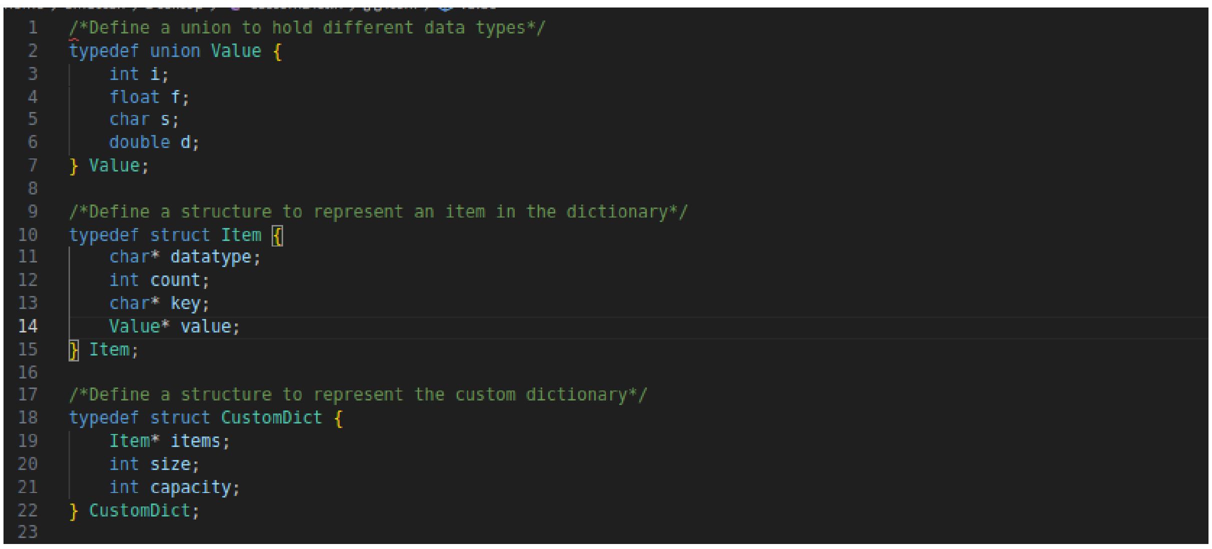
CSE 102 Spring 2024 –Computer Programming Assignment 12

Umutcan Ocak // 220104004903

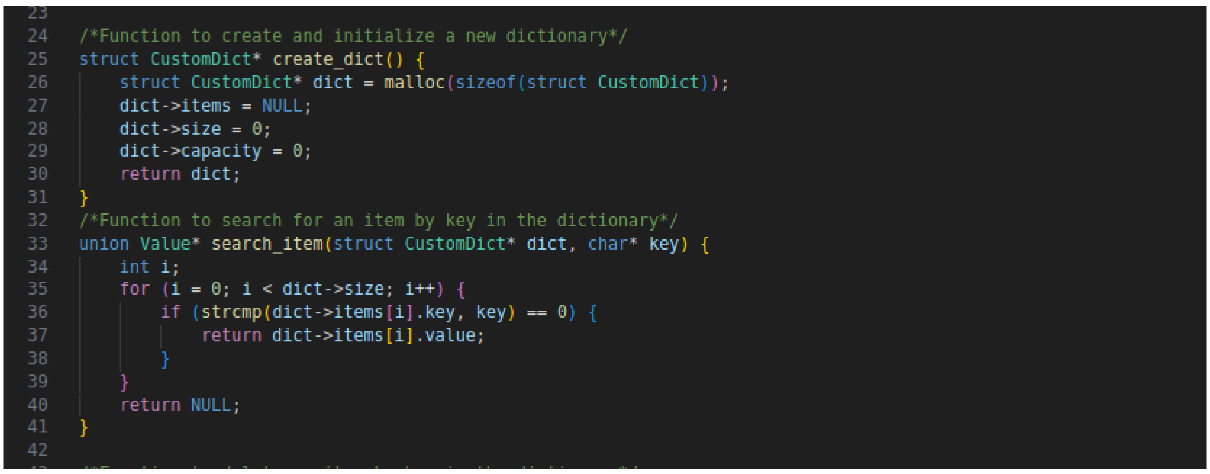
Youtube : https://youtu.be/fWr0N9xLsvw

This code fragment creates a dictionary structure and contains various functions to perform operations such as adding, deleting, searching, sorting and printing data in this dictionary. It uses union and item structures to represent data types. When reading data from a CSV file, it processes the data in each line and adds it to the dictionary as appropriate. In summary, this code implements a dictionary management system using data structures and file processing techniques.



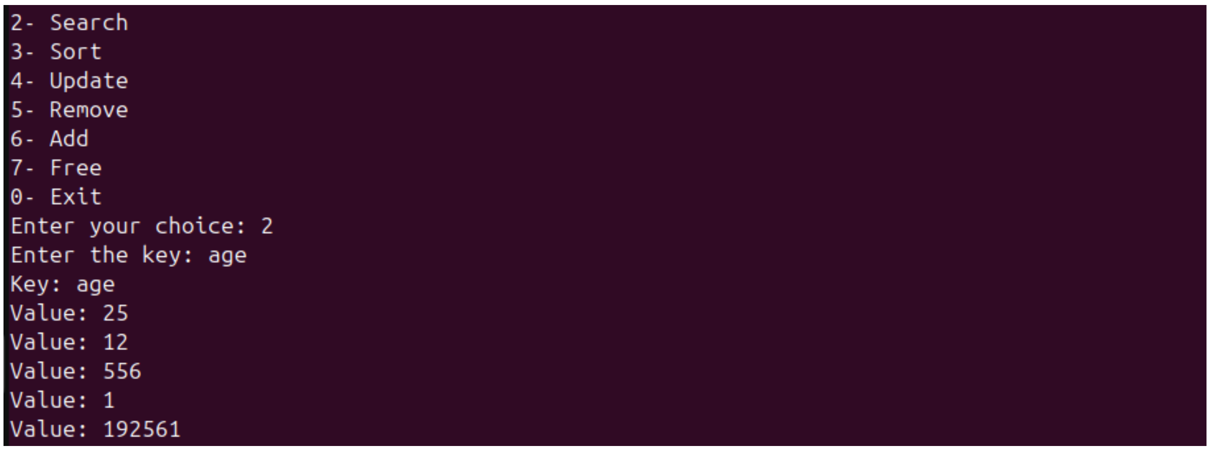
```c
/*Define a union to hold different data types*/
typedef union Value {
    int i;
    float f;
    char s;
    double d;
} Value;

/*Define a structure to represent an item in the dictionary*/
typedef struct Item {
    char* datatype;
    int count;
    char* key;
    Value* value;
} Item;

/*Define a structure to represent the custom dictionary*/
typedef struct CustomDict {
    Item* items;
    int size;
    int capacity;
} CustomDict;

```

This code defines a union for multiple data types and structures for a dictionary item and a custom dictionary, including data type, key, value, size, and capacity information.

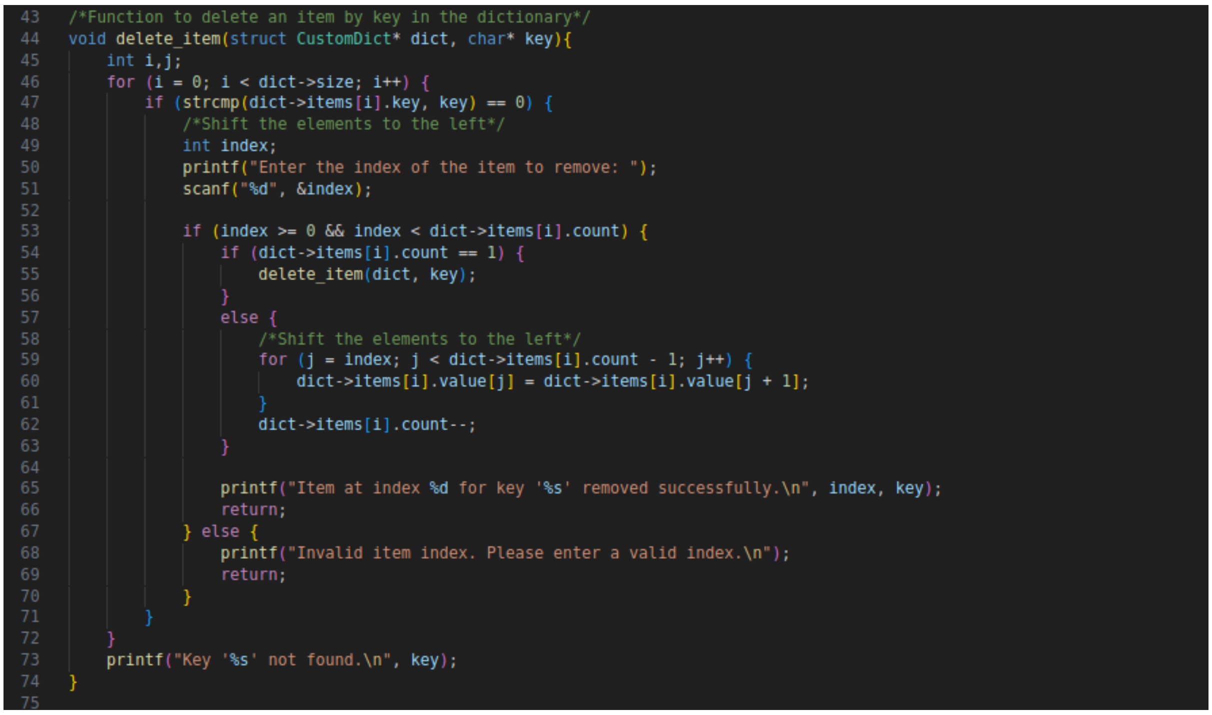


```c

/*Function to create and initialize a new dictionary*/
struct CustomDict* create_dict() {
    struct CustomDict* dict = malloc(sizeof(struct CustomDict));
    dict->items = NULL;
    dict->size = 0;
    dict->capacity = 0;
    return dict;
}
/*Function to search for an item by key in the dictionary*/
union Value* search_item(struct CustomDict* dict, char* key) {
    int i;
    for (i = 0; i < dict->size; i++) {
        if (strcmp(dict->items[i].key, key) == 0) {
            return dict->items[i].value;
        }
    }
    return NULL;
}

```

This code includes functions to create and initialize a new custom dictionary and to search for an item by key within the dictionary.

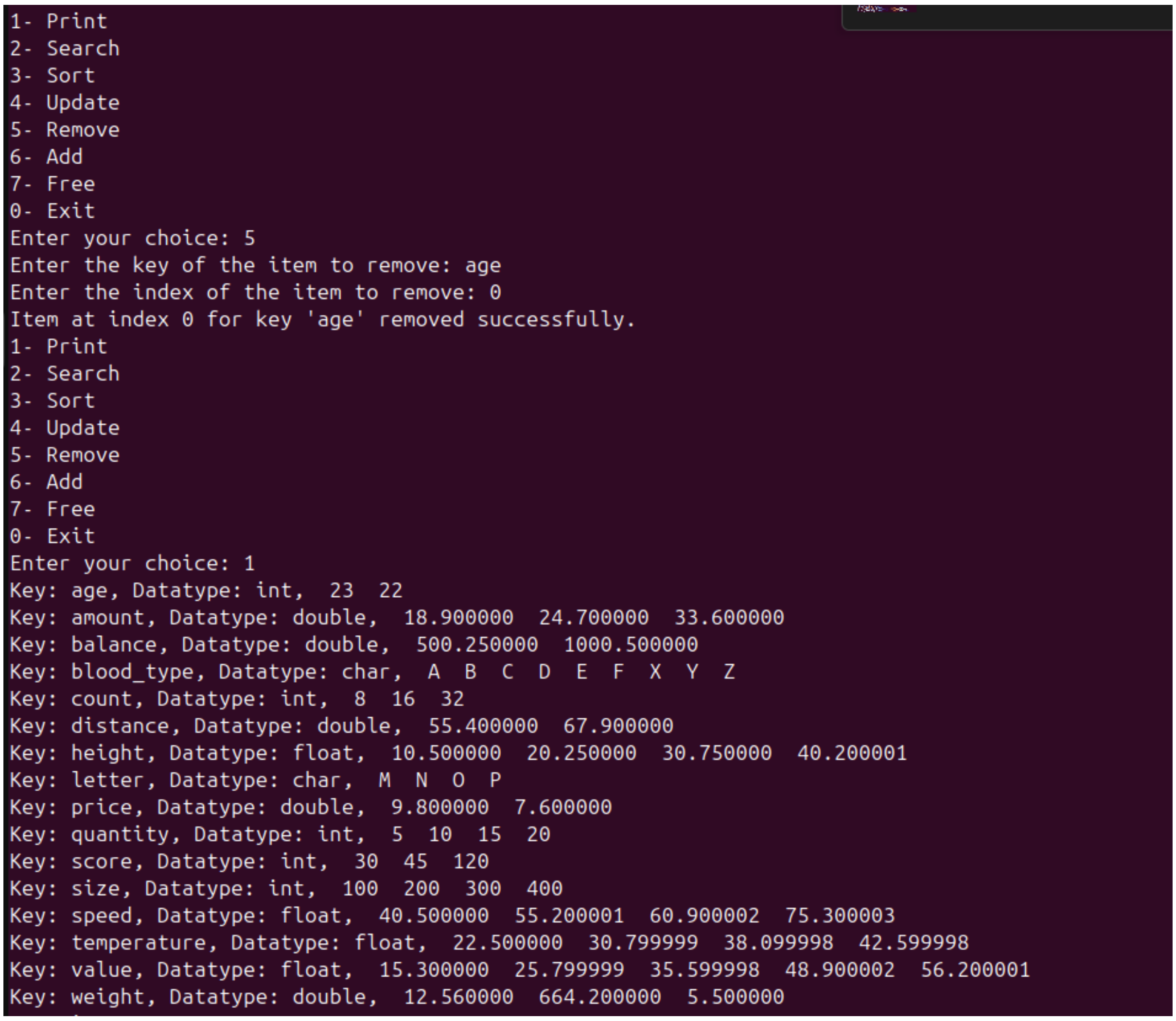


```
2- Search
3- Sort
4- Update
5- Remove
6- Add
7- Free
0- Exit
Enter your choice: 2
Enter the key: age
Key: age
Value: 25
Value: 12
Value: 556
Value: 1
Value: 192561
```

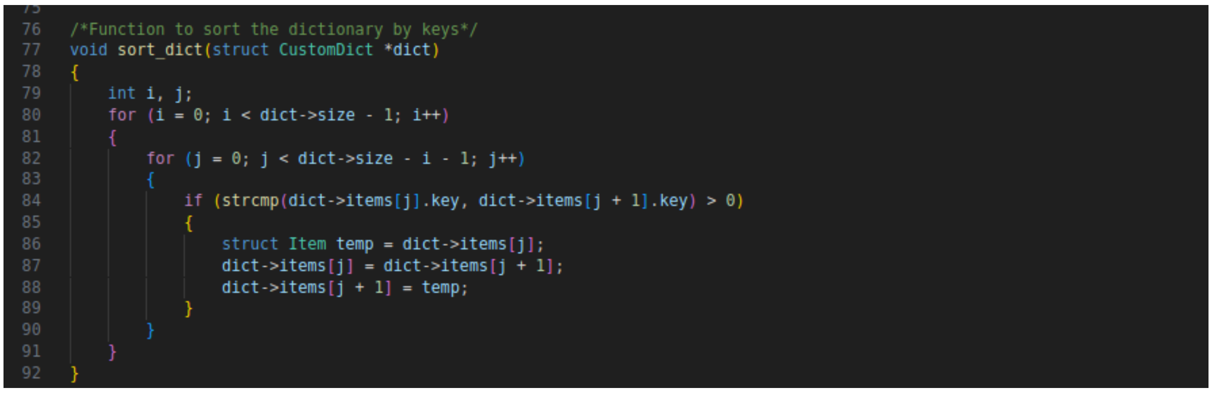


```c
/*Function to delete an item by key in the dictionary*/
void delete_item(struct CustomDict* dict, char* key){
    int i,j;
    for (i = 0; i < dict->size; i++) {
        if (strcmp(dict->items[i].key, key) == 0) {
            /*Shift the elements to the left*/
            int index;
            printf("Enter the index of the item to remove: ");
            scanf("%d", &index);

            if (index >= 0 && index < dict->items[i].count) {
                if (dict->items[i].count == 1) {
                    delete_item(dict, key);
                }
                else {
                    /*Shift the elements to the left*/
                    for (j = index; j < dict->items[i].count - 1; j++) {
                        dict->items[i].value[j] = dict->items[i].value[j + 1];
                    }
                    dict->items[i].count--;
                }

                printf("Item at index %d for key '%s' removed successfully.\n", index, key);
                return;
            } else {
                printf("Invalid item index. Please enter a valid index.\n");
                return;
            }
        }
    }
    printf("Key '%s' not found.\n", key);
}
```

This code defines a function to delete an item by key from a custom dictionary, including shifting elements and adjusting the count if needed.

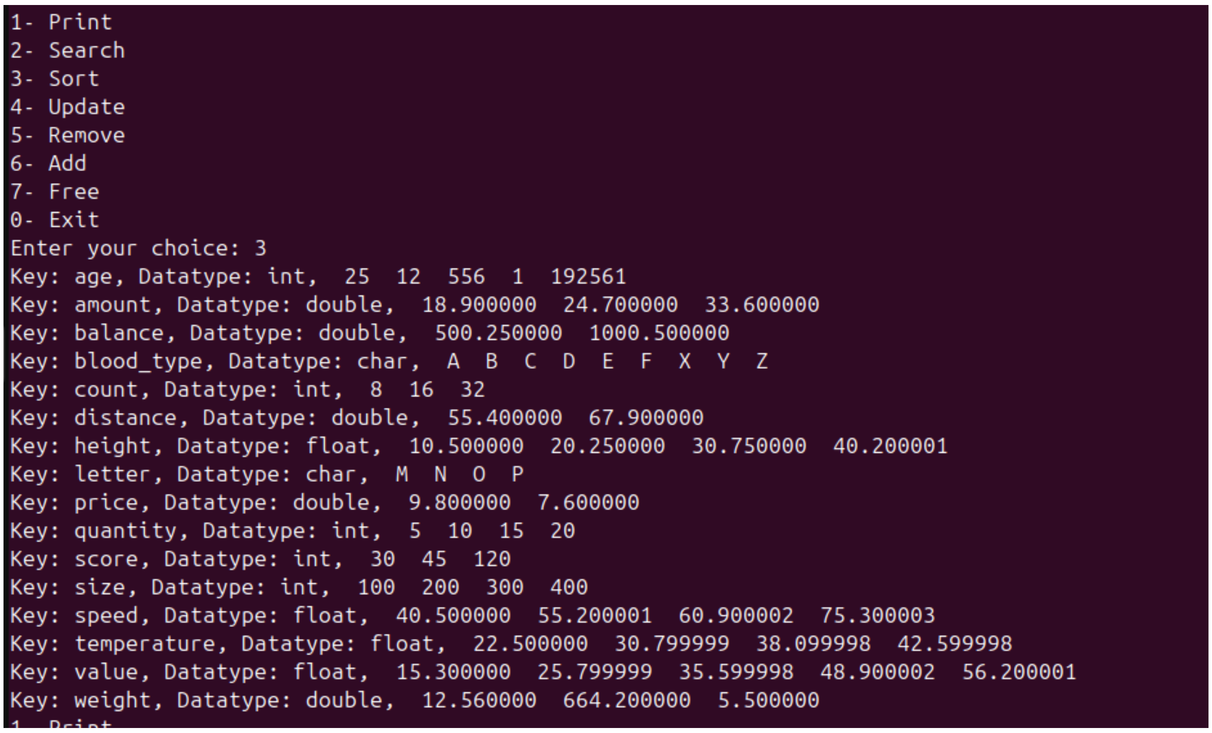


```
1- Print
2- Search
3- Sort
4- Update
5- Remove
6- Add
7- Free
0- Exit
Enter your choice: 5
Enter the key of the item to remove: age
Enter the index of the item to remove: 0
Item at index 0 for key 'age' removed successfully.
1- Print
2- Search
3- Sort
4- Update
5- Remove
6- Add
7- Free
0- Exit
Enter your choice: 1
Key: age, Datatype: int,  23  22
Key: amount, Datatype: double,  18.900000  24.700000  33.600000
Key: balance, Datatype: double,  500.250000  1000.500000
Key: blood_type, Datatype: char,  A  B  C  D  E  F  X  Y  Z
Key: count, Datatype: int,  8  16  32
Key: distance, Datatype: double,  55.400000  67.900000
Key: height, Datatype: float,  10.500000  20.250000  30.750000  40.200001
Key: letter, Datatype: char,  M  N  O  P
Key: price, Datatype: double,  9.800000  7.600000
Key: quantity, Datatype: int,  5  10  15  20
Key: score, Datatype: int,  30  45  120
Key: size, Datatype: int,  100  200  300  400
Key: speed, Datatype: float,  40.500000  55.200001  60.900002  75.300003
Key: temperature, Datatype: float,  22.500000  30.799999  38.099998  42.599998
Key: value, Datatype: float,  15.300000  25.799999  35.599998  48.900002  56.200001
Key: weight, Datatype: double,  12.560000  664.200000  5.500000
```

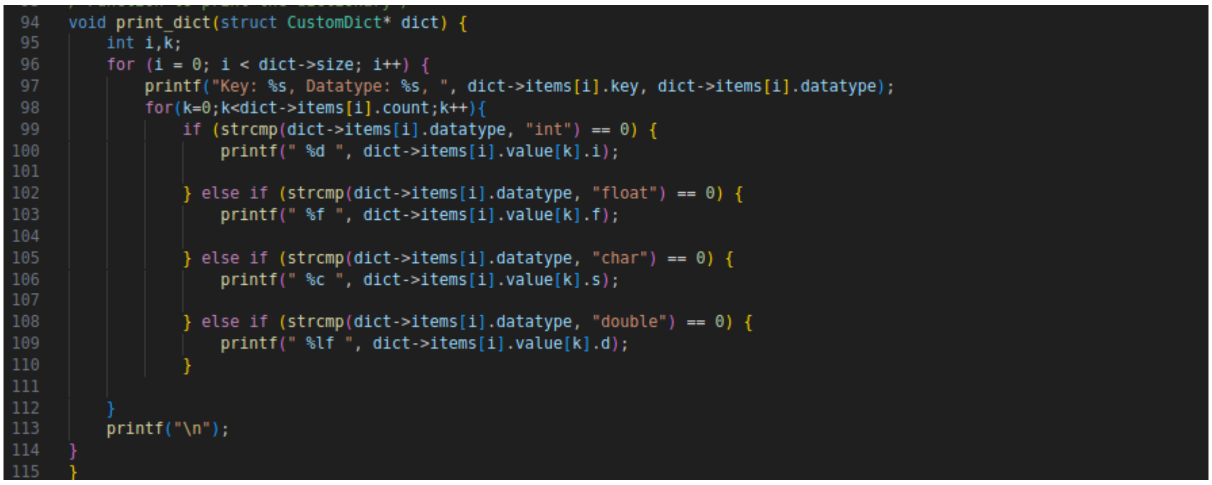


```c
/*Function to sort the dictionary by keys*/
void sort_dict(struct CustomDict *dict)
{
    int i, j;
    for (i = 0; i < dict->size - 1; i++)
    {
        for (j = 0; j < dict->size - i - 1; j++)
        {
            if (strcmp(dict->items[j].key, dict->items[j + 1].key) > 0)
            {
                struct Item temp = dict->items[j];
                dict->items[j] = dict->items[j + 1];
                dict->items[j + 1] = temp;
            }
        }
    }
}
```

This code defines a function to sort the custom dictionary by keys using a simple bubble sort algorithm.

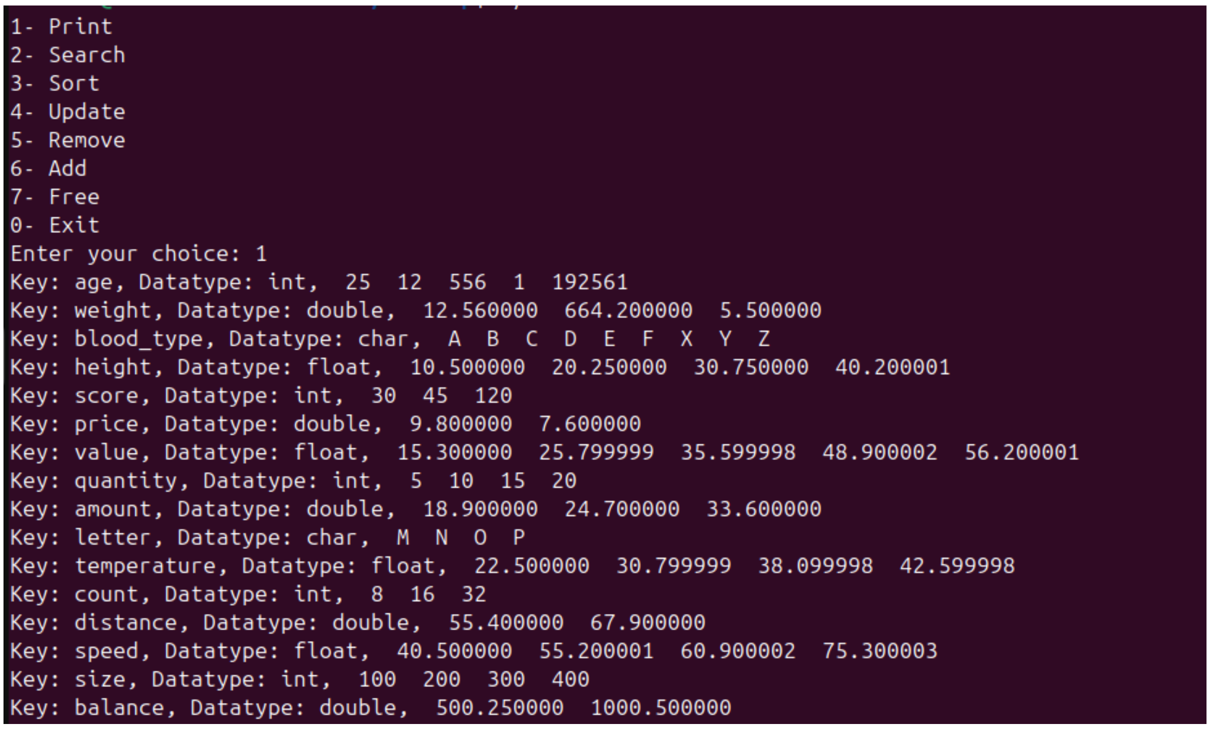


```
1- Print
2- Search
3- Sort
4- Update
5- Remove
6- Add
7- Free
0- Exit
Enter your choice: 3
Key: age, Datatype: int,  25  12  556  1  192561
Key: amount, Datatype: double,  18.900000  24.700000  33.600000
Key: balance, Datatype: double,  500.250000  1000.500000
Key: blood_type, Datatype: char,  A  B  C  D  E  F  X  Y  Z
Key: count, Datatype: int,  8  16  32
Key: distance, Datatype: double,  55.400000  67.900000
Key: height, Datatype: float,  10.500000  20.250000  30.750000  40.200001
Key: letter, Datatype: char,  M  N  O  P
Key: price, Datatype: double,  9.800000  7.600000
Key: quantity, Datatype: int,  5  10  15  20
Key: score, Datatype: int,  30  45  120
Key: size, Datatype: int,  100  200  300  400
Key: speed, Datatype: float,  40.500000  55.200001  60.900002  75.300003
Key: temperature, Datatype: float,  22.500000  30.799999  38.099998  42.599998
Key: value, Datatype: float,  15.300000  25.799999  35.599998  48.900002  56.200001
Key: weight, Datatype: double,  12.560000  664.200000  5.500000
```

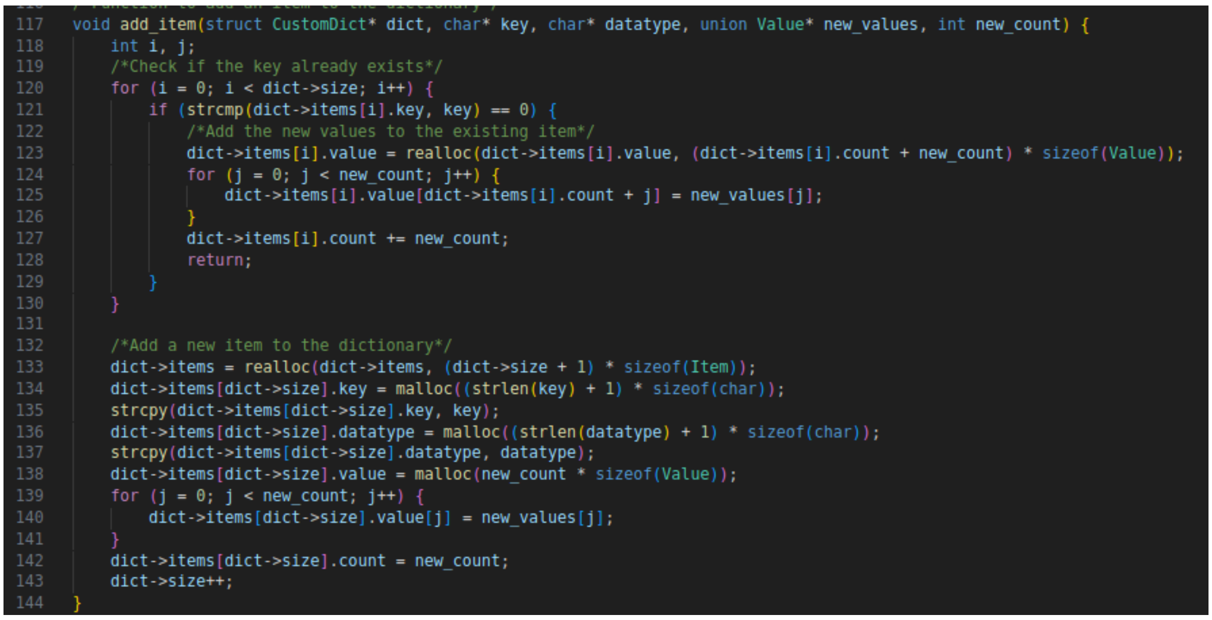


```c
void print_dict(struct CustomDict* dict) {
    int i,k;
    for (i = 0; i < dict->size; i++) {
        printf("Key: %s, Datatype: %s, ", dict->items[i].key, dict->items[i].datatype);
        for(k=0;k<dict->items[i].count;k++){
            if (strcmp(dict->items[i].datatype, "int") == 0) {
                printf(" %d ", dict->items[i].value[k].i);

            } else if (strcmp(dict->items[i].datatype, "float") == 0) {
                printf(" %f ", dict->items[i].value[k].f);

            } else if (strcmp(dict->items[i].datatype, "char") == 0) {
                printf(" %c ", dict->items[i].value[k].s);

            } else if (strcmp(dict->items[i].datatype, "double") == 0) {
                printf(" %lf ", dict->items[i].value[k].d);
            }

    }
    printf("\n");
}
}
```

This code defines a function to print all items in the custom dictionary, displaying each key, datatype, and corresponding values based on the datatype.

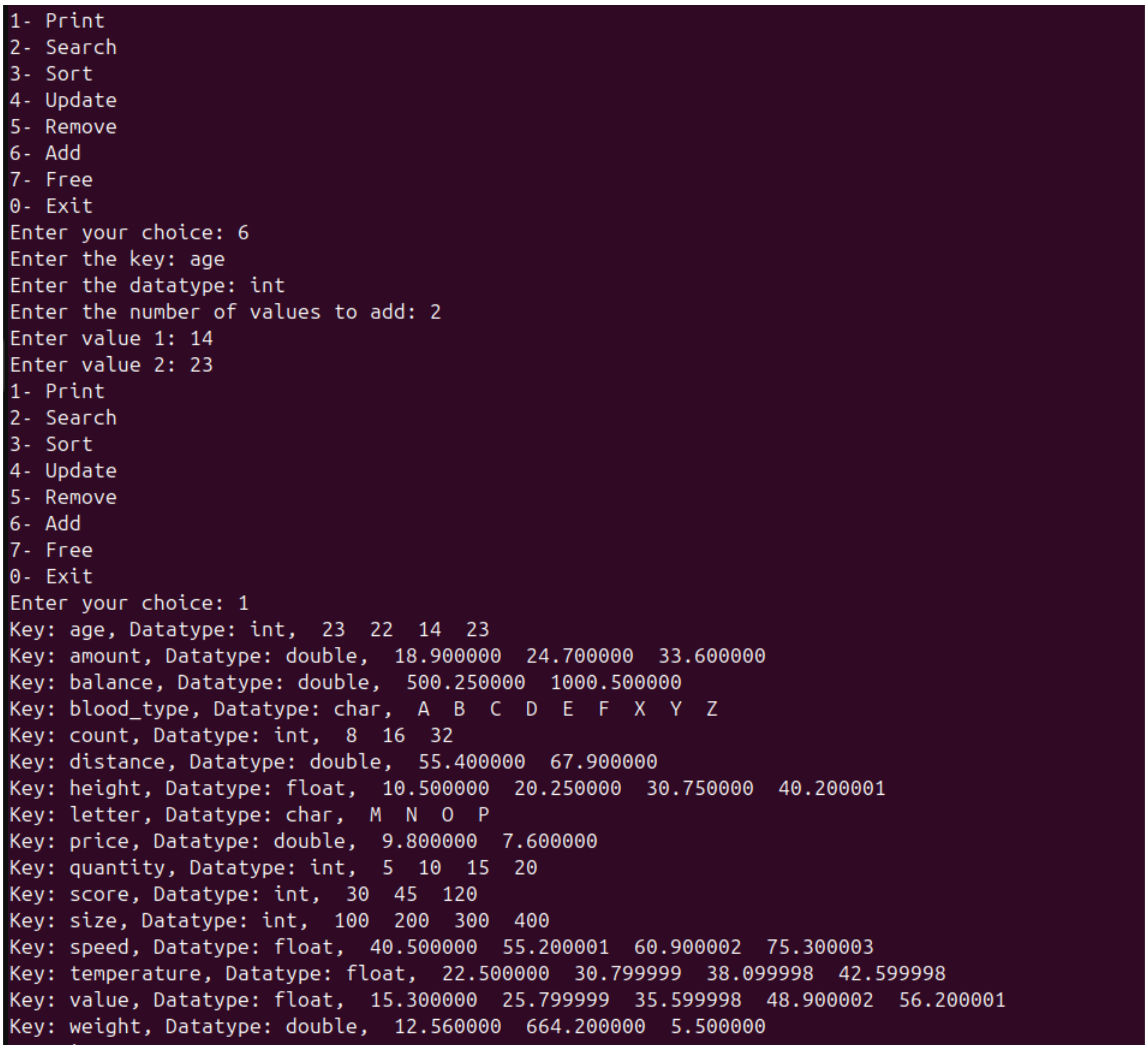


```
1- Print
2- Search
3- Sort
4- Update
5- Remove
6- Add
7- Free
0- Exit
Enter your choice: 1
Key: age, Datatype: int,  25  12  556  1  192561
Key: weight, Datatype: double,  12.560000  664.200000  5.500000
Key: blood_type, Datatype: char,  A  B  C  D  E  F  X  Y  Z
Key: height, Datatype: float,  10.500000  20.250000  30.750000  40.200001
Key: score, Datatype: int,  30  45  120
Key: price, Datatype: double,  9.800000  7.600000
Key: value, Datatype: float,  15.300000  25.799999  35.599998  48.900002  56.200001
Key: quantity, Datatype: int,  5  10  15  20
Key: amount, Datatype: double,  18.900000  24.700000  33.600000
Key: letter, Datatype: char,  M  N  O  P
Key: temperature, Datatype: float,  22.500000  30.799999  38.099998  42.599998
Key: count, Datatype: int,  8  16  32
Key: distance, Datatype: double,  55.400000  67.900000
Key: speed, Datatype: float,  40.500000  55.200001  60.900002  75.300003
Key: size, Datatype: int,  100  200  300  400
Key: balance, Datatype: double,  500.250000  1000.500000
```

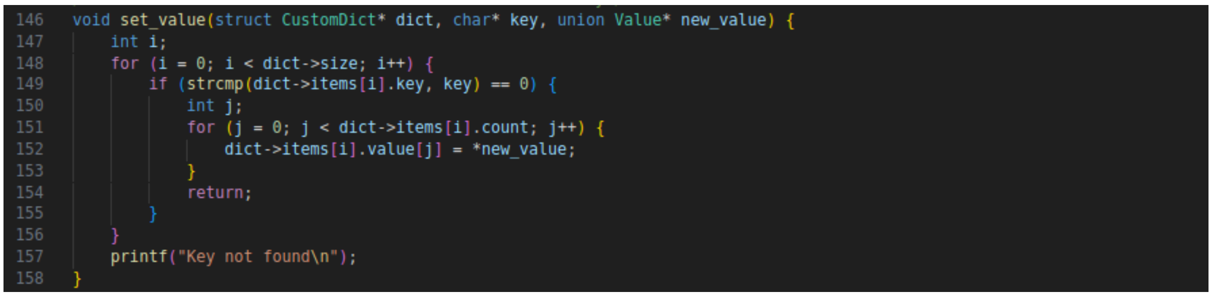


```c
void add_item(struct CustomDict* dict, char* key, char* datatype, union Value* new_values, int new_count) {
    int i, j;
    /*Check if the key already exists*/
    for (i = 0; i < dict->size; i++) {
        if (strcmp(dict->items[i].key, key) == 0) {
            /*Add the new values to the existing item*/
            dict->items[i].value = realloc(dict->items[i].value, (dict->items[i].count + new_count) * sizeof(Value));
            for (j = 0; j < new_count; j++) {
                dict->items[i].value[dict->items[i].count + j] = new_values[j];
            }
            dict->items[i].count += new_count;
            return;
        }
    }

    /*Add a new item to the dictionary*/
    dict->items = realloc(dict->items, (dict->size + 1) * sizeof(Item));
    dict->items[dict->size].key = malloc((strlen(key) + 1) * sizeof(char));
    strcpy(dict->items[dict->size].key, key);
    dict->items[dict->size].datatype = malloc((strlen(datatype) + 1) * sizeof(char));
    strcpy(dict->items[dict->size].datatype, datatype);
    dict->items[dict->size].value = malloc(new_count * sizeof(Value));
    for (j = 0; j < new_count; j++) {
        dict->items[dict->size].value[j] = new_values[j];
    }
    dict->items[dict->size].count = new_count;
    dict->size++;
}
```

This code defines a function to add an item to the custom dictionary, either by appending values to an existing item or by adding a new item if the key doesn't already exist.

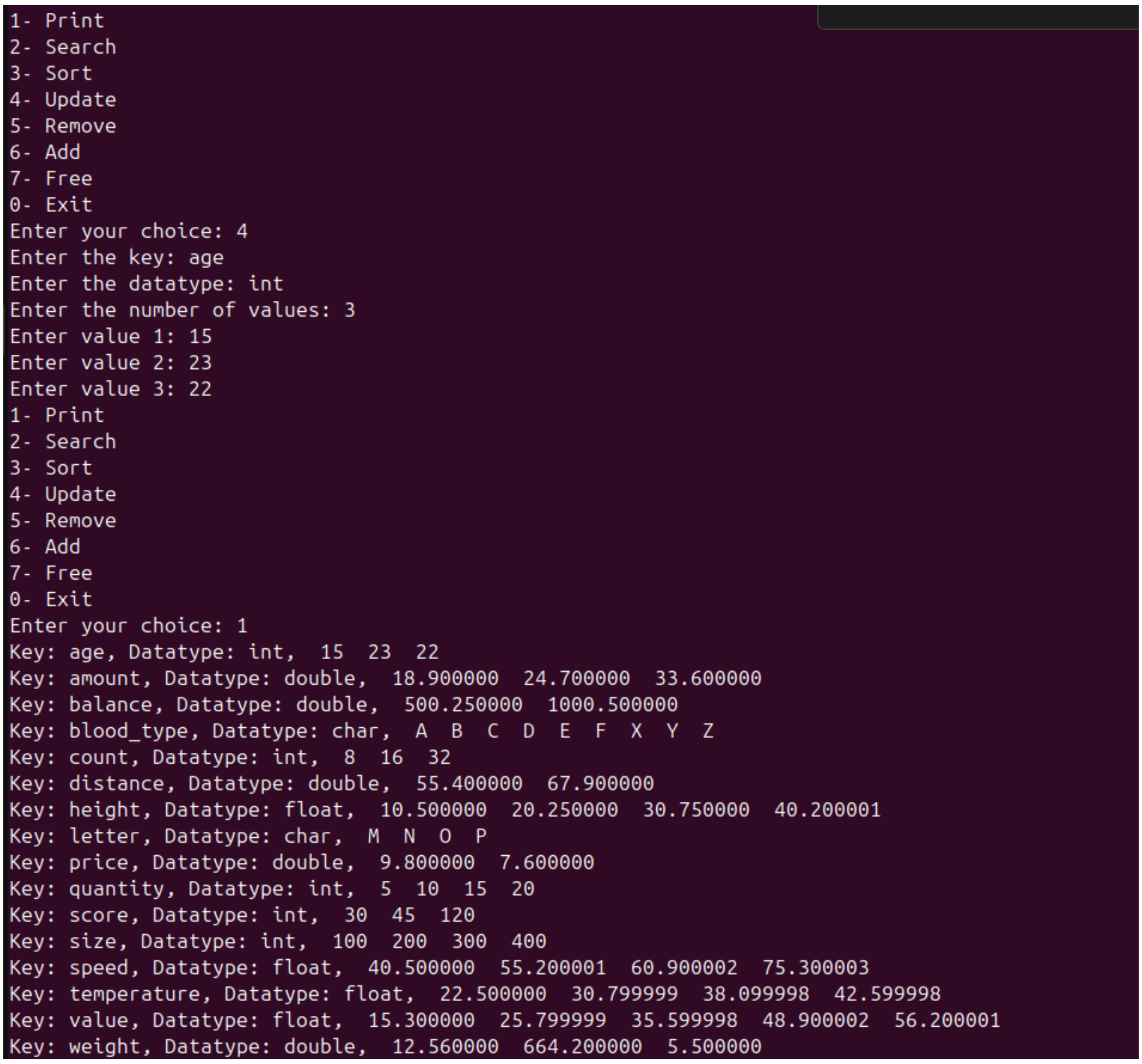


```
1- Print
2- Search
3- Sort
4- Update
5- Remove
6- Add
7- Free
0- Exit
Enter your choice: 6
Enter the key: age
Enter the datatype: int
Enter the number of values to add: 2
Enter value 1: 14
Enter value 2: 23
1- Print
2- Search
3- Sort
4- Update
5- Remove
6- Add
7- Free
0- Exit
Enter your choice: 1
Key: age, Datatype: int,  23  22  14  23
Key: amount, Datatype: double,  18.900000  24.700000  33.600000
Key: balance, Datatype: double,  500.250000  1000.500000
Key: blood_type, Datatype: char,  A  B  C  D  E  F  X  Y  Z
Key: count, Datatype: int,  8  16  32
Key: distance, Datatype: double,  55.400000  67.900000
Key: height, Datatype: float,  10.500000  20.250000  30.750000  40.200001
Key: letter, Datatype: char,  M  N  O  P
Key: price, Datatype: double,  9.800000  7.600000
Key: quantity, Datatype: int,  5  10  15  20
Key: score, Datatype: int,  30  45  120
Key: size, Datatype: int,  100  200  300  400
Key: speed, Datatype: float,  40.500000  55.200001  60.900002  75.300003
Key: temperature, Datatype: float,  22.500000  30.799999  38.099998  42.599998
Key: value, Datatype: float,  15.300000  25.799999  35.599998  48.900002  56.200001
Key: weight, Datatype: double,  12.560000  664.200000  5.500000
```

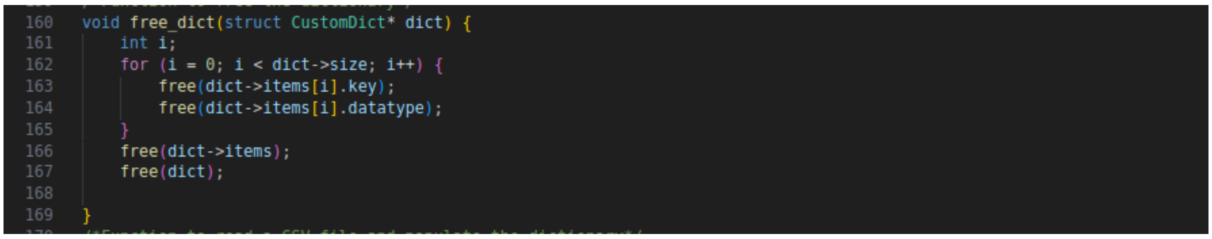


```c
void set_value(struct CustomDict* dict, char* key, union Value* new_value) {
    int i;
    for (i = 0; i < dict->size; i++) {
        if (strcmp(dict->items[i].key, key) == 0) {
            int j;
            for (j = 0; j < dict->items[i].count; j++) {
                dict->items[i].value[j] = *new_value;
            }
            return;
        }
    }
    printf("Key not found\n");
}
```

This code defines a function to set the value of all elements associated with a specific key in the custom dictionary, updating them to the new value provided, and printing a message if the key is not found.

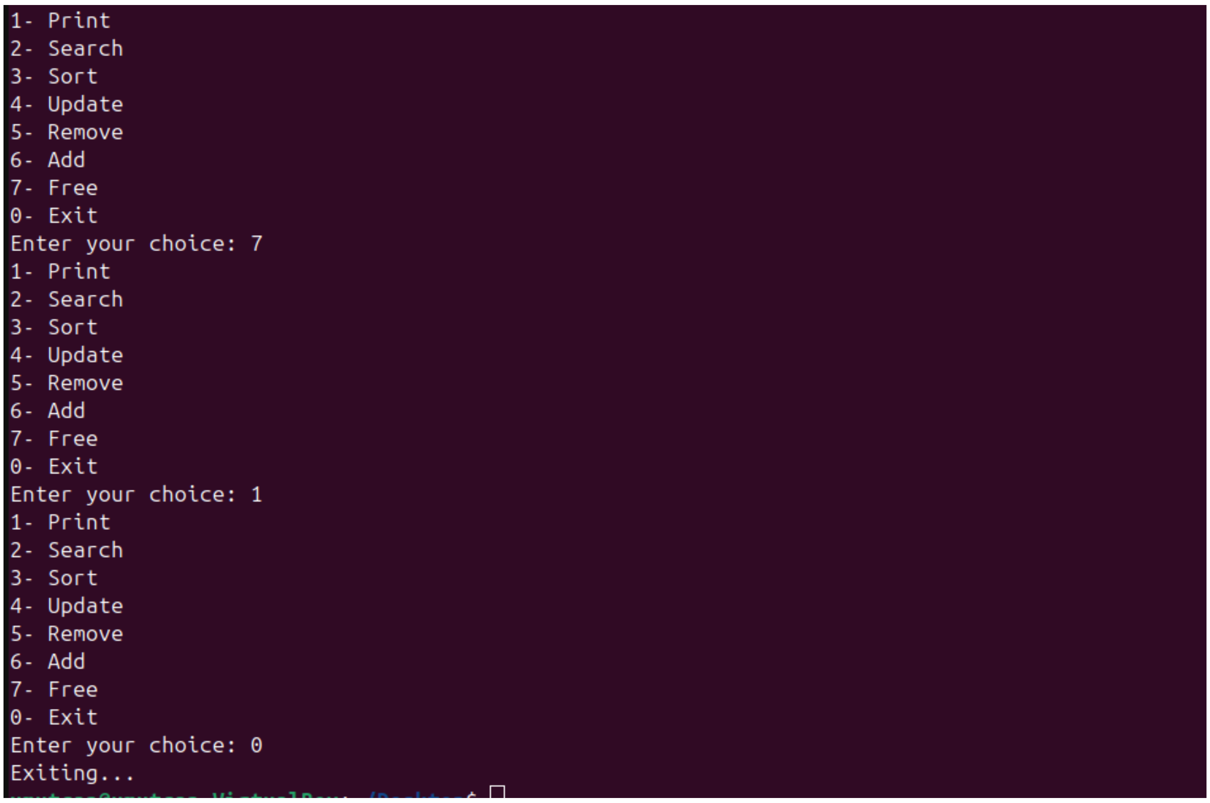


```
1- Print
2- Search
3- Sort
4- Update
5- Remove
6- Add
7- Free
0- Exit
Enter your choice: 4
Enter the key: age
Enter the datatype: int
Enter the number of values: 3
Enter value 1: 15
Enter value 2: 23
Enter value 3: 22
1- Print
2- Search
3- Sort
4- Update
5- Remove
6- Add
7- Free
0- Exit
Enter your choice: 1
Key: age, Datatype: int,  15  23  22
Key: amount, Datatype: double,  18.900000  24.700000  33.600000
Key: balance, Datatype: double,  500.250000  1000.500000
Key: blood_type, Datatype: char,  A  B  C  D  E  F  X  Y  Z
Key: count, Datatype: int,  8  16  32
Key: distance, Datatype: double,  55.400000  67.900000
Key: height, Datatype: float,  10.500000  20.250000  30.750000  40.200001
Key: letter, Datatype: char,  M  N  O  P
Key: price, Datatype: double,  9.800000  7.600000
Key: quantity, Datatype: int,  5  10  15  20
Key: score, Datatype: int,  30  45  120
Key: size, Datatype: int,  100  200  300  400
Key: speed, Datatype: float,  40.500000  55.200001  60.900002  75.300003
Key: temperature, Datatype: float,  22.500000  30.799999  38.099998  42.599998
Key: value, Datatype: float,  15.300000  25.799999  35.599998  48.900002  56.200001
Key: weight, Datatype: double,  12.560000  664.200000  5.500000
```

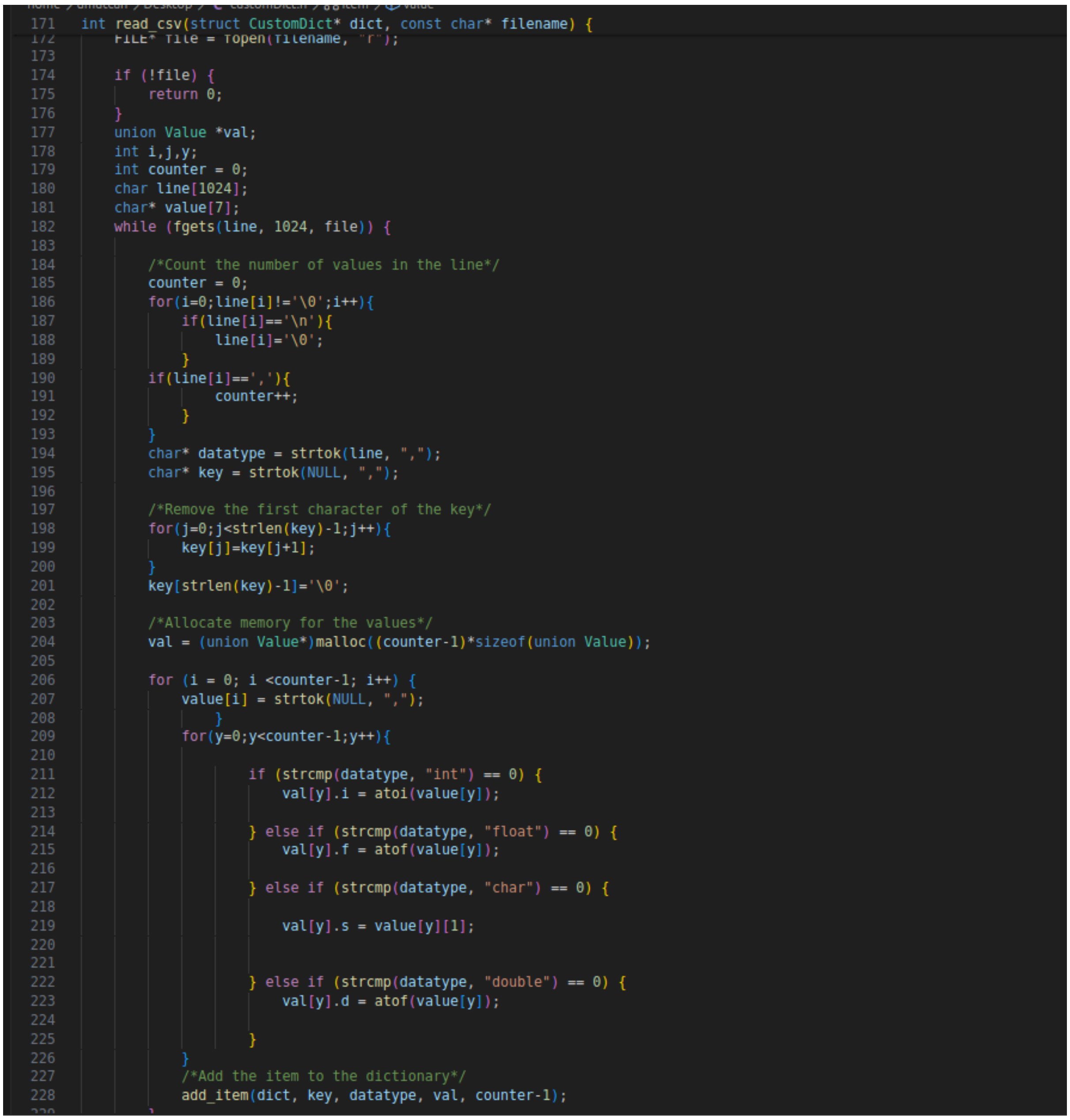


```c
void free_dict(struct CustomDict* dict) {
    int i;
    for (i = 0; i < dict->size; i++) {
        free(dict->items[i].key);
        free(dict->items[i].datatype);
    }
    free(dict->items);
    free(dict);

}
```

This code defines a function to free the memory allocated for the custom dictionary, including all keys and datatypes for each item, as well as the dictionary structure itself.



```
1- Print
2- Search
3- Sort
4- Update
5- Remove
6- Add
7- Free
0- Exit
Enter your choice: 7
1- Print
2- Search
3- Sort
4- Update
5- Remove
6- Add
7- Free
0- Exit
Enter your choice: 1
1- Print
2- Search
3- Sort
4- Update
5- Remove
6- Add
7- Free
0- Exit
Enter your choice: 0
Exiting...
[illegible]
```



```c
int read_csv(struct CustomDict* dict, const char* filename) {
    FILE* file = fopen(filename, "r");

    if (!file) {
        return 0;
    }
    union Value *val;
    int i,j,y;
    int counter = 0;
    char line[1024];
    char* value[7];
    while (fgets(line, 1024, file)) {

        /*Count the number of values in the line*/
        counter = 0;
        for(i=0;line[i]!='\0';i++){
            if(line[i]=='\n'){
                line[i]='\0';
            }
        if(line[i]==','){
                counter++;
            }
        }
        char* datatype = strtok(line, ",");
        char* key = strtok(NULL, ",");

        /*Remove the first character of the key*/
        for(j=0;j<strlen(key)-1;j++){
            key[j]=key[j+1];
        }
        key[strlen(key)-1]='\0';

        /*Allocate memory for the values*/
        val = (union Value*)malloc((counter-1)*sizeof(union Value));

        for (i = 0; i <counter-1; i++) {
            value[i] = strtok(NULL, ",");
                }
            for(y=0;y<counter-1;y++){

                    if (strcmp(datatype, "int") == 0) {
                        val[y].i = atoi(value[y]);

                    } else if (strcmp(datatype, "float") == 0) {
                        val[y].f = atof(value[y]);

                    } else if (strcmp(datatype, "char") == 0) {

                        val[y].s = value[y][1];


                    } else if (strcmp(datatype, "double") == 0) {
                        val[y].d = atof(value[y]);

                    }
            }
            /*Add the item to the dictionary*/
            add_item(dict, key, datatype, val, counter-1);
```

This function reads a CSV file, parses its content, extracts data types, keys, and values, then populates a custom dictionary accordingly. It handles data type conversions for values and dynamically allocates memory to store them in the dictionary.