1. **INTRODUCTION**

Computers have become a powerful tool for the rapid and economical production of pictures. There is virtually no area in which graphical displays cannot be used to some advantage, and so it is not surprising to find the use of computer graphics so widespread. Although early applications in engineering and science had to rely on expensive and cumbersome equipment, advances in computer technology have made interactive computer graphics a practical tool. Today, we find computer graphics used routinely in such diverse areas as science, engineering, medicine, business, industry, government, art, entertainment, advertising, education, and training.

Computer graphics are graphics created using computers and, more generally, the representation and manipulation of image data by a computer. The development of computer graphics has made computers easier to interact with, and better for understanding and interpreting many types of data. Developments in computer graphics have had a profound impact on many types of media and have revolutionized animation, movies and the video game industry.

A major use of computer graphics is in design processes, particularly for engineering and architectural systems, but almost all products are now computer designed. Generally referred to as CAD, computer-aided design methods are now routinely used in the design of buildings, automobiles, aircraft, watercraft, spacecraft, computers, textiles, and many, many other products.

Here we have used “OpenGL” as the graphics software system to implement our mini project, “Singly Linked List”. Now let us have a quick look at OpenGL.

OpenGL is a library for doing computer graphics. By using it, we can create interactive applications which render high-quality color images composed of 3D geometric objects and images. OpenGL is window and operating system independent. As such, the part of our application which does rendering is platform independent.

However, in order for OpenGL to be able to render, it needs a window to draw into. Generally, this is controlled by the windowing system on whatever platform we are working on. As OpenGL is platform independent, we need some way to integrate OpenGL into each windowing system.

Every windowing system where OpenGL is supported has additional API calls for managing OpenGL windows, color maps and other features. These additional APIs are platform dependent. For the sake of simplicity, we are using an additional freeware library for simplifying interacting with windowing systems, GLUT.

GLUT, the OpenGL Utility Toolkit is a library to make writing OpenGL programs regardless of windowing systems much easier.

1.1 OpenGL architecture

OpenGL is a collection of several hundred functions providing access to all the features offered by your graphics hardware. Internally, it acts as a state machine--a collection of states that tells OpenGL what to do. Using the API, you can set various aspects of the state machine, including such things as the current color, lighting, blending, and so on. When rendering, everything drawn is affected by the current settings of the state machine. It's important to be aware of what the various states are, and the effect they have, because it's not uncommon to have unexpected results due to having one or more states set incorrectly.

At the core of OpenGL is the rendering pipeline, as shown in Figure 1.1. You don't need to understand everything that happens in the pipeline at this point, but you should at least be aware that what you see on the screen results from a series of steps. Fortunately, OpenGL handles most of these steps for you.

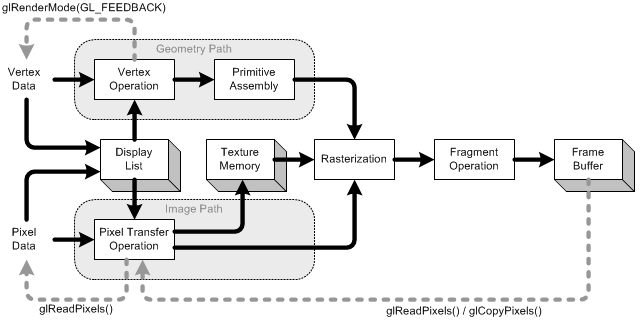


Fig 1.1 The OpenGL rendering pipeline.

Under Windows, OpenGL provides an alternative to using the Graphics Device Interface (GDI). GDI architects designed it to make the graphics hardware entirely invisible to Windows programmers. This provides layers of abstraction that help programmers avoid dealing with Device-specific issues. However, GDI is intended for use with applications and thus lacks the speed required for games. Figure 1.1 illustrates the OpenGL hierarchy under Windows.

1. **PROBLEM STATEMENT**
   1. **Aim:**

To implement singly Linked List using OpenGL.

* 1. **Singly Linked List overview:**

A linked list is a series of nodes in memory such that:

* There is a starting node.
* Each node contains a pointer that points to the next or child node.
* If a node does not have a child node then its pointer is set to NULL.
* Each node contains data, maybe a lot of it.
* The linked list also has functions that manage the list by performing additions and deletions of nodes.
  1. **Project Description:**

The Mini Project Singly linked list demonstrates the use of various OpenGL functions and its applications to model the requirements of an application programmer. It also illustrates the use of algorithmic functions to control the execution of the project that is presented.

It highlights the key features of the data structures and its high quality efficiency that is obtained on its usage in the application program. This project consists of Singly Linked List which is constructed by using different primitives available in OpenGL library and combining them together in a required manner.

* 1. **Scope**

This project can be used for teaching the undergraduate students since it is helpful in understanding the concept of Singly Linked List clearly.

1. **REQUIREMENTS**

3.1 Hardware requirements:

* Processor: Intel Pentium 3.0, 1.5 GHz
* RAM: 512 MB
* Monitor: 1024 \* 768 display resolution , High Colors 16-bit
* Hard Disk Space: 4.0 GB
* DVD drive
* Keyboard and Mouse

3.2 Software requirements:

* An MS-DOS based operating system like Windows 98 or Window 2000 or Window XP is the platform required to develop the 2D or 3D graphics application.
* A Visual C/C++ compiler like Microsoft Visual Studio 2005 is required for compiling the source code to make the executable file which can then be directly executed.
* Microsoft Office for documentation and presentation of the mini graphics project.
* OpenGL , a software interface for graphics hardware with built in graphics libraries like glut and glut32, and header files like glut.h.

**4. DESIGN AND IMPLEMENTATION**

4.1 Inbuilt functions:

* **glBegin()**

C Specification: void **glBegin** (Glenum mode);

Parameters: mode

Description: Specifies the primitive(s) that will be created from vertices present between glBegin and the subsequent glEnd. Ten symbolic constants are accepted.

* **glEnd()**

C Specification: void **glEnd()**;

Description: glBegin and glEnd delimit the vertices that define a primitive or a group of like primitives. glBegin accepts a single argument that specifies in which often ways the vertices are interpreted. Taking ‘n’ as an integer count starting at one, and ‘N’ as the total number of vertices specified, the interpretations are as follows:

**GL\_LINES:** Treats each pair of vertices as an independent line segment. Vertices 2n-1 and 2n define line n. N/2 lines are drawn.

e drawn.

**GL\_POLYGON:** Draws a single, convex polygon. Vertices 1 through N define this polygon.

* **glClearColor()**

C Specification: void **glClearColor**(GLClamp red, GLClamp green, GLClamp blue, GLClamp alpha);

Parameters: red, green, blue, alpha specifies the red, blue, green and alpha values used when the color buffers are cleared, the initial values are all 10.

Description: glClearColor specifies the red, green, blue, and alpha values used by glClear to clear the color buffers. Values specified by glClearColor are clamped to the range [0, 1].

* **glClear()**

C specification: void **glClear**(GLbitfield mask);

Parameters: mask bitwise OR of masks that indicate the buffers to be cleared. The four masks are GL\_COLOR\_BUFFER\_BIT, GL\_DEPTH\_BUFFER\_BIT, GL\_ACCUM\_BUFFER\_BIT, and GL\_STENCIL\_BUFFER\_BIT.

Description: glClear takes a single argument that is bitwise OR of several values indicating which buffer is to be cleared. The values are as follows: GL\_COLOR\_BUFFER\_BIT-indicates the buffers currently enabled for color writing.

GL\_DEPTH\_BUFFER\_BIT - indicates the depth buffer. GL\_ACCUM\_BUFFER\_BIT - indicates the accumulation buffer.

GL\_STENCIL\_BUFFER\_BIT - indicates the stencil bit.

* **glFlush()**

C Specification: void **glFlush()**;

Description: glFlush empties all the buffers causing all issued commands to be executed as quickly as they are accepted by the actual rendering engine.

* **glLoadIdentity()**

C specification: void **glLoadIdentity()**;

Description: glLoadIdentity replaces the current matrix with identity matrix.

* **glMatrixMode()**

C Specification: void **glMatrixMode**(Glenum mode);

Parameters: mode specifies which matrix stack is the target for subsequent matrix operations. Three values accepted are: GL\_MODELVIEW, GL\_PROJECTION, and GL\_TEXTURE.

Description: glMatrixMode sets the current matrix mode. mode can assume one of the following values: GL\_MODEL\_VIEW applies subsequent matrix operations to the modelview matrix stack GL\_PROJECTION applies subsequent matrix operations to the projection matrix stack.

* **glOrtho2D()**

C Specification: void **gluOrtho2D**(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top);

Description: gluOrtho2D sets up a two-dimensional orthographic viewing region. This is equivalent to calling glOrtho with near = -1 and far = 1.

* **glVertex()**

C Specification: void **glVertex**{234}{sifd}[v](TYPE co-ordinates);

Description: glVertex commands are used within glBegin/glEnd pairs to specify point, line and polygon vertices. It specifies a pointer to an array of two, three or four elements.

* **glViewport()**

C Specification: void **glViewport**(Glint x, Glint y, Glsizei width, Glsizei height);

Parameters: -x, y specify the lower left corner of the viewport rectangle, in pixels. -width, height specify the width and height of the viewport.

Description: glViewport specifies the affine transformation of x and from the normalized device coordinates to window coordinates.

* **glutBitmapCharacter()**

C Specification: void **glutBitmapCharacter**(void \*font, int character);

Description: glutBitmapCharacter renders the character in the named bitmap font.

* **glutCreateWindow()**

C Specification: int **glutCreateWindow**(char \*name);

Description: glutCreateWindow creates a top-level window. The name will be provided to the window’s name.

* **glutDisplayFunc()**

C Specification: void **glutDisplayFunc** (void (\*func)(void));

Description: glutDisplayFunc sets the display for the current window.

* **glutReshapeFunc()**

C specification: void **glutReshapeFunc**(void (\*func)(int width, int height));

Description: glutReshapeFunc sets the reshape callback for the current window. The reshape callback is triggered when a window is reshaped.

* **glutMainLoop()**

C Specification: void **glutMainLoop**(void);

Description: glutMainLoop enters the GLUT event processing loop. This routine should be called at most once in GLUT.

* **glCreateMenu()**

C specification: int **glutCreateMenu**(void (\*func)(int value));

Description: glutCreateMenu creates a new pop-up menu and returns a unique small integer identifier. The range of allocated identifiers starts at one. The menu identifier range is separate from the window identifier range. Implicitly, the current menu is set to the newly created menu. This menu identifier can be used when calling glutSetMenu.

* **glutAddMenu()**

C specification: void **glutAddMenuEntry**(char \*name, int value);

Description: glutAddMenuEntry adds a menu entry to the bottom of the current menu. The string name will be displayed for the newly added menu entry. If the menu entry is selected by the user, the menu's callback will be called passing value as the callback's parameter.

* **glRasterPosition()**

C specification: void **glRasterPos**{234}{sifd}(TYPE co-ordinates);

Description: The GL maintains a 3D position in window coordinates. This position, called the raster position, is used to position pixel and bitmap write operations. It is maintained with subpixel accuracy.

4.2 Code for user defined functions

* **display():**

To display the output on the screen.

void display()

{

int i=0,j=0,k=0,b[20];

NODE temp;

glClearColor(1.0,1.0,1.0,0.0);

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(1.0,0.0,0.6);

PrintString(200.0,450.0,"SINGLY LINKED LIST",1);

if(first==NULL)

{

printf("List is empty\n");

EmptyList();

return;

}

printf("Contents of singly linked list \n");

temp=first;

while(temp!=NULL)

{

for(k=0;k<count;k++)

{

GLfloat x1=50,x2=80,x3=110,y1=200,y2=225,y3=250,x4=95,y4=225,x5=160,y5=210,y6=240,x6=180;

if(first->link==NULL)

{

glColor3f(0.0,0.0,0.0);

glBegin(GL\_LINES);

glVertex2i(x2+k\*130,y3);

glVertex2i(x3+k\*130,y1);

glEnd();

}

if(temp->link==NULL)

{

glColor3f(1.0,0.0,0.0);

glBegin(GL\_POLYGON);

glVertex2i(x1+k\*130,y1);

glVertex2i(x1+k\*130,y3);

glVertex2i(x2+k\*130,y3);

glVertex2i(x2+k\*130,y1);

glEnd();

glColor3f(0.0,0.0,0.0);

glBegin(GL\_POLYGON);

glVertex2i(x2+k\*130,y1);

glVertex2i(x2+k\*130,y3);

glVertex2i(x3+k\*130,y3);

glVertex2i(x3+k\*130,y1);

glEnd();

glColor3f(1.0,1.0,1.0);

glBegin(GL\_LINES);

glVertex2i(x2+k\*130,y3);

glVertex2i(x3+k\*130,y1);

glEnd();

glBegin(GL\_LINES);

glVertex2i(x2+k\*130,y1);

glVertex2i(x3+k\*130,y3);

glEnd();

}

else

{

glColor3f(0.0f,0.0f,0.0f);

glBegin(GL\_LINES);

glVertex2i(x4+k\*130,y4);

glVertex2i(x6+k\*130,y4);

glEnd();

glColor3f(0.0f,0.0f,0.0f);

glBegin(GL\_LINES);

glVertex2i(x5+k\*130,y6);

glVertex2i(x6+k\*130,y4);

glVertex2i(x6+k\*130,y4);

glVertex2i(x5+k\*130,y5);

glEnd();

glColor3f(1.0,0.0,0.0);

glBegin(GL\_POLYGON);

glVertex2i(x1+k\*130,y1);

glVertex2i(x1+k\*130,y3);

glVertex2i(x2+k\*130,y3);

glVertex2i(x2+k\*130,y1);

glEnd();

glColor3f(0.0,0.0,0.0);

glBegin(GL\_POLYGON);

glVertex2i(x2+k\*130,y1);

glVertex2i(x2+k\*130,y3);

glVertex2i(x3+k\*130,y3);

glVertex2i(x3+k\*130,y1);

glEnd();

}

b[j]=temp->info;

printf("%d",temp->info);

temp=temp->link;

printf("\n");

ar[i].p=b[j]/10;

ar[i].q=b[j]%10;

ar[i].p=ar[i].p+48;

ar[i].q=ar[i].q+48;

glColor3f(1.0,1.0,0.0);

glRasterPos2f(60.0+k\*130,225.0);

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_18,ar[i].p);

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_18,ar[i].q);

PrintString1(50.0+k\*130,185,"INFO",1);

PrintString1(85.0+k\*130,185,"LINK",1);

i++;

j++;

glFlush();

}

}

}

* **GetNode ():**

To create a node.

NODE GetNode()

{

NODE x;

x=(NODE)malloc(sizeof(struct node));

if(x==NULL)

{

printf("insufficient memory \n");

exit(0);

}

return x;

}

* **EmptyList():**

To display the empty list.

void EmptyList()

{

glColor3f(1.0f,1.0f,1.0f);

glBegin(GL\_POLYGON);

glVertex2i(50,200);

glVertex2i(50,250);

glVertex2i(130,250);

glVertex2i(130,200);

glEnd();

glFlush();

}

* **PrintString() and PrintString1():**

To print a string on the screen.

void PrintString(float x,float y,char \*string,int col)

{

char \*c;

glColor3fv(color[col]);

glRasterPos2f(x,y);

for(c=string;\*c!='\0';c++)

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_18,\*c);

}

void PrintString1(float x,float y,char \*string,int col)

{

char \*c;

glColor3fv(color[col]);

glRasterPos2f(x,y);

for(c=string;\*c!='\0';c++)

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_12,\*c);

}

* **Title():**

To display the introduction page.

void Title()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(1.0,0.0,0.0);

PrintString(200.0,450.0,"SINGLY LINKED LIST",1);

PrintString(50.0,350.0,"NAME: SWAROOP D",3);

PrintString(50.0,300.0,"University Number: 1BG11CS110",3);

PrintString(50.0,100.0,"COLLEGE: BNMIT",3);

PrintString(150.0,50.0,"(Click right mouse button for options)",2);

glFlush();

}

* **delay(), delay1() and IntroDelay():**

Delays for the introduction page.

void delay()

{

int i,j;

for(i=0;i<5000;i++)

for(j=0;j<5000;j++);

}

void delay1()

{

int i,j;

for(i=0;i<50000;i++)

for(j=0;j<50000;j++);

}

void IntroDelay()

{

if(flag)

{

int d=10;

Title();

while(d>0)

{

delay();

d--;

}

flag=0;

}

glClear(GL\_COLOR\_BUFFER\_BIT);

}

* **InsertFront():**

To insert the node from the front.

NODE InsertFront(int item,NODE first)

{

NODE temp;

temp=GetNode();

temp->info=item;

temp->link=first;

return temp;

}

* **InsertRear():**

To insert the node from rear.

NODE InsertRear(int item,NODE first)

{

NODE temp;

NODE cur;

temp=GetNode();

temp->info=item;

temp->link=NULL;

if(first==NULL)

return temp;

cur=first;

while(cur->link!=NULL)

cur=cur->link;

cur->link=temp;

return first;

}

* **InsertSpecified():**

To insert a node after a particular (specified) node.

NODE InsertSpecified(int item,NODE first)

{

NODE next,cur,temp;

temp=GetNode();

temp->info=item;

if(first==NULL)

{

printf("list is empty.inserting node at front end...\n");

temp->link=NULL;

first=temp;

return first;

}

printf("Enter the key : ");

scanf("%d",&key);

cur=first;

while(cur!=NULL)

{

next=cur->link;

if(cur->info==key)

{

printf("key found\n");

cur->link=temp;

temp->link=next;

return first;

}

cur=cur->link;

}

if(cur==NULL)

{

printf("search unsuccessful\n");

count--;

}

return first;

}

* **DeleteFront():**

To delete the node in the front.

NODE DeleteFront(NODE first)

{

NODE temp;

if(first==NULL)

{

printf("list is empty cannot delete\n");

EmptyList();

count++;

return first;

}

temp=first;

temp=temp->link;

printf("item deleted=%d \n",first->info);

free(first);

return temp;

}

* **DeleteRear():**

To delete the node in the end.

NODE DeleteRear(NODE first)

{

NODE cur,prev;

if(first==NULL)

{

printf("list is empty cannot delete \n");

EmptyList();

count++;

return first;

}

if(first->link==NULL)

{

printf("item to be deleted is %d \n",first->info);

free(first);

return NULL;

}

prev=NULL;

cur=first;

while(cur->link!=NULL)

{

prev=cur;

cur=cur->link;

}

printf("item deleted is %d \n",cur->info);

free(cur);

prev->link=NULL;

return first;

}

* **DeleteSpecified():**

To delete a particular (specified) node.

NODE DeleteSpecified(NODE first)

{

NODE prev,cur;

if(first==NULL)

{

printf("list is empty\n");

EmptyList();

count++;

return NULL;

}

printf("enter the key");

scanf("%d",&key);

if(key==first->info)

{

cur=first;

first=first->link;

free(cur);

return first;

}

prev=NULL;

cur=first;

while(cur!=NULL)

{

if(key==cur->info)

break;

prev=cur;

cur=cur->link;

}

if(cur==NULL)

{

printf("search is unsuccessful\n");

count++;

return first;

}

prev->link=cur->link;

free(cur);

return first;

}

* **SLLmenu():**

Menu having the following choices:

* Insert front.
* Insert rear.
* Insert specified.
* Delete front.
* Delete rear.
* Delete specified.
* Exit.

void SLLmenu(int choice)

{

switch(choice)

{

case 1:

printf("enter the item to be inserted");

scanf("%d",&item);

first=InsertFront(item,first);

count++;

display();

break;

case 2:

printf("enter the item to be inserted");

scanf("%d",&item);

first=InsertRear(item,first);

count++;

display();

break;

case 3:

printf("enter the item to be inserted : ");

scanf("%d",&item);

count++;

display();

break;

case 4:

first=DeleteFront(first);

count--;

display();

break;

case 5:

first=DeleteRear(first);

count--;

display();

break;

case 6:

first=DeleteSpecified(first);

count--;

display();

break;

case 7:

exit(0);

}

}

1. **Results & Snapshots**

Singly linked list is successfully implemented and has been made interactive.

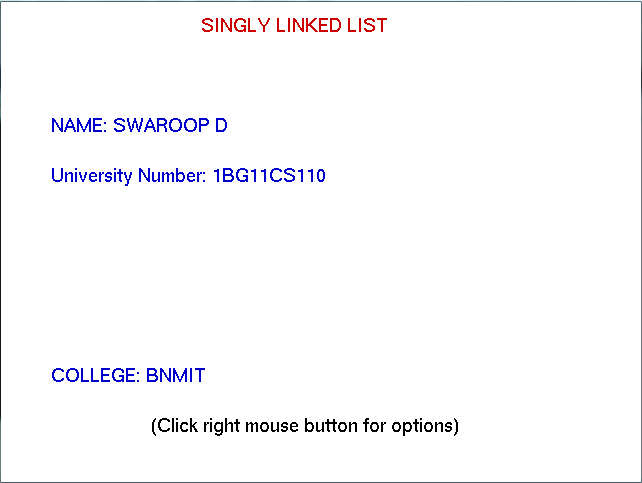


Fig 5.1 the introduction page

The user has to press right button of mouse to view the options.

* Insert front.
* Insert rear.
* Insert specified.
* Delete front.
* Delete rear.
* Delete specified.

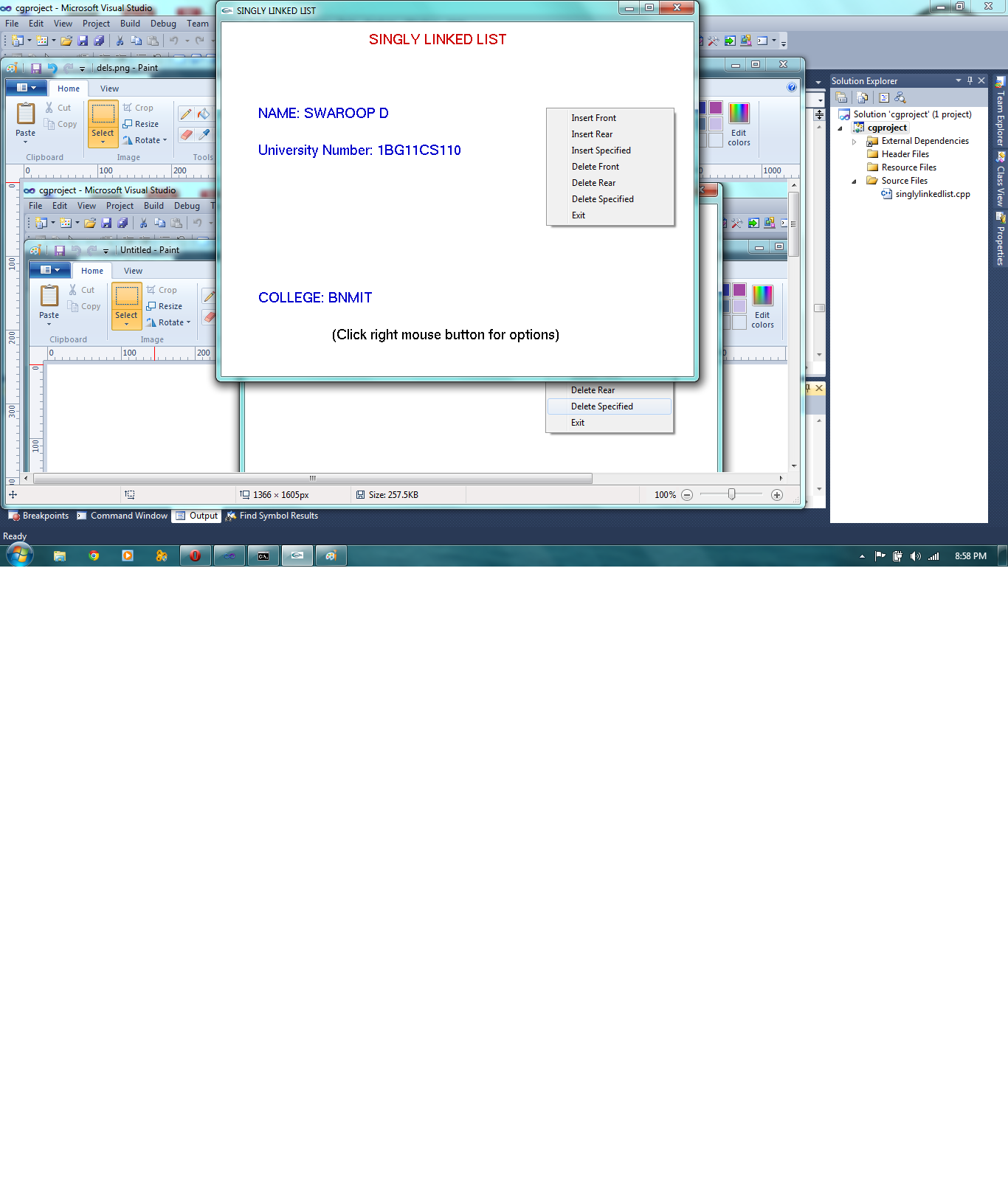
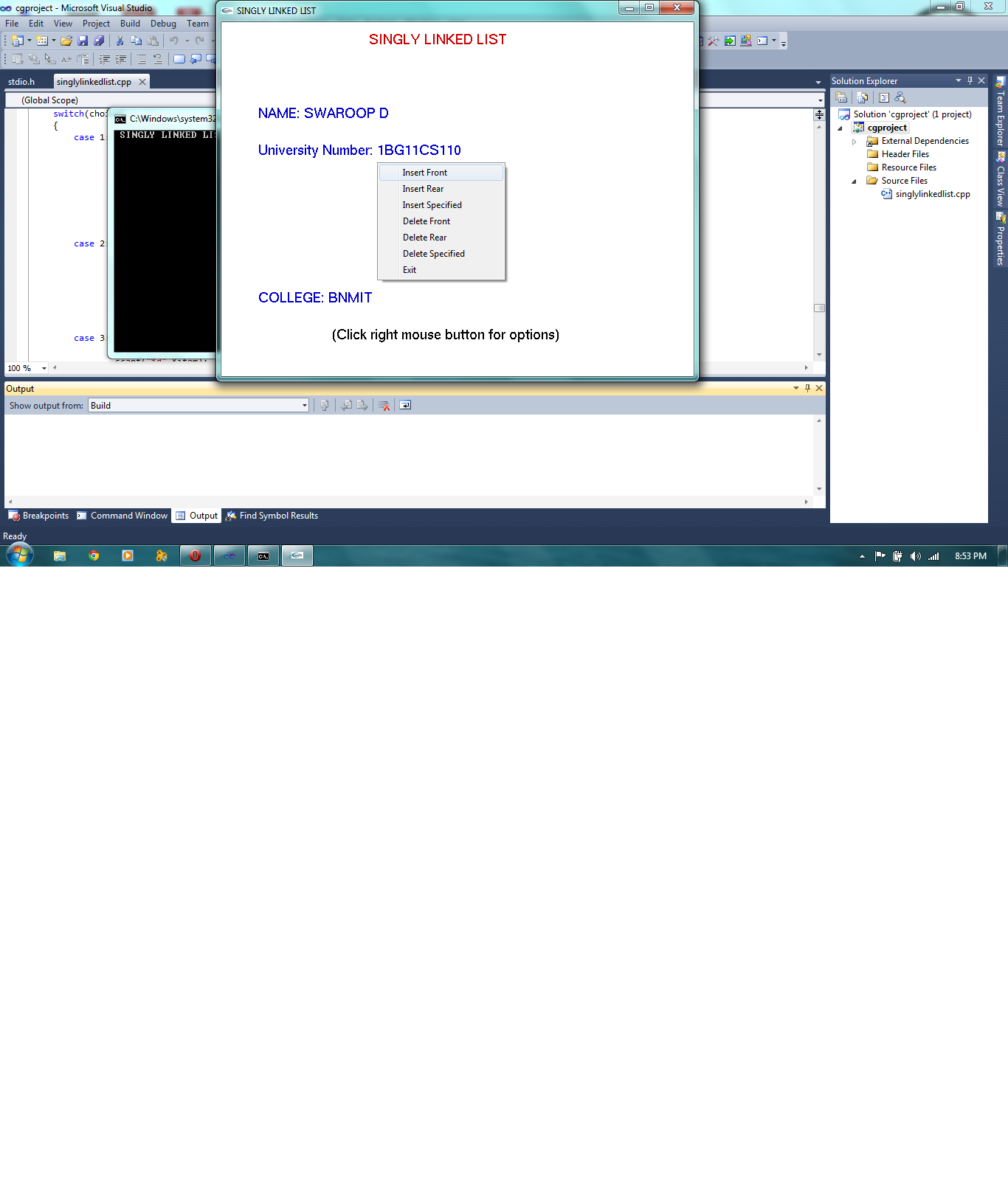
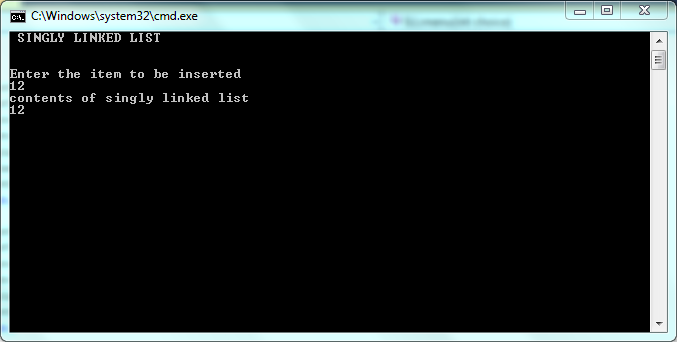


Fig 5.2 Options

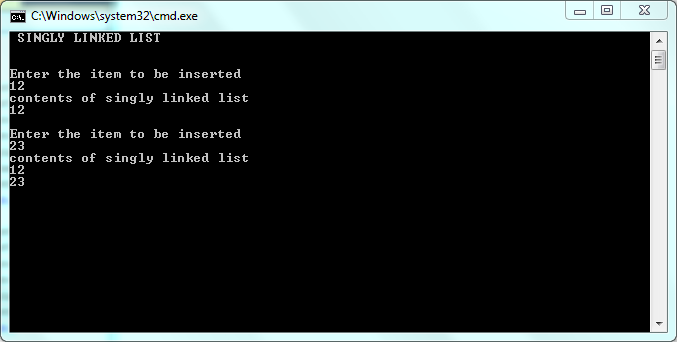
Let us see the options one by one.

* Insert front



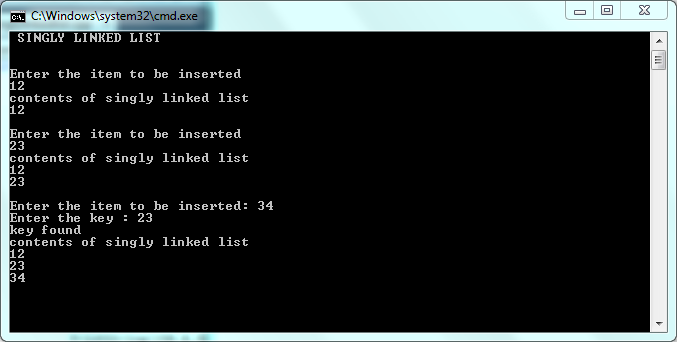


* Insert rear



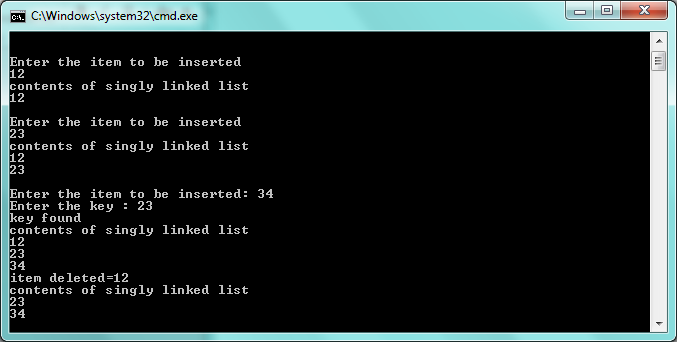


* Inserting specified



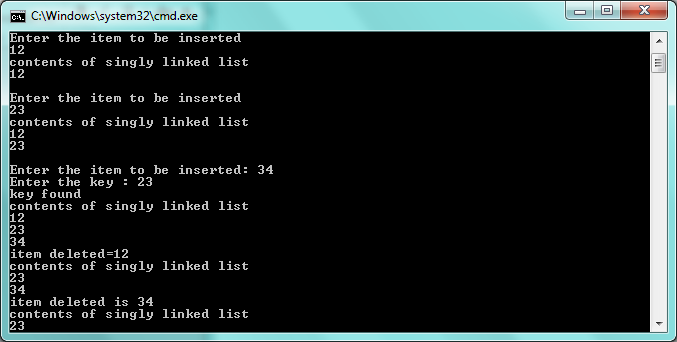


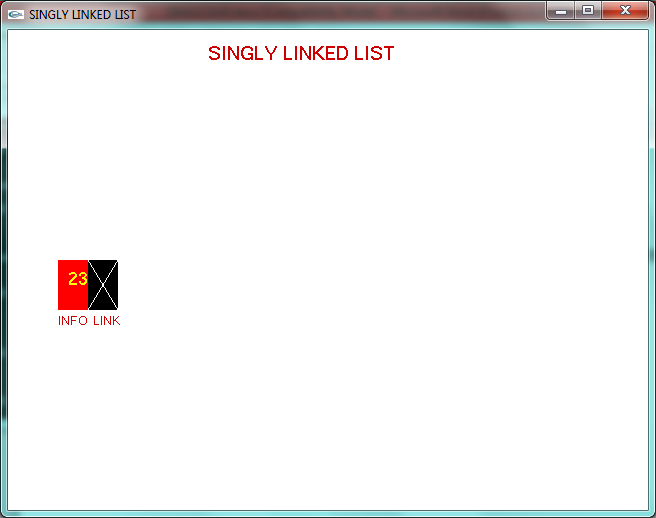
* Delete front



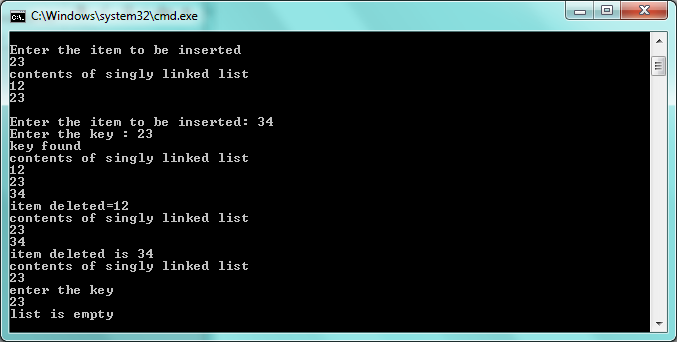


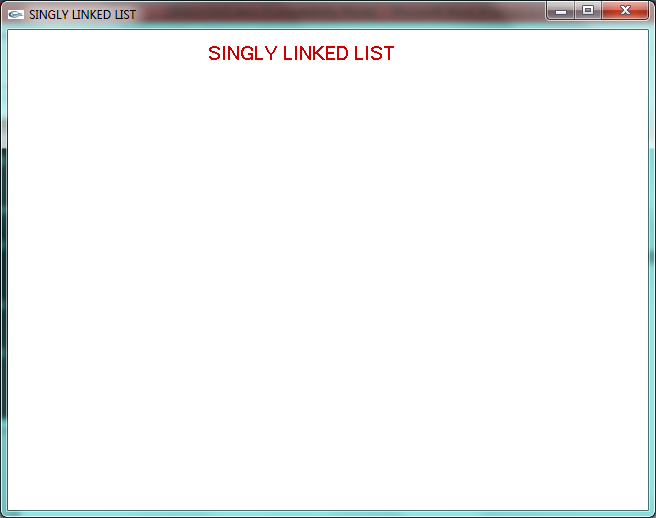
* Delete rear





* Delete a specified node





1. **CONCLUSION AND FUTURE ENHANCEMENT**

6.1 Conclusion

The project “Singly Linked List” has been designed and implemented as an exercise of “Computer Graphics and visualizations” laboratory.

* 1. Future work
* We can use mouse interaction to delete a particular node by clicking on it.
* We can make use the logic used in this project to design “Doubly Linked List”.

1. **BIBLIOGRAPHY**

7.1 Reference books

* Edward Angel: Interactive Computer Graphics A Top-Down Approach with OpenGL, 2nd Edition, Addison-Wesley, 2000.
* Foley, VanDam, Feiner, Huges, “Computer Graphics”, Addison-wesely 1997.
* Data structures, by Schaumm’s outline series
* http://en.wikipedia.org/wiki/Linked\_list
* http://www.opengl.org/News/Special/oim/Orth.html