**Lab Exercise 1.4– Simple switch-case in CUDA Program**

**Objective**

* Learn how to use a switch-case statement inside a CUDA kernel.
* Understand how threads can select different actions based on a value.

**Program:**

#include <iostream>

#include <cuda\_runtime.h>

\_\_global\_\_ void gradeCategory(int \*marks, char \*grades, int N) {

int tid = blockIdx.x \* blockDim.x + threadIdx.x; // Global thread ID

if (tid < N) {

int category = marks[tid] / 10; // Group by tens

switch (category) {

case 10:

case 9:

grades[tid] = 'A';

break;

case 8:

grades[tid] = 'B';

break;

case 7:

grades[tid] = 'C';

break;

default:

grades[tid] = 'F';

}

}

}

int main() {

const int N = 5;

int h\_marks[N] = {95, 82, 76, 65, 45}; // Input marks

char h\_grades[N]; // Output grades

int \*d\_marks;

char \*d\_grades;

cudaMalloc(&d\_marks, N \* sizeof(int));

cudaMalloc(&d\_grades, N \* sizeof(char));

cudaMemcpy(d\_marks, h\_marks, N \* sizeof(int), cudaMemcpyHostToDevice);

gradeCategory<<<1, N>>>(d\_marks, d\_grades, N);

cudaMemcpy(h\_grades, d\_grades, N \* sizeof(char), cudaMemcpyDeviceToHost);

std::cout << "Marks and Grades:\n";

for (int i = 0; i < N; i++) {

std::cout << h\_marks[i] << ": " << h\_grades[i] << "\n";

}

cudaFree(d\_marks);

cudaFree(d\_grades);

return 0;

}

**Step-by-Step Execution and Explanation**

**Step 1: Include Headers**

#include <iostream>

#include <cuda\_runtime.h>

* iostream: For output (std::cout).
* cuda\_runtime.h: Provides CUDA runtime API functions.

**Step 2: Define the CUDA Kernel**

\_\_global\_\_ void gradeCategory(int \*marks, char \*grades, int N)

* \_\_global\_\_: Marks this function as a **kernel** to be run on the **GPU**.
* Purpose: Based on marks, assign a **grade** using switch-case.

**Step 3: Calculate Global Thread ID**

int tid = blockIdx.x \* blockDim.x + threadIdx.x;

* tid uniquely identifies the thread's element in the array.

**Step 4: Apply switch-case inside Kernel**

int category = marks[tid] / 10;

* Group marks into categories:
  + 90–100 → category 9 or 10
  + 80–89 → category 8
  + 70–79 → category 7
  + Others → default

switch (category) {

case 10:

case 9:

grades[tid] = 'A';

break;

case 8:

grades[tid] = 'B';

break;

case 7:

grades[tid] = 'C';

break;

default:

grades[tid] = 'F';

}

* Depending on the category, assign the corresponding grade to grades[tid].

**Step 5: Host Code: Main Function**

const int N = 5;

int h\_marks[N] = {95, 82, 76, 65, 45};

char h\_grades[N];

* h\_marks: Input marks.
* h\_grades: Array to store the grades.

**Step 6: Allocate Device Memory**

cudaMalloc(&d\_marks, N \* sizeof(int));

cudaMalloc(&d\_grades, N \* sizeof(char));

* Allocate memory for marks and grades on GPU.

**Step 7: Copy Data from Host to Device**

cudaMemcpy(d\_marks, h\_marks, N \* sizeof(int), cudaMemcpyHostToDevice);

* Copy marks from CPU to GPU.

**Step 8: Launch Kernel**

gradeCategory<<<1, N>>>(d\_marks, d\_grades, N);

* Launch 1 block with N=5 threads.

**Step 9: Copy Data from Device to Host**

cudaMemcpy(h\_grades, d\_grades, N \* sizeof(char), cudaMemcpyDeviceToHost);

* Copy grades from GPU back to CPU.

**Step 10: Print Results**

std::cout << "Marks and Grades:\n";

for (int i = 0; i < N; i++) {

std::cout << h\_marks[i] << ": " << h\_grades[i] << "\n";

}

* Display marks and their corresponding grades.

**Step 11: Free Device Memory**

cudaFree(d\_marks);

cudaFree(d\_grades);

* Free the GPU memory.

**Expected Output**

Marks and Grades:

95: A

82: B

76: C

65: F

45: F

**Explanation**:

* 95 → category 9 → grade A
* 82 → category 8 → grade B
* 76 → category 7 → grade C
* 65 → below 70 → grade F
* 45 → below 70 → grade F

**Summary**

* **switch-case** inside CUDA kernels helps **branch based on multiple conditions**.
* Each thread independently evaluates its switch-case and assigns a result.
* CUDA supports all basic C/C++ control flow structures inside kernels.