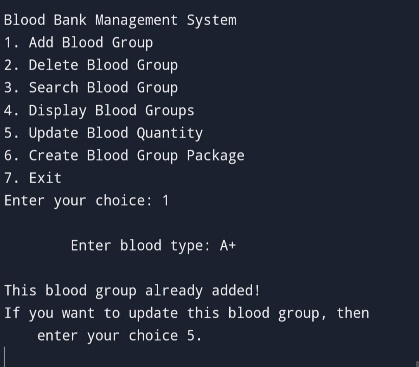
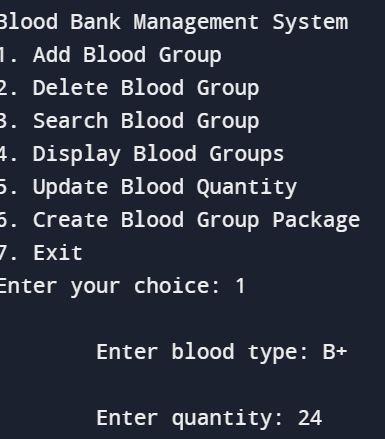
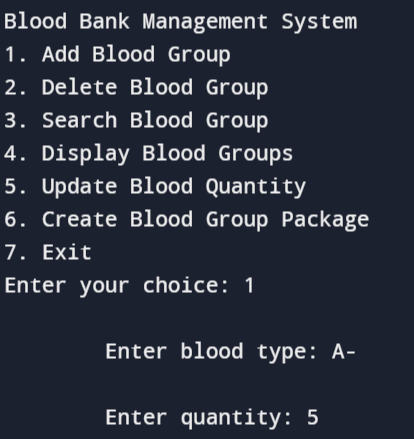
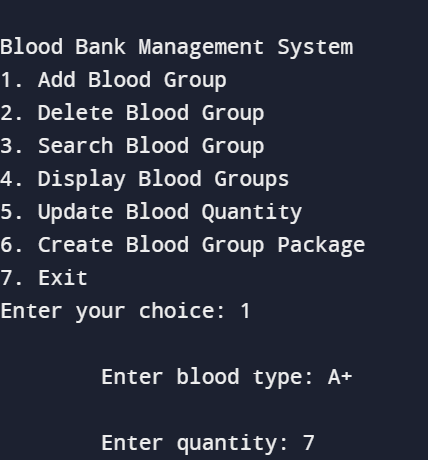
**struct BloodGroupPackage**

* A structure to store packages in the system.
* It contains two members:
  1. **N:** An integer to store the number of blood group in a package.
  2. **BloodGroup bloodPackage[10]:** A structure array to store blood groups and their quantity.

# Add Blood Group

**addBloodGroup(struct BloodGroup \* bloodGroups, int \* count , int maxCount)**

Output

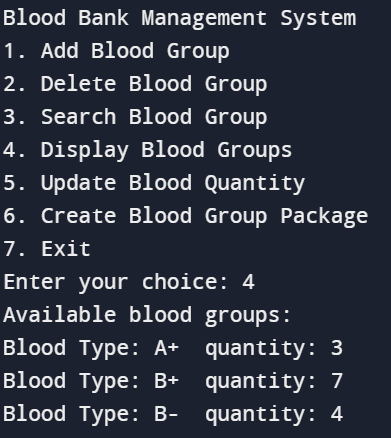
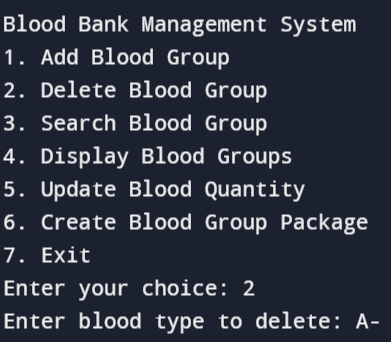
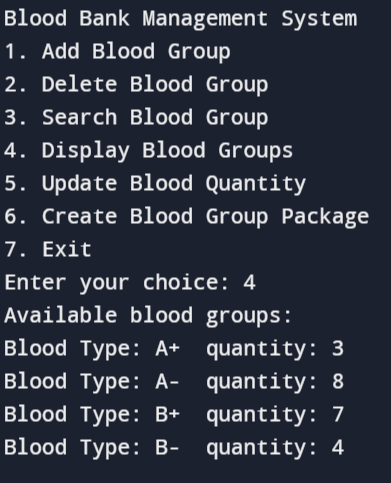


* This function adds a new blood group to the system.
* It takes three arguments:
  1. bloodGroups: A pointer to an array of BloodGroup structs.
  2. count: A pointer to an integer variable that keeps track of the number of blood groups currently stored.
  3. maxCount: An integer representing the maximum number of blood groups the system can hold.
* It first checks if the blood group array is full using \*count and maxCount.
* If full, it displays a message and returns.
* Otherwise, it prompts the user to enter the blood type.
* Then it calls searchBloodGroup to check if the blood type already exists or not.
* If the blood type already exists, it displays a message and returns.
* If everything is valid, it accept the quantity and copies the blood type string to the next available element in the bloodGroups array using strcpy. Then it sets the quantity for the new blood group.
* Finally, it increments the count to reflect the addition of a new blood group.

# Delete Blood Group

**deleteBloodGroup(struct BloodGroup \*bloodGroups, int count, const char bloodType)**

Output



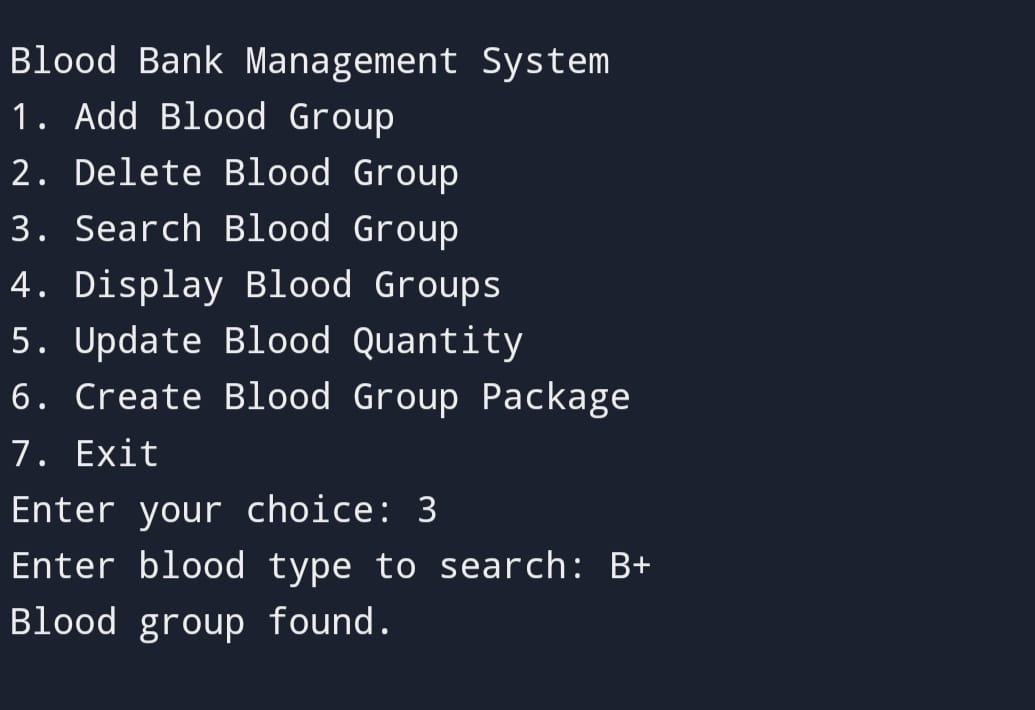
This function deletes a blood group from the system.

* It takes three arguments:
  1. **bloodGroups:** A pointer to an array of BloodGroup structs.
  2. **count:** A pointer to an integer variable that keeps track of the number of blood groups currently stored.
  3. **bloodType:** A constant character pointer to the blood type to delete.
* It first calls searchBloodGroup to find the index of the blood group to delete.
* If the blood group is not found (when searchBloodGroup returns -1), it displays a message and return to main function.
* Otherwise, it iterates through the bloodGroups array starting from the index of the blood group to be deleted, up to count-1 (excluding the last element).
* For each element in this loop, it shifts the elements one position to the left, effectively overwriting the element to be deleted with the next element in the array.
* Then it decrements the count to reflect the deletion of a blood group.
* After deleting the blood group, it calls the function updateBloodPackageAfterDelete to update blood group packages affected by the deletion

# Search Blood Group

**searchBloodGroup(struct BloodGroup \* bloodGroups, int count, const char \* bloodType)**

Output



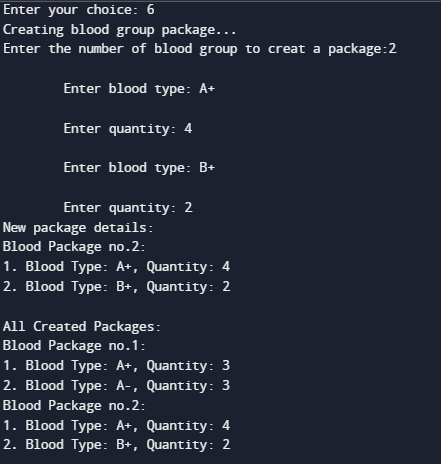
This function searches for a specific blood group in the system.

* It takes three arguments:
  1. bloodGroups: A pointer to an array of BloodGroup structs.
  2. count: An integer representing the number of blood groups currently stored.
  3. bloodType: A constant character pointer to the blood type to search for.
* It iterates through the bloodGroups array from index 0 to count-1.
* For each blood group, it compares the bloodType with the current blood group's bloodType using strcmp.
* If a match is found, it returns the index of the matching blood group in the array.
* If no match is found after iterating through all elements, it returns -1.

# Blood Group Package

**createBloodGroupPackage (struct BloodGroup \* bloodGroups, int count, const char \* packageDetails)**

Output



This function creates a new blood group package.

* . It takes three arguments:

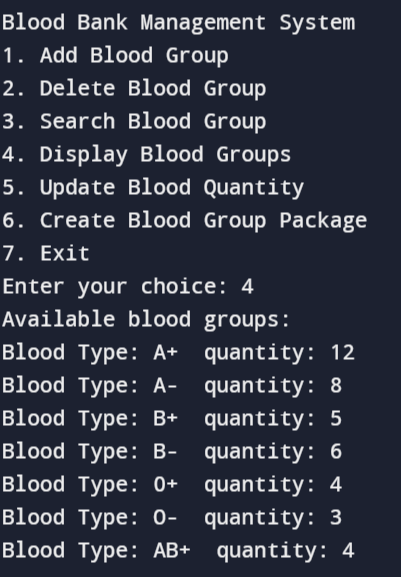
1. **bloodGroups:** The array of struct BloodGroup.
2. **count:** The number of blood groups currently stored.

* The function first prompt user to entered the number of blood groups for creating a new blood package and store at Packages[package\_count].N for further access.
* For each blood group in the package :
* It first take the blood type then it calls searchBloodGroup to check if the blood type exists or not. If the type is not found it leave an error massage and return to main function.
* Otherwise it take take the blood quantity checks if the blood bank has enough quantity of the entered blood type.
* If sufficient quantity exists, adds the blood type and its quantity to the index of the array bloodPackages.
* Then the index increament by the 1 to store next blood group detail and this process continue until the index is less then Packages[package\_count].N.
* After creating the new package this program display it then display all created package.

# Display Blood Groups

**displayBloodGroups(struct BloodGroup \* bloodGroups, int count)**

Output



This function displays information about all blood groups currently stored in the system.

* It takes two arguments:

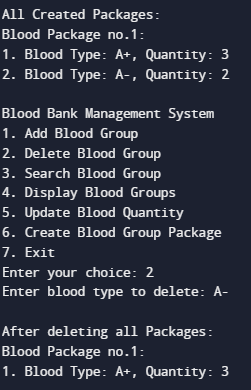
1. bloodGroups: A pointer to an array of BloodGroup structs.
2. Count: An integer representing the number of blood groups currently stored.

* If count is 0 (no blood groups), it displays a message indicating no blood groups are found.
* Otherwise, it iterates through the bloodgroups array and print the blood group with quantity of each element.

# Update Blood Package After Delete

**updateBloodPackageAfterDelete(const char \*bloodType)**

Output

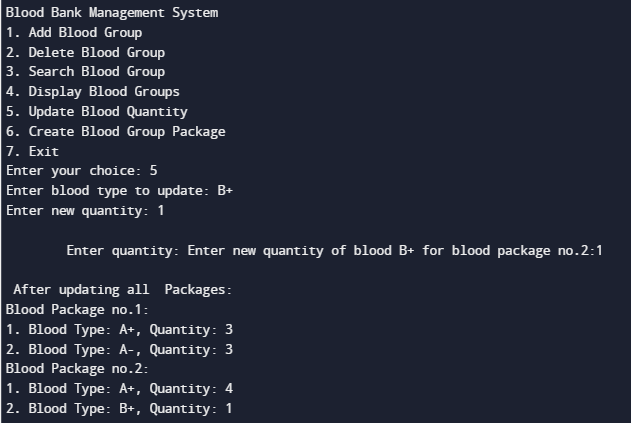


This function updates blood group packages after a blood group is deleted.

* It takes one argument, bloodType which is a pointer of deleted blood group.
* It iterates through all created blood packages (Packages array).
* For each package, it iterates through the blood groups within the package.
* If the blood type in the package matches the deleted blood type, it shifts the remaining blood groups in the package to fill the gap and decrements the package size (N).

# Update Blood Package After Update

**updateBloodPackageAfterUpdate(struct BloodGroup \*bloodGroups, int count, const char \*bloodType)**



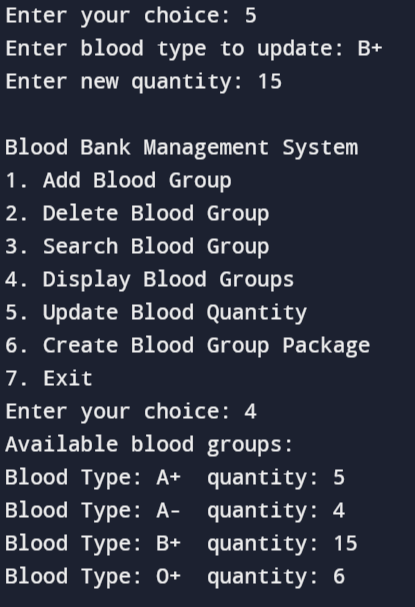
This function updates the quantity of a specific blood group within blood packages after the blood group's quantity is updated.

* It takes three arguments:
  1. **bloodGroups:** A pointer to an array of BloodGroup structs.
  2. **count:** An integer representing the number of blood groups currently stored.
  3. **bloodType:** A constant character pointer to the blood type to search for updating quantity.
* It iterates through all created blood packages (Packages array).
* For each package, it iterates through the blood groups within the package.
* If the blood type in the package matches the updated blood type, it prompts the user to enter the new quantity and updates it in the package. Then it print all packages.

# Update Blood Quantity

**updateBloodQuantity(struct BloodGroup \* bloodGroups, int count, const char \* bloodType, int quantity)**

Output



This function updates the quantity of a specific blood group. It takes four arguments:

* It takes four arguments:

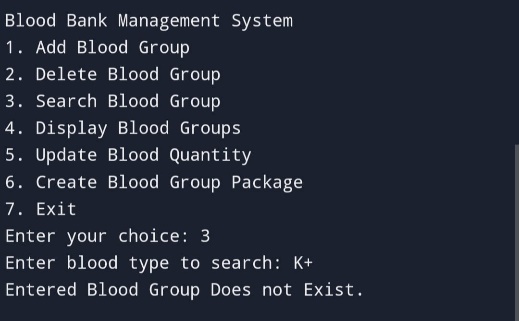
1. bloodGroups: The array of struct BloodGroup.
2. count: The number of blood groups currently stored.
3. bloodType: The blood type to update (as a string).
4. quantity: The new quantity for the blood type.

* The function first calls searchBloodGroup to find the index of the blood type to update. If the blood type is not found, it displays an error message and returns.
* Otherwise, it updates the quantity of the blood group at the found index with the provided quantity.
* After updating the quantity it calls updateBloodPackageAfterUpdate to update blood group packages affected by the quantity change.

# Blood Type Validity

**isValidBloodType(const char \*bloodType)**

Output



* This function checks if the provided blood type is valid.
* It maintains a hardcoded array of valid blood types ("A+", "A-", "B+", etc.).
* It iterates through the array and compares the provided blood type with each valid type using strcmp.
* If a match is found, it returns true, indicating a valid blood type.
* Otherwise, it returns false.

# Conclusion

This C code provides a foundational framework for a blood bank management system. It offers functionalities for adding, deleting, searching, displaying, and updating blood group information. Additionally, it includes a function for creating blood group packages to access easily.

For further development, the system could benefit from:

* Implementing functionalities within createBloodGroupPackage to handle actual package creation and management.
* Error handling for invalid user inputs beyond blood type validation.
* Integration with a database for persistent storage of blood bank data.
* User authentication and access control for secure operation.