Programming Prep Documentation

1. **Simulate the behavior of cp command in linux.**

LOGIC:

1. Open the src_file in read mode and the dest_file in write mode
2. Check if no. of arguments match the requirements for the code to run
3. Check if src or dest files are NULL and display corresponding error messages and exit.
4. Else read character by character from src_file until EOF is reached and consecutively write the characters read in dest_file
5. Close both the files and exit.

CODE:

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char const *argv[])
{
   FILE *src, *dest;
   char c;

   if (argc != 3)
   {
      printf("Number of arguments don't match.\nSyntax: ./copy <src_file>
<dest_file>\nAborting.\n");
      return 0;
   }
   src = fopen(argv[1], "r");
   dest = fopen(argv[2], "w");

   if (src == NULL || dest == NULL)
   {
      printf("Unable to open source/destination file!\n");
      return 0;
   }

   while ((c = fgetc(src)) != EOF)
      fputc(c, dest);

   printf("Copied file successfully!\n");

   fclose(src);
   fclose(dest);
   return 0;
}
```

OUTPUT:

```
pav@Pavendhan-PAV:~/CS/OS/Aug 7$ ls *.txt
sample.txt
pav@Pavendhan-PAV:~/CS/OS/Aug 7$ cat sample.txt
COPY TEXT
pav@Pavendhan-PAV:~/CS/OS/Aug 7$ ./copy sample.txt new_sample.txt
Copied file successfully!
pav@Pavendhan-PAV:~/CS/OS/Aug 7$ ls *.txt
new_sample.txt  sample.txt
pav@Pavendhan-PAV:~/CS/OS/Aug 7$ cat new_sample.txt
COPY TEXT
```

2. **Simulate the behavior of rm command in linux.**

LOGIC:

1. Check if no. of arguments match the requirements for the code to run
2. Remove argv[1] which points to the file to be removed using remove, if removed it returns 0, based on this run if loop and print corresponding message to the user.

CODE:

```
#include <stdio.h>
int main(int argc, char *argv[])
{
   if (argc != 2)
   {
      printf("Number of arguments don't match. Aborting.\n");
      return 0;
   }
   if (!remove(argv[1]))
      printf("File deleted successfully!\n");
   else
      printf("Error in deleting file\n");

   return 0;
}
```

OUTPUT:

```
pav@Pavendhan-PAV:~/CS/OS/Aug 7$ ls *.txt
new_sample.txt  sample.txt
pav@Pavendhan-PAV:~/CS/OS/Aug 7$ ./rem new_sample.txt
File deleted successfully!
pav@Pavendhan-PAV:~/CS/OS/Aug 7$ ls
'Aug 7.zip'    fpsort.c                 remove.c            templatesort.cpp
 copy          fsort                    sample.txt          tsort
 copy.c        progprepassignment.pdf   sort                typesort.cpp
 copy.png      rem                      sortapplication.c
```

3. **Sort an array of varying numbers of integers in ascending or descending order.**

LOGIC:

1. From the command line, get the integers and the order of sort (ascending/descending).
2. Check for minimum arguments and print corresponding error messages and abort.
3. Then, convert char* to int and pass it to the bubble sort function.
4. Based on whether the array is to be sorted in ascending or descending order, sort the array using bubble sort

CODE:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

int main(int argc, char** argv)
{
        if (argc <= 3)
  {
    printf("syntax: ./sort <size> <0-des/1-asc> [<integers>/<decimals>/<strings>]\n");
    return EXIT_FAILURE;
  }
        int n=((int)*argv[1] - 48)+3;
        //printf("%d",n);

        if(n != argc)
        {
                printf("Number of arguments mismatch! Aborting.\n");
                return 0;
        }

        int choice=((int)*argv[2] - 48);

        switch(choice)
        {
                case 0:
                        printf("DESCENDING ORDER: ");
                        for(int i=3;i<n;i++)
                        {
                                for(int j=3;j<n-1;j++)
                                {
                                        //printf("%dhjjjjj",(int)*argv[j+1]-48);
                                        if((int)*argv[j]<(int)*argv[j+1])
                                        {
                                                int t=((int)*argv[j]-48);
                                                //printf("%d",t);
                                                *argv[j]=((int)*argv[j+1]);
                                                *argv[j+1]=t+48;
```

```
                        }
                    }
                }
                for(int i=3;i<n;i++)
                    printf("%d ",((int)*argv[i])-48);

                printf("\n");
                break;

            case 1:
                printf("ASCENDING ORDER: ");
                for(int i=3;i<n;i++)
                    for(int j=3;j<n-1;j++)
                    {
                        if((int)*argv[j]>(int)*argv[j+1])
                        {
                            int t=((int)*argv[j]-48);
                            //printf("%d",t);
                            *argv[j]=((int)*argv[j+1]);
                            *argv[j+1]=t+48;
                        }
                    }
                for(int i=3;i<n;i++)
                    printf("%d ",((int)*argv[i])-48);
                printf("\n");
                break;

            default:
                printf("\nNO SUCH CASE");
                break;
        }
}
```

OUTPUT:

```
pav@Pavendhan-PAV:~/CS/OS/Aug 7$ gcc sortapplication.c -o sort
pav@Pavendhan-PAV:~/CS/OS/Aug 7$ ./sort
syntax: ./sort <size> <0-des/1-asc> [<integers>/<decimals>/<strings>]
pav@Pavendhan-PAV:~/CS/OS/Aug 7$ ./sort 3 1 3 2
Number of arguments mismatch! Aborting.
pav@Pavendhan-PAV:~/CS/OS/Aug 7$ ./sort 3 1 3 2 1
ASCENDING ORDER: 1 2 3
pav@Pavendhan-PAV:~/CS/OS/Aug 7$ ./sort 3 0 1 2 3
DESCENDING ORDER: 3 2 1
```

4. **implement the above sorting (both ascending or descending) using only function internally for sorting logic - Function pointers**

LOGIC:

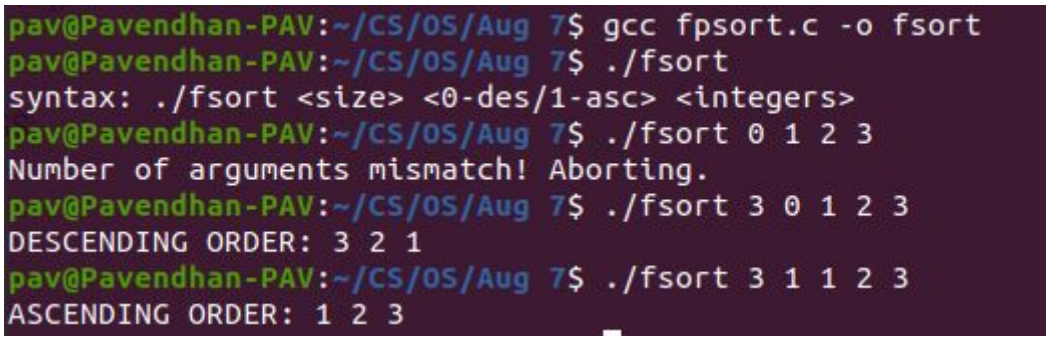
1. From the command line, get the integers and the order of sort (ascending/descending).
2. Check for minimum arguments and print corresponding error messages and abort.
3. Then, convert char* to int and pass it to the bubble sort function.
4. Based on whether the array is to be sorted in ascending or descending order, sort the array using bubble sort, But the change here is there is one code for bubble sort which points to one more function (FUNCTION POINTER)
5. This function “compare” lets the program do sorting through proper swap conditions based on the type of sort in command line

CODE:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

int asc(int a, int b)
{
   return a > b;
}

int desc(int a, int b)
{
   return b > a;
}

void bubblesort(int arr[], int size, int (*compare)(int a, int b))
{
        for(int i=0;i<size;i++)
        {
                for(int j=0;j<size-1;j++)
                {
                        if((*compare)(arr[j],arr[j+1]))
                        {
                                int t=arr[j];
                                //printf("%d",t);
                                arr[j]=arr[j+1];
                                arr[j+1]=t;
                        }
                }
        }
        for(int i=0;i<size;i++)
                printf("%d ",arr[i]);
```

```
}

int main(int argc, char** argv)
{
        if (argc <= 2)
   {
      printf("syntax: ./fsort <size> <0-des/1-asc> <integers>\n");
      return EXIT_FAILURE;
   }

        int size=((int)*argv[1] - 48);

        if(size != argc-3)
        {
                printf("Number of arguments mismatch! Aborting.\n");
                return 0;
        }

        int choice=((int)*argv[2] - 48);

        int arr[size];
   for (int i = 0; i < size; i++)
      arr[i] = ((int)*argv[i+3] -48);

        switch(choice)
        {
                case 0:
                        printf("DESCENDING ORDER: ");
                        bubblesort(arr, size, desc);
                        printf("\n");
                        break;

                case 1:
                        printf("ASCENDING ORDER: ");
                        bubblesort(arr, size, asc);
                        printf("\n");
                        break;

                default:
                        printf("\nNO SUCH CASE");
                        printf("\n");
                        break;
        }
        return EXIT_SUCCESS;
}
```

OUTPUT:



5. **Sorting an array of integers or floating point or characters passed at the command line.**

LOGIC:

Here, we have to use function overloading. Function overloading is a feature where two or more functions can have the same name but different parameters. It can be considered as an example of polymorphism.

1. From the command line, get the data and the order of sort (ascending/descending).
2. Then based on whether the given data is integer, double or string, we define the array and pass it to the bubble sort function.
3. Based on whether the array is to be sorted in ascending or descending order, print the sorted array.
4. While getting input from the command line, we also have to check some conditions like if the number of arguments is greater than 2 and the second argument is nothing other than the specified two options.
5. If any irregularities are there while taking input, we have to return the proper way to use the command.

CODE:

```
#include <iostream>
#include <string.h>

using namespace std;

void bubbleSort(int arr[], int size)
{
   for (int i = 0; i < size; i++)
   {
      for (int j = 0; j < size - 1; j++)
      {
         if (arr[j] > arr[j + 1])
         {
            int temp = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = temp;
```

```
        }
      }
   }
}

void bubbleSort(double arr[], int size)
{
   for (int i = 0; i < size; i++)
   {
      for (int j = 0; j < size - 1; j++)
      {
         if (arr[j] > arr[j + 1])
         {
            double temp = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = temp;
         }
      }
   }
}

void bubbleSort(string arr[], int size)
{
   for (int i = 0; i < size; i++)
   {
      for (int j = 0; j < size - 1; j++)
      {
         if (arr[j] > arr[j + 1])
         {
            string temp = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = temp;
         }
      }
   }
}


void sort(int arr[], int size, char const *argv)
{
   bubbleSort(arr, size);

   if (strcmp(argv, "1") == 0)
      for (int i = 0; i < size; i++)
         cout << arr[i] << " ";
```

```
    if (strcmp(argv, "0") == 0)
        for (int i = size - 1; i >= 0; i--)
            cout << arr[i] << " ";

    cout << endl;
}

void sort(double arr[], int size, char const *argv)
{
    bubbleSort(arr, size);

    if (strcmp(argv, "1") == 0)
        for (int i = 0; i < size; i++)
            cout << arr[i] << " ";

    if (strcmp(argv, "0") == 0)
        for (int i = size - 1; i >= 0; i--)
            cout << arr[i] << " ";

    cout << endl;
}

void sort(string arr[], int size, char const *argv)
{
    bubbleSort(arr, size);

    if (strcmp(argv, "1") == 0)
        for (int i = 0; i < size; i++)
            cout << arr[i] << " ";

    if (strcmp(argv, "0") == 0)
        for (int i = size - 1; i >= 0; i--)
            cout << arr[i] << " ";

    cout << endl;
}

void error(int x)
{
            if(x==0)
    cout << "syntax: ./tsort <0-des/1-asc> <integers>/<decimals>/<strings>]" << endl;
    else
    cout<<"Array isn't of same datatype! Aborting."<<endl;
}

int main(int argc, char const *argv[])
```

```
{
   int size = argc - 2; //2 is offset
   int Count = 0;

   if (argc <= 2)
   {
      error(0);
      return EXIT_FAILURE;
   }

i:       if(Count<size)
         {
                  for(int i=0; i<size; i++)
                  {
                           if (strspn(argv[i+2], "-0123456789") == strlen(argv[i+2]))
                                    Count++;
                  }

                  if(Count!=size && Count>0)
                  {
                           error(1);
                           return EXIT_FAILURE;
                  }

                  else if(Count == size)
                  {
                           int arr[size];
                           for (int i = 0; i < size; i++)
         arr[i] = stoi(argv[i+2]);

         sort(arr, size, argv[1]);
         return EXIT_SUCCESS;
                  }

                  else
                           goto f;
         }


 f:      for(int i=0; i<size; i++)
                  {
                           if( strspn(argv[i+2],".") == 1 || strspn(argv[i+2],"-0.") == 2 || (
strspn(argv[i+2],"-0.")==3 && strspn(argv[i+2],"-") == 1 ) )
                           {
                                    if (strspn(argv[i+2], "-.0123456789") == strlen(argv[i+2]))
                                             Count++;
```

```
                }
        }

        if(Count!=size && Count>0)
        {
                error(1);
                return EXIT_FAILURE;
        }

        else if(Count == size)
        {
                double arr[size];
                for (int i = 0; i < size; i++)
    arr[i] = stod(argv[i+2]);

    sort(arr, size, argv[1]);
    return EXIT_SUCCESS;
        }

        else
                goto s;


s:      for(int i=0; i<size; i++)
        {
                if (isalpha(*argv[i+2]))
                        Count++;
        }

        if(Count!=size && Count>0)
        {
                error(1);
                return EXIT_FAILURE;
        }

        else if(Count == size)
        {
                string arr[size];
                for (int i = 0; i < size; i++)
    arr[i] = (string)(argv[i+2]);

    sort(arr, size, argv[1]);
    return EXIT_SUCCESS;
        }
}
```

OUTPUT:

```
pav@Pavendhan-PAV:~/CS/OS/Aug 7$ ./tsort
syntax: ./tsort <0-des/1-asc> <integers>/<decimals>/<strings>]
pav@Pavendhan-PAV:~/CS/OS/Aug 7$ ./tsort 0 1 2 3
3 2 1
pav@Pavendhan-PAV:~/CS/OS/Aug 7$ ./tsort 3 3 2 1

pav@Pavendhan-PAV:~/CS/OS/Aug 7$ ./tsort
syntax: ./tsort <0-des/1-asc> <integers>/<decimals>/<strings>]
pav@Pavendhan-PAV:~/CS/OS/Aug 7$ ./tsort 0 1 2 3
3 2 1
pav@Pavendhan-PAV:~/CS/OS/Aug 7$ ./tsort 1 3 2 1
1 2 3
pav@Pavendhan-PAV:~/CS/OS/Aug 7$ ./tsort 0 x y z
z y x
pav@Pavendhan-PAV:~/CS/OS/Aug 7$ ./tsort 1 z y x
x y z
pav@Pavendhan-PAV:~/CS/OS/Aug 7$ ./tsort 0 0.1 0.2 -0.5
0.2 0.1 -0.5
pav@Pavendhan-PAV:~/CS/OS/Aug 7$ ./tsort 1 0.18 0.27 -0.5
-0.5 0.18 0.27
pav@Pavendhan-PAV:~/CS/OS/Aug 7$ ./tsort 1 0.18 0.27 r
Array isn't of same datatype! Aborting.
```

6. **Same as above but you should define sort function only once internally and leave it to the compiler to generate data type specific functions.**

LOGIC:

Here, we have to use a function template. A template is a simple and yet very powerful tool. The simple idea is to pass data type as a parameter so that we don't need to write the same code for different data types.

1. From the command line, get the data and the order of sort (ascending/descending).
2. Then based on whether the given data is integer, double or string, we define the array and pass it to the bubble sort template function.
3. Based on whether the array is to be sorted in ascending or descending order, print the sorted array.
4. While getting input from the command line, we also have to check some conditions like if the number of arguments is greater than 2 and the second argument is nothing other than the specified two options.
5. If any irregularities are there while taking input, we have to return the proper way to use the command.

CODE:

```
#include <iostream>
#include <string.h>

using namespace std;
```

```
template <class T>
void bubbleSort(T arr[], int size)
{
   for (int i = 0; i < size; i++)
   {
      for (int j = 0; j < size - 1; j++)
      {
         if (arr[j] > arr[j + 1])
         {
            T temp = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = temp;
         }
      }
   }
}

template <class T>
void sort(T arr[], int size, char const *argv)
{
   bubbleSort(arr, size);

   if (strcmp(argv, "1") == 0)
      for (int i = 0; i < size; i++)
         cout << arr[i] << " ";

   if (strcmp(argv, "0") == 0)
      for (int i = size - 1; i >= 0; i--)
         cout << arr[i] << " ";

   cout << endl;
}

void error(int x)
{
         if(x==0)
   cout << "syntax: ./tsort <0-des/1-asc> [<integers>/<decimals>/<strings>]" << endl;
   else
   cout<<"Array isn't of same datatype! Aborting."<<endl;
}

int main(int argc, char const *argv[])
{
   int size = argc - 2; //2 is offset
   int count = 0;
```

```
    if (argc <= 2)
    {
        error(0);
        return EXIT_FAILURE;
    }

i:      if(count<size)
        {
                for(int i=0; i<size; i++)
                {
                        if (strspn(argv[i+2], "-0123456789") == strlen(argv[i+2]))
                                count++;
                }

                if(count!=size && count>0)
                {
                        error(1);
                        return EXIT_FAILURE;
                }

                else if(count == size)
                {
                        int arr[size];
                        for (int i = 0; i < size; i++)
        arr[i] = stoi(argv[i+2]);

        sort<int>(arr, size, argv[1]);
        return EXIT_SUCCESS;
                }

                else
                        goto f;
        }


 f:     for(int i=0; i<size; i++)
                {
                        if( strspn(argv[i+2],".") == 1 || strspn(argv[i+2],"-0.") == 2 || (
strspn(argv[i+2],"-0.")==3 && strspn(argv[i+2],"-") == 1 ) )
                        {
                                if (strspn(argv[i+2], "-.0123456789") == strlen(argv[i+2]))
                                        count++;
                        }
                }
```

```
		if(count!=size && count>0)
		{
			error(1);
			return EXIT_FAILURE;
		}

		else if(count == size)
		{
			double arr[size];
			for (int i = 0; i < size; i++)
	arr[i] = stod(argv[i+2]);

	sort<double>(arr, size, argv[1]);
	return EXIT_SUCCESS;
		}

		else
			goto s;

 s:	for(int i=0; i<size; i++)
		{
			if (isalpha(*argv[i+2]))
				count++;
		}

		if(count!=size && count>0)
		{
			error(1);
			return EXIT_FAILURE;
		}

		else if(count == size)
		{
			string arr[size];
			for (int i = 0; i < size; i++)
	arr[i] = (string)(argv[i+2]);

	sort<string>(arr, size, argv[1]);
	return EXIT_SUCCESS;
		}
}
```

OUTPUT:

```
pav@Pavendhan-PAV:~/CS/OS/Aug 7$ g++ templatesort.cpp -o tsort
pav@Pavendhan-PAV:~/CS/OS/Aug 7$ ./tsort
syntax: ./tsort <0-des/1-asc> [<integers>/<decimals>/<strings>]
pav@Pavendhan-PAV:~/CS/OS/Aug 7$ ./tsort 0 1 2 3
3 2 1
pav@Pavendhan-PAV:~/CS/OS/Aug 7$ ./tsort 1 3 2 -1
-1 2 3
pav@Pavendhan-PAV:~/CS/OS/Aug 7$ ./tsort 0 x y z
z y x
pav@Pavendhan-PAV:~/CS/OS/Aug 7$ ./tsort 1 z y x
x y z
pav@Pavendhan-PAV:~/CS/OS/Aug 7$ ./tsort 1 0.18 0.27 -0.5
-0.5 0.18 0.27
pav@Pavendhan-PAV:~/CS/OS/Aug 7$ ./tsort 0 0.1 0.2 -0.5
0.2 0.1 -0.5
pav@Pavendhan-PAV:~/CS/OS/Aug 7$ ./tsort 0 x y 1
Array isn't of same datatype! Aborting.
```