# VIRTUAL MEMORY SIMULATOR

## Generator.cpp

This cpp file generates the page reference string keepig in mind the locality of reference. We have taken inputs asking user to feed the probability of locality of reference and maximum pages and read probability. First of all we have randomly initialised a working set from the total number of pages and then we generate the next page reference number by checking if the random number given by "rand()%100" is less than the % locality of reference. If it is so then we ouput a page number from the working set otherwise output a page number which is not there in the working set. Similary we have generated whether the page is to be read or written.

## Paging.cpp

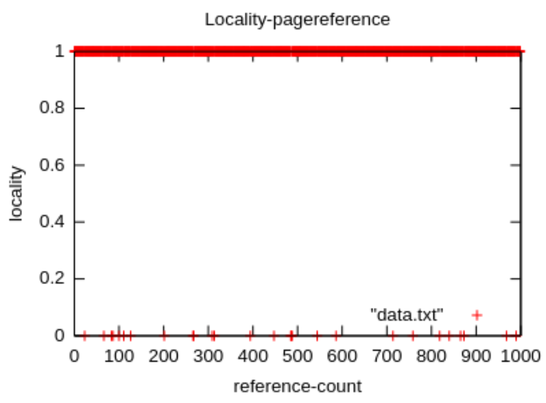
This cpp file outputs the actions to be taken against each page reference. We have two data structures one is page table which is stored as an array of integers and secondly a frame structure which stores the page number that the corresponding physical frame stores and the last time it was accesed.

#### Sequence of the whole process :

1. Given a page number to reference first of all we check wheather that page number is there in the page table or not and if it is there, is it's valid bit 1 or not which indicates wheather that page has been alloted a physical page frame or not.
2. If it the valid bit one the we just do the operation (read or write) on that page.
3. If the page has not been alloted a physical frame then we search for an unallocated physical frame. If there is one we just allocate that frame and update our current frame counter.
4. If there is no unallocated physical frame available (this is done by checking if current frame count < total number of frames). If so then we apply the replacement_policy as given by the user to select the victim page.
5. In case of FIFO replacement_policy we have maintained a pointer to the current victim page (the page which will be sacrificed if there is no physical frame left). We unmap this page set the valid bit to 0 for its corresponding entry in the page table and increment the vixtim pointer by one.
6. If the replacement_policy is random then we random select a page and then unmap it from physical frame.
7. If the replacement_policy is LRU then we select a page that has the lowest value of counter and unmap that page.
8. If the replacement_policy is NRU then we keep on searching for a page in the physical frame which has it's refernce bit 0 . If we are unable to find one then we sacrifice (last_use_ptr + 1)th page. All the refernce bits are set to 0 after every 1000 MEMORY accesses or 10th page fault.
9. If the replacement_policy is second_chance then we keep on searching for a page in the physical frame which has it's refernce bit 0 and mean while we change the reference bit of pages which have their reference bit set to 1 during the scan.

## Image file :

This the graph corresponding to refernce representing wheather the locality of refernce is preserved or not.
0 : Not preserved
1 : Preserved



**Submitted by: (Group No. 46)**

| | | |
|---|---|---|
| Shashi Ranjan Prakash | : | 15CS30028 |
| Shivam Goenka | : | 15CS30029 |