

Practical 1

```
#include <iostream>
#include <cstdio>
#include <cstring>
```

```
int main() {
    const char* code[9][4] = {
        {"PROG1", "START", "", ""},
        {"", "USING", "*", "07"},
        {"DATA1", "DC", "100", ""},
        {"DATA2", "DC", "200", ""},
        {"BUF1", "DS", "4F", ""},
        {"BUF2", "DS", "3F", ""},
        {"CYCLE", "A", "", ""},
        {"ENDP", "END", "", ""},
        {"", "", "", ""}
    };
};
```

```
char av[2], avail[15] = {'N','N','N','N','N','N','N','N','N','N','N','N','N','N','N'};
int i, j, k, count[3], lc[9] = {0}, loc = 0;
```

```
std::cout << "-----" << std::endl;
std::cout << "LABEL\t\tOPCODE\t\tOPERAND1\t\tOPERAND2\n";
std::cout << "-----" << std::endl;
```

```
for (i = 0; i <= 8; i++) {
    for (j = 0; j <= 3; j++) {
        std::cout << code[i][j] << "\t\t";
    }
    std::cout << std::endl;
}
```

```
std::cin.get();
```

```
std::cout << "-----\nVALUES FOR LC:\n\n";
for (j = 0; j <= 8; j++) {
    if ((strcmp(code[j][1], "START") != 0) &&
        (strcmp(code[j][1], "USING") != 0) &&
        (strcmp(code[j][1], "A") != 0)) {
        lc[j] = lc[j - 1] + 4;
    }
    std::cout << lc[j] << "\t";
}
```

```
}
```

```
std::cout << "\n\nSYMBOL TABLE:\n";
std::cout << "-----\n";
std::cout << "SYMBOL\t\tVALUE\t\tLENGTH\t\tR/A\n";
std::cout << "-----\n";
```

```
for (i = 0; i < 9; i++) {
    if (strcmp(code[i][1], "START") == 0) {
        std::cout << code[i][0] << "\t\t" << loc << "\t\t4\t\tR\n";
    } else if (strcmp(code[i][0], "") != 0) {
        std::cout << code[i][0] << "\t\t" << loc << "\t\t4\t\tR\n";
        loc += 4;
    } else if (strcmp(code[i][1], "USING") == 0) {
        // Do nothing
    } else {
        loc += 4;
    }
}
```

```
std::cout << "-----\n";
```

```
std::cout << "\n\nBASE TABLE:\n";
std::cout << "-----\n";
std::cout << "REG NO\t\tAVAILABILITY\t\tCONTENTS OF BASE TABLE\n";
std::cout << "-----\n";
```

```
for (j = 0; j <= 8; j++) {
    if (strcmp(code[j][1], "USING") == 0) {
        strcpy(av, code[j][3]);
    }
}
```

```
count[0] = (int)av[0] - 48;
count[1] = (int)av[1] - 48;
count[2] = count[0] * 10 + count[1];
avail[count[2] - 1] = 'Y';
```

```
for (k = 0; k < 16; k++) {
    std::cout << k << "\t\t" << ((k == (count[2] - 1)) ? 'Y' : 'N') << "\n";
}
```

```
std::cout << "-----\n";
std::cout << "Continue..??\n";
```


BASE TABLE:

REG NO	AVAILABILITY	CONTENTS OF BASE TABLE
0	N	
1	N	
2	N	
3	N	
4	N	
5	N	
6	Y	
7	N	
8	N	
9	N	
10	N	
11	N	
12	N	
13	N	
14	N	
15	N	

Continue..??

y

PASS2 TABLE:

LABEL	OPCODE	OPERAND1	OPERAND2
PROG1	START USING	*	07
DATA1	DC	100	
DATA2	DC	200	
BUF1	DS	4F	

Continue..??

y

PASS2 TABLE:

LABEL	OPCODE	OPERAND1	OPERAND2
PROG1	START USING	*	07
DATA1	DC	100	
DATA2	DC	200	
BUF1	DS	4F	
BUF2	DS	3F	
CYCLE	A		
ENDP	END		

user@user-OptiPlex-3050:~/Documents/SPOS\$

Practical 2

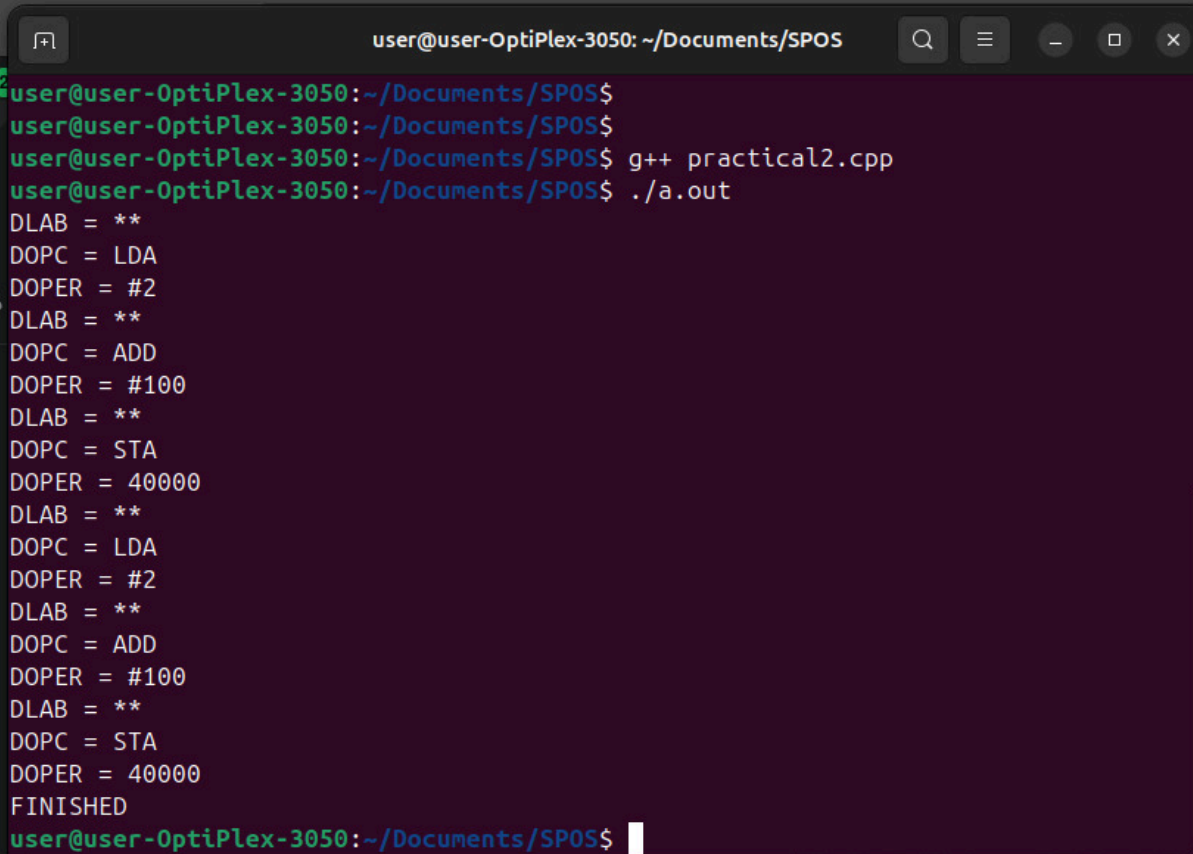
```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <cstdlib>
class deftab {
public:
    char lab[10];
    char opc[10];
    char oper[10];
} d[50];
int main() {
    char label[10], opcode[10], operand[10];
    char macroname[10];
    int i = 0, lines = 0;
    FILE *f1, *f2, *f3;
    f1 = fopen("macin.dat", "r");
    if (!f1) {
        std::cerr << "Error: cannot open macin.dat\n";
        return 1;
    }
    f2 = fopen("macout.dat", "w");
    f3 = fopen("deftab.dat", "w");
    if (!f2 || !f3) {
        std::cerr << "Error: cannot create output files\n";
        return 1;
    }
    if (fscanf(f1, "%s%s%s", label, opcode, operand) == EOF) {
        std::cerr << "Error: input file empty\n";
        return 1;
    }
    while (strcmp(opcode, "END") != 0) {
        if (strcmp(opcode, "MACRO") == 0) {
            strcpy(macroname, label);
            fscanf(f1, "%s%s%s", label, opcode, operand);
            lines = 0;
            while (strcmp(opcode, "MEND") != 0) {
                fprintf(f3, "%s\t%s\t%s\n", label, opcode, operand);
                strcpy(d[lines].lab, label);
                strcpy(d[lines].opc, opcode);
                strcpy(d[lines].oper, operand);
                fscanf(f1, "%s%s%s", label, opcode, operand);
                lines++;
            }
        }
        else {
            strcpy(d[i].lab, label);
            strcpy(d[i].opc, opcode);
            strcpy(d[i].oper, operand);
            i++;
        }
    }
}
```

```

    }
}
else if (strcmp(opcode, macroname) == 0) {
    for (i = 0; i < lines; i++) {
        fprintf(f2, "%s\t%s\t%s\n", d[i].lab, d[i].opc, d[i].oper);
        printf("DLAB = %s \nDOPC = %s \nDOPER = %s \n", d[i].lab, d[i].opc, d[i].oper);
    }
}
else {
    fprintf(f2, "%s\t%s\t%s\n", label, opcode, operand);
}
if (fscanf(f1, "%s%s%s", label, opcode, operand) == EOF) break;
}
fprintf(f2, "%s\t%s\t%s\n", label, opcode, operand);
fclose(f1);
fclose(f2);
fclose(f3);
printf("FINISHED\n");
return 0;
}

```

Output



```

user@user-OptiPlex-3050: ~/Documents/SPOS
user@user-OptiPlex-3050:~/Documents/SPOS$
user@user-OptiPlex-3050:~/Documents/SPOS$ g++ practical2.cpp
user@user-OptiPlex-3050:~/Documents/SPOS$ ./a.out
DLAB = **
DOPC = LDA
DOPER = #2
DLAB = **
DOPC = ADD
DOPER = #100
DLAB = **
DOPC = STA
DOPER = 40000
DLAB = **
DOPC = LDA
DOPER = #2
DLAB = **
DOPC = ADD
DOPER = #100
DLAB = **
DOPC = STA
DOPER = 40000
FINISHED
user@user-OptiPlex-3050:~/Documents/SPOS$

```

Practical 3

```
#include <iostream>
#include <iomanip>
#include <queue>
#include <algorithm>
#include <string.h>
using namespace std;

struct process {
    int pid, arrival_time, burst_time;
    int start_time, completion_time, turnaround_time;
    int waiting_time, response_time;
};

bool compare1(process p1, process p2) { return p1.arrival_time < p2.arrival_time; }
bool compare2(process p1, process p2) { return p1.pid < p2.pid; }

int main() {
    int n,tq;
    process p[100];
    int burst_remaining[100], mark[100];
    float avg_tat, avg_wt, avg_rt, cpu_util, throughput;
    int total_tat=0,total_wt=0,total_rt=0,total_idle=0;

    cout << fixed << setprecision(2);
    cout << "Enter number of processes: "; cin >> n;
    cout << "Enter time quantum: "; cin >> tq;

    for(int i=0;i<n;i++) {
        cout << "Enter arrival time of process " << i+1 << ": ";
        cin >> p[i].arrival_time;
        cout << "Enter burst time of process " << i+1 << ": ";
        cin >> p[i].burst_time;
        burst_remaining[i]=p[i].burst_time;
        p[i].pid=i+1;
    }

    sort(p,p+n,compare1);
    queue<int> q;
    int current_time=0, completed=0;
    memset(mark,0,sizeof(mark));
    q.push(0); mark[0]=1;
```

```

while(completed!=n) {
    int idx=q.front(); q.pop();

    if(burst_remaining[idx]==p[idx].burst_time) {
        p[idx].start_time=max(current_time,p[idx].arrival_time);
        total_idle+=p[idx].start_time-current_time;
        current_time=p[idx].start_time;
    }

    if(burst_remaining[idx]-tq>0) {
        burst_remaining[idx]-=tq;
        current_time+=tq;
    } else {
        current_time+=burst_remaining[idx];
        burst_remaining[idx]=0;
        completed++;

        p[idx].completion_time=current_time;
        p[idx].turnaround_time=p[idx].completion_time-p[idx].arrival_time;
        p[idx].waiting_time=p[idx].turnaround_time-p[idx].burst_time;
        p[idx].response_time=p[idx].start_time-p[idx].arrival_time;

        total_tat+=p[idx].turnaround_time;
        total_wt +=p[idx].waiting_time;
        total_rt +=p[idx].response_time;
    }

    for(int i=1;i<n;i++) {
        if(burst_remaining[i]>0 && p[i].arrival_time<=current_time && mark[i]==0) {
            q.push(i); mark[i]=1;
        }
    }
    if(burst_remaining[idx]>0) q.push(idx);
    if(q.empty()) {
        for(int i=1;i<n;i++) {
            if(burst_remaining[i]>0) { q.push(i); mark[i]=1; break; }
        }
    }
}

avg_tat=(float)total_tat/n;
avg_wt=(float)total_wt/n;
avg_rt=(float)total_rt/n;
cpu_util=((p[n-1].completion_time-total_idle)/(float)p[n-1].completion_time)*100;

```

```

throughput=(float)n/(p[n-1].completion_time-p[0].arrival_time);

sort(p,p+n,compare2);

cout << "\n#P\tAT\tBT\tST\tCT\tTAT\tWT\tRT\n";
for(int i=0;i<n;i++) {

cout<<p[i].pid<<"\t"<<p[i].arrival_time<<"\t"<<p[i].burst_time<<"\t"<<p[i].start_time<<"\t"
    <<p[i].completion_time<<"\t"<<p[i].turnaround_time<<"\t"<<p[i].waiting_time<<"\t"
    <<p[i].response_time<<"\n";
}
cout << "Average TAT = " << avg_tat << "\n";
cout << "Average WT = " << avg_wt << "\n";
cout << "Average RT = " << avg_rt << "\n";
cout << "CPU Utilization = " << cpu_util << "%\n";
cout << "Throughput = " << throughput << " process/unit time\n";
}

```

Output

```

user@user-OptiPlex-3050: ~/Documents/SPOS
user@user-OptiPlex-3050:~/Documents/SPOS$ g++ roundrobin.cpp
user@user-OptiPlex-3050:~/Documents/SPOS$ ./a.out
Enter number of processes: 4
Enter time quantum: 10
Enter arrival time of process 1: 2
Enter burst time of process 1: 5
Enter arrival time of process 2: 1
Enter burst time of process 2: 7
Enter arrival time of process 3: 4
Enter burst time of process 3: 3
Enter arrival time of process 4: 4
Enter burst time of process 4: 10

#P      AT      BT      ST      CT      TAT      WT      RT
1        2        5        0       12       10        5      -2
2        1        7        4        5        4       -3        3
3        4        3       12       15       11        8        8
4        4       10       15       25       21       11       11
Average TAT = 11.50
Average WT  = 5.25
Average RT  = 5.00
CPU Utilization = 100.00%
Throughput = 0.17 process/unit time
user@user-OptiPlex-3050:~/Documents/SPOS$

```


Practical 4

```
#include <iostream>

int main() {
    int ReferenceString[10], PageFaults = 0, s, pages, frames;

    std::cout << std::endl << "Enter the number of Pages:\t";
    std::cin >> pages;

    std::cout << std::endl << "Enter reference string values: " << std::endl;

    for (int i = 0; i < pages; i++) {
        std::cout << "value no. " << i + 1 << ":\t";
        std::cin >> ReferenceString[i];
    }

    std::cout << std::endl << "What are the total number of frames:\t";
    std::cin >> frames;

    int temp[frames];

    for (int i = 0; i < frames; i++) {
        temp[i] = -1;
    }

    for (int m = 0; m < pages; m++) {
        s = 0;
        for (int n = 0; n < frames; n++) {
            if (ReferenceString[m] == temp[n]) {
                s++;
                PageFaults--;
            }
        }
        PageFaults++;
        if ((PageFaults <= frames) && (s == 0)) {
            temp[m] = ReferenceString[m];
        }
        else if (s == 0) {
            temp[(PageFaults - 1) % frames] = ReferenceString[m];
        }

        std::cout << std::endl;
        for (int n = 0; n < frames; n++) {
```

```
std::cout << temp[n] << "\t";
    }
}

std::cout << std::endl << "Total Page Faults:\t" << PageFaults << std::endl;
return 0;
}
```

Output

[illegible]