DATE: 03-10-2024

FAMILIARIZATION : 1

FAMILIARISE GCC

AIM: Advanced use of gcc : Important Options -o, -c, -D, -l, -I, -g, -O, -save-temps, -pg

DESCRIPTION:

- gcc stands for GNU C/C++ Compiler
- A popular console-based compiler for Unix/Linux based platforms andothers; can cross-compile code for various architectures gcc performs all of these:

  preprocessing
  compilation
  assembly and linking

Options: -c

- gcc performs compilation and assembly of the source file without linking.
- The output are usually object code files; they can later be linked and form the desired executables.
- Generates one object file per source file keeping the same prefix (before) of the filename.

Options: -D

- gcc defines a macro to be used by preprocessor.

Options: -o

- Places resulting file into the filename specified instead of the default one.
- Can be used with any generated files (object, executables, assembly, etc.)
- If you have the file called source.c; the defaults are: –source.o
  if -c was specified –a.out if executable
- These can be overridden with the -o option.

Options: -g

- Includes debugging information in the generated object code. This information can later be used in gdb.
- gcc allows to use -g with the optimization turned on (-O) in case there is a need to debug or trace the optimized code

Options: -ggdb

- In addition to -g produces the most GDB friendly output if enabled

Options: -O, -O1, -O2, -O3, -O0, -Os

- Various levels of optimization of the code
- -O1 to -O3 are various degrees of optimization targeted for speed
- If -O is added, then the code size is considered
- -O0 means “no optimization”
- -Os targets generated code size (forces not to use optimizations resulting in bigger code).

Options: -I

- gcc to look for include files (.h).
- Can be any number of these.
- Usually needed when including headers from various-depth directories in non-standard places without necessity specifying these directories with the .c files themselves.

Options: -pg

- gcc performs to compile a source file for profiling,specify the pg option when run the compiler.

Options: -save-temps

- gcc stores the normally temporary intermediate files permanently.

RESULT :

Familiarised gcc

DATE: 03-10-2024

FAMILIARIZATION : 2

FAMILIARISE GDB

AIM: Familiarisation with gdb :

Important Commands -break, run, next, print, display, help

DESCRIPTION:

gdb is the acronym for GNU Debugger. This tool helps to debug the programs written in C, C++, Ada, Fortran, etc. The console can be opened using the gdb command on terminal.

Commands

- run [args] : This command runs the current executable file.eg: The program was executed twice, one with the command line argument 10 and another with the command line argument , and their corresponding outputs were printed.
- break : The command break [function name] helps to pause the program during execution when it starts to execute the function. It helps to debug the program at that point.
- next or n : This command helps to execute the next instruction after it encounters the break point.Whenever it encounters the above command, it executes the next instruction of the executable by printing the line in execution.
- display: These command enables automatic displaying of expressions each time whenever the execution encounters a break point or the n command. The undisplay command is used to remove display expressions.
- print : This command prints the value of a given expression. The display command prints all the previously displayed values whenever it encounters a break point or the next command, whereas the print command saves all the previously displayed values and prints whenever it is called print[Expression].
- help : It launches the manual of gdb along with all list of classes of individual commands.

RESULT :

Familiarised gdb.

Date: 03-10-2024

FAMILIARIZATION : 3

FAMILIARISE GPROF

AIM: Using gprof : Compile, Execute and Profile

DESCRIPTION

Compile

- The first step in generating profile information for your program is to compile and link it with profiling enabled.
- To compile a source file for profiling, specify the -pgoption when run the compiler.
- To link the program for profiling, if you use a compiler such as cc to do the linking, simply specify -pg in addition to usual options. The same option, -pg, alters either compilation or linking to do what is necessary for profiling.

Execute

- Once the program is compiled for profiling, you must run it in order to generate the information that gprof needs. Simply run the program as usual, using the normal arguments, file names, etc. The program should run normally, producing the same output as usual. It will, however, run somewhat slower than normal because of the time spent collecting and writing the profile data.

- The way to run the program—the arguments and input that you give it—may have a dramatic effect on what the profile information shows. The profile data will describe the parts of the program that were activated for the particular input you use. For example, if the first command you give to your program is to quit, the profile data will show the time used in initialization and in cleanup, but not much else.

Profile

- Describes the GNU profiler,gpof, and how can use it to determine which parts of a program are taking most of the execution time. We assume that to know how to write, compile, and execute programs. GNU gprof was written by Jay Fenlason.

RESULT:

Familiarised gprof

RESULT :

Program is executed successfully and output is obtained .

```
    else
    {
        printf("Search Unsuccessfull, Element Not Found ");
    }
}
```

RESULT:

Program is executed successfully and output is obtained .

DATE: 17-10-2024

PROGRAM NO: 5.1

SORTING ALGORITHMS -MERGE TWO SORTED ARRAYS

AIM: Write a program to merge two sorted arrays.

SOURCE CODE:

```
#include<stdio.h>
void main()
{
  int a[50],b[50],c[100],m,n,i,j,k;
  printf("\n Enter the size of the first array:");
  scanf("%d",&m);

  printf("\n Enter the Elemnts of first array (sorted order): ");
   for(i=0;i<m;i++)
  {
     scanf("%d",&a[i]);
   }

  printf("\n Enter the size of the second array:");
  scanf("%d",&n);

  printf("\n Enter the Elemnts of second  array (sorted order): ");
  for(i=0;i<n;i++)
   {
     scanf("%d",&b[i]);
   }
    i=0;
    j=0;
    k=0;
```

```
  for(i=0;i<m+n;i++)
 {
       printf("%d ",c[i]);
   }
}
```

RESULT:

Program is executed successfully and output is obtained.

```
   printf("\n Sorted Array = " );
   for(i=0;i<n;i++)
      {
         printf("%d ",array[i]);
      }
}
```

RESULT:

Program is executed successfully and output is obtained.

```
            case 2:  pop();
                     break;

            case 3:  peek();
                     break;

            case 4:  display();
                     break;

            case 5:  printf("\nExiting program.");
                     return 0;

            default: printf("\nInvalid Choice!");

        }
    }
    return 0;
}
```

RESULT:

Program runs successfully and output is obtained.

```
    i = front;
    do {
        if (queue[i] == key)
        {
            printf("\nElement %d found at position %d.\n", key, i);
            found = 1;
            break;
        }
        i = (i + 1) % MAX;
    } while (i != (rear + 1) % MAX);
    if ( !found)
    {
        printf("\nElement %d not found in the queue.\n", key);
    }
}

int main()
{
    int choice;
    printf("\n CIRCULAR QUEUE USING ARRAY \n");

    do{
        printf("\n1. ENQUEUE \n2. DEQUEUE \n3. DISPLAY \n4. SEARCH \n5. EXIT \n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice)
        {

            case 1: enqueue();
                    break;

            case 2: dequeue();
                    break;

            case 3: display();
                    break;

            case 4: search();
                    break;

            case 5: printf("\n Exiting.... \n");
                    return 0;

            default: printf("\n INVALID CHOICE \n");
                     break;
        }

    } while (choice != 5);

    return 0;
}
```

RESULT :

Program is executed successfully and output is obtained.

```
                break;
        case 2: insertLast();
                break;
        case 3: insertLocation();
                break;
        case 4: deleteFirst();
                break;
        case 5: deleteLast();
                break;
        case 6: deleteLocation();
                break;
        case 7: search();
                break;
        case 8: display();
                break;
        case 9: printf("\n Exit \n");
                exit(0);

        default: printf("\n INVALID CHOICE \n");

      }

  }while(choice != 9);

}
```

RESULT:

Program run successfully and output is obtained.

RESULT:

Program run successfully and output is obtained.

```
      case 2:  dequeue();
               break;
      case 3:  display();
               break;
      case 4:  search();
               break;
      case 5:  printf("\n EXIT \n");
               break;

     default:  printf("Enter a valid Choice");
               break;
    }

 }while(choice != 5);
}
```

RESULT:

Program run successfully and output is obtained.

DATE: 08-11-2024

PROGRAM NO: 10

DOUBLY LINKED LIST

AIM: Write a program to implement doubly linked list..

SOURCE CODE:

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


struct node
{
  int data;
  struct node *Llink;
  struct node *Rlink;
};

  struct node *head = NULL;


void insertFirst()
{

        struct node *newnode;
        newnode=(struct node*)malloc(sizeof(struct node));

        if(newnode == NULL)
        {
              printf("\n No Space available");
              return;
        }

        newnode->Llink = NULL;
        newnode->Rlink = NULL;

        printf("\n Enter the element to insert");
        scanf("%d",&newnode->data);

        if(head == NULL)
        {
              head=newnode;
        }
        else
        {
              newnode->Rlink = head;
              head->Llink = newnode;
              head = newnode;
         }

        printf("\n %d inserted sucessfully",newnode->data);

}
```

```
                case 3: insertLocation();
                        break;

                case 4: deleteFirst();
                        break;

                case 5: deleteLast();
                        break;

                case 6: deleteLocation();
                        break;

                case 7: search();
                        break;

                case 8: display();
                        break;

                 case 9: printf("\nEXIT\n");
                        exit(0);

                default: printf("enter valid choice");
                        break;

            }

        } while(choice != 9);

}
```

RESULT :

Program is executed successfully and output is obtained.

DATE: 21-11-2024

PROGRAM NO: 11

IMPLEMENTATION OF SET

AIM: Given a U={1,2,3,4,5}, A={1,4,5} and B={2,3,4}. Write a C program to find A union B, A intersection B, A-B, B-A in bit vector.

SOURCE CODE:

```
#include<stdio.h>
void main()
{
int i;
  int U[5]={1,2,3,4,5};
  int A[5]={1,0,0,1,1};
  int B[5]={0,1,1,1,0};
  int uni[5],ints[5],diffB[5],diffA[5],compA[5],compB[5];

  // Display Universal Set
  printf("\n UNIVERSAL SET IS {");
  for(i=0;i<5;i++){
      printf("%d ",U[i]);
  }
  printf("} \n");

  //Display Set A
  printf("\n SET A {");
  for(i=0;i<5;i++){
       if(A[i]==1){
           printf("%d ",U[i]);
       }
  }
  printf("} \n");
```

```
//Display Set B
printf("\n SET B {");
for(i=0;i<5;i++){
    if(B[i]==1){
        printf("%d ",U[i]);
     }
}
printf("} \n");

//Union of A and B
printf("Union of A and B in bit representation is=");
for(i=0;i<5;i++){
      uni[i]=A[i]|B[i];
      printf("%d",uni[i]);
}
printf("\n UNION {");
for(i=0;i<5;i++){
      if(uni[i]==1){
          printf("%d ",U[i]);
       }
}
printf("} \n");

//Intersection of A and B
printf("Intersection of A and B in bit representation is:");
for(i=0;i<5;i++){
      ints[i]=A[i]&B[i];
      printf("%d",ints[i]);
}
```

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


 return 0;
}
```

RESULT:

Program runs successfully and output is obtained.

```
        Parent[rootX]=rootY;
    }
}

  int Connected(int x,int y)
  {
        return find(x)==find(y);
  }

int main()
{
    int n=10;
    initialize(n);
    union_set(1,2);
    union_set(3,4);
    union_set(2,3);
    printf("Are 1 and 4 connected ?%s \n",Connected(1,4)? "Yes":"No");
    printf("Are 1 and 5 connected ?%s \n",Connected(1,5)? "Yes":"No");
    return 0;
 }
```

RESULT:

Program runs successfully and output is obtained.

```
    while (front < rear) {
        int current = queue[front++];
        printf("%d ", current);
        for (int i = 0; i < n; i++) {
            if (graph[current][i] == 1 && !visited[i]) {
                visited[i] = 1;
                queue[rear++] = i;
            }
        }
    }
    printf("\n");
}
```

RESULT:

Program runs successfully and output is obtained.

```
void DFS(int i)
{
   int j;
   printf("%d ",i);
   visited[i] = 1;

   for(j=0;j<n;j++) {
      if( !visited[j]  &&  G[i][j] == 1) {
          DFS(j);
      }
   }
}
```

RESULT:

Program runs successfully and output is obtained.

```
int main()
{
   int n, edges, u, v;

   printf("Enter the number of vertices: ");
   scanf("%d", &n);


   for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
         adj[i][j] = 0;
      }
      visited[i] = 0;
   }

   printf("Enter the number of edges: ");
   scanf("%d", &edges);

   printf("Enter the edges (u v) (0-based index):\n");
   for (int i = 0; i < edges; i++) {
      scanf("%d %d", &u, &v);
      addEdge(u, v);

   }

   topologicalSort(n);

   return 0;
}
```

RESULT :

Program runs successfully and output is obtained

RESULT :

Program run successfully and output is obtained.

```
for (int i = 0; i < E; i++)
{
    int u, v, w;
    scanf("%d %d %d", &u, &v, &w);
    graph[u][v] = w;
    graph[v][u] = w;
}
primsMST(graph, V);
return 0;
}
```

RESULT :

Program run successfully and output is obtained