Write a C Program to implement priority scheduling.

Following is the example of non preemptive scheduling with arrival time zero.

Step 1: Start the Program.  
Step 2: Input the number of processes.  
Step 3: Input the burst time and priority for each process.  
Step 4: Sort the element on the basis of priority.  
Step 5: Print order of execution of their process with their time stamp (wait time and turnaround time).  
Step 6: End the Program.

…………………….

1. */\**
2. *\* C program to implement priority scheduling*
3. *\*/*
5. #include <stdio.h>
7. *//Function to swap two variables*
8. void swap(int \*a,int \*b)
9. {
10. int temp=\*a;
11. \*a=\*b;
12. \*b=temp;
13. }
14. int main()
15. {
16. int n;
17. printf("Enter Number of Processes: ");
18. scanf("%d",&n);
20. *// b is array for burst time, p for priority and index for process id*
21. int b[n],p[n],index[n];
22. for(int i=0;i<n;i++)
23. {
24. printf("Enter Burst Time and Priority Value for Process %d: ",i+1);
25. scanf("%d %d",&b[i],&p[i]);
26. index[i]=i+1;
27. }
28. for(int i=0;i<n;i++)
29. {
30. int a=p[i],m=i;
32. *//Finding out highest priority element and placing it at its desired position*
33. for(int j=i;j<n;j++)
34. {
35. if(p[j] > a)
36. {
37. a=p[j];
38. m=j;
39. }
40. }
42. *//Swapping processes*
43. swap(&p[i], &p[m]);
44. swap(&b[i], &b[m]);
45. swap(&index[i],&index[m]);
46. }
48. *// T stores the starting time of process*
49. int t=0;
51. *//Printing scheduled process*
52. printf("Order of process Execution is**\n**");
53. for(int i=0;i<n;i++)
54. {
55. printf("P%d is executed from %d to %d**\n**",index[i],t,t+b[i]);
56. t+=b[i];
57. }
58. printf("**\n**");
59. printf("Process Id Burst Time Wait Time TurnAround Time**\n**");
60. int wait\_time=0;
61. for(int i=0;i<n;i++)
62. {
63. printf("P%d %d %d %d**\n**",index[i],b[i],wait\_time,wait\_time + b[i]);
64. wait\_time += b[i];
65. }
66. return 0;
67. }

……………………….

**Program Explanation**

1. First, enter the total number of processes and store it in variable **n**.  
2. After that, provide the burst time and priority and store it in variable **b** and **p**.  
3. Finding out highest priority element and placing it at its desired position.  
4. Sort the processes on the basis of the priority.  
5. After that print the processed with their time stamp (starting time and ending time). Variable **T** stores the starting time of process.  
6. In the end, print the **waiting time** and **turnaround time** for each process. Waiting time is the time spent in the ready queue, while turnaround time is the total time taken by process **(burst time + waiting time)**.

…………………………………………………………………………………………………………………………………………………………

**Round Robin Scheduling Algorithm:**  
Step 1: Start the Program.  
Step 2: Input the number of processes.  
Step 3: Input the burst time and arrival time of each process and the limit of the time slot.  
Step 4: Push all processes into the ready queue according to their arrival time. Then execute each process upto time slot and push left over process in queue again for execution.  
Step 5: After a process is completely executed, print its turn around time and waiting time.

C program to implement Round Robin scheduling.

1. */\**
2. *\* Round Robin Scheduling Program in C*
3. *\*/*
5. #include<stdio.h>
7. int main()
8. {
9. *//Input no of processed*
10. int n;
11. printf("Enter Total Number of Processes:");
12. scanf("%d", &n);
13. int wait\_time = 0, ta\_time = 0, arr\_time[n], burst\_time[n], temp\_burst\_time[n];
14. int x = n;
16. *//Input details of processes*
17. for(int i = 0; i < n; i++)
18. {
19. printf("Enter Details of Process %d **\n**", i + 1);
20. printf("Arrival Time: ");
21. scanf("%d", &arr\_time[i]);
22. printf("Burst Time: ");
23. scanf("%d", &burst\_time[i]);
24. temp\_burst\_time[i] = burst\_time[i];
25. }
27. *//Input time slot*
28. int time\_slot;
29. printf("Enter Time Slot:");
30. scanf("%d", &time\_slot);
32. *//Total indicates total time*
33. *//counter indicates which process is executed*
34. int total = 0, counter = 0,i;
35. printf("Process ID Burst Time Turnaround Time Waiting Time**\n**");
36. for(total=0, i = 0; x!=0; )
37. {
38. *// define the conditions*
39. if(temp\_burst\_time[i] <= time\_slot && temp\_burst\_time[i] > 0)
40. {
41. total = total + temp\_burst\_time[i];
42. temp\_burst\_time[i] = 0;
43. counter=1;
44. }
45. else if(temp\_burst\_time[i] > 0)
46. {
47. temp\_burst\_time[i] = temp\_burst\_time[i] - time\_slot;
48. total += time\_slot;
49. }
50. if(temp\_burst\_time[i]==0 && counter==1)
51. {
52. x--; *//decrement the process no.*
53. printf("**\n**Process No %d **\t\t** %d**\t\t\t\t** %d**\t\t\t** %d", i+1, burst\_time[i],
54. total-arr\_time[i], total-arr\_time[i]-burst\_time[i]);
55. wait\_time = wait\_time+total-arr\_time[i]-burst\_time[i];
56. ta\_time += total -arr\_time[i];
57. counter =0;
58. }
59. if(i==n-1)
60. {
61. i=0;
62. }
63. else if(arr\_time[i+1]<=total)
64. {
65. i++;
66. }
67. else
68. {
69. i=0;
70. }
71. }
72. float average\_wait\_time = wait\_time \* 1.0 / n;
73. float average\_turnaround\_time = ta\_time \* 1.0 / n;
74. printf("**\n**Average Waiting Time:%f", average\_wait\_time);
75. printf("**\n**Avg Turnaround Time:%f", average\_turnaround\_time);
76. return 0;
77. }

**Program Explanation**

1. Ask the user for number of processes **n**.  
2. After that, ask the user for the arrival time and burst time of each process. Also input the time quantum.  
3. In the loop, if time slot is greater than left burst time, execute process and find burst time.  
4. Else if burst time is greater than time slot, execute it up to time slot and again push into the queue.  
5. when the execution is completed, print the process information such as turnaround time and waiting time.

…………………………………………………………………………………………………………………………………………….

Implementation of Pre-emptive SJF Scheduling in C language

The algorithm of Shortest Job First is as follows:

First of all, all the processes are sorted according to the arrival time.

After that, the process having minimum arrival and burst time is selected.

Then that process gets executed and all the process that arrives waits in a queue while the first process is in the execution state, and after completion again the process with minimum arrival and burst time is selected for the next execution.

#include<stdio.h>

int main()

{

int burst\_time[20],p[20],waiting\_time[20],tat[20],i,j,n,total=0,pos,temp;

float avg\_waiting\_time,avg\_tat;

printf("please enter number of process: ");

scanf("%d",&n);

printf("\n enter the Burst Time:\n");

for(i=0;i<n;i++)

{

printf("p%d:",i+1);

scanf("%d",&burst\_time[i]);

p[i]=i+1;

}

// from here, burst times sorted

for(i=0;i<n;i++)

{

pos=i;

for(j=i+1;j<n;j++)

{

if(burst\_time[j]<burst\_time[pos])

pos=j;

}

temp=burst\_time[i];

burst\_time[i]=burst\_time[pos];

burst\_time[pos]=temp;

temp=p[i];

p[i]=p[pos];

p[pos]=temp;

}

waiting\_time[0]=0;

for(i=1;i<n;i++)

{

waiting\_time[i]=0;

for(j=0;j<i;j++)

waiting\_time[i]+=burst\_time[j];

total+=waiting\_time[i];

}

avg\_waiting\_time=(float)total/n;

total=0;

printf("\nProcess\t Burst Time \tWaiting Time\tTurnaround Time");

for(i=0;i<n;i++)

{

tat[i]=burst\_time[i]+waiting\_time[i];

total+=tat[i];

printf("\np%d\t\t %d\t\t %d\t\t\t%d",p[i],burst\_time[i],waiting\_time[i],tat[i]);

}

avg\_tat=(float)total/n;

printf("\n\n the average Waiting Time=%f",avg\_waiting\_time);

printf("\n the average Turnaround Time=%f\n",avg\_tat);

}

……………………………………

Source from where code is taken

<https://www.sanfoundry.com/c-program-round-robin-scheduling/>

<https://www.sanfoundry.com/c-program-priority-scheduling/>

<https://www.scaler.com/topics/sjf-scheduling-in-c/> (preemptive)