**Assignment #1**

**전자전기공학과 20170771 배성찬**

**Povis ID : seongchanb**

|  |
| --- |
| **Honor Code**  **나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다.** |

**CSV Reader와 SVG 변환 프로그램 구현**

**1. 개요**

본 프로그램의 목적은 2차원 벡터 그래픽을 표현하는 XML 파일을 간단한 명령어 몇 가지를 통해 손쉽게 생성하는 것을 돕는 것을 목적으로 한다. 사용자가 내릴 수 있는 명령은 SVG 객체를 추가하는 Append, SVG 객체를 삭제하는 Remove, 현재 SVG 객체의 위치를 표현해 주고, 위치 이동을 돕는 Select, end SVG 객체에 데이터를 추가하는 cattr tattr, 각 오류상황 또는 사용자가 요청할 때마다 도움말을 제공하는 help, 프로그램을 종료하는 exit 등이 있다.

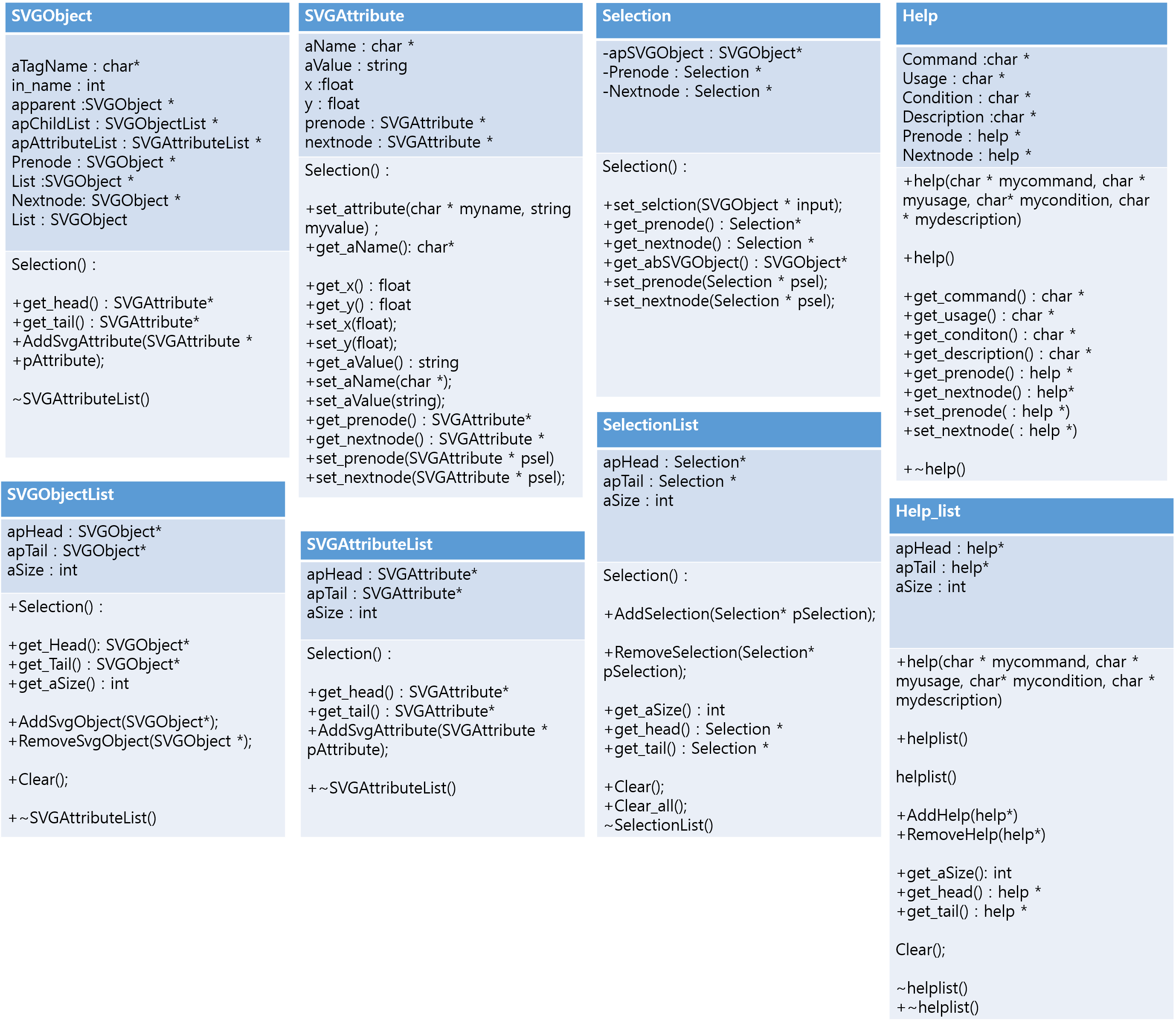
위와 같은 기능을 구현하기 위해, SVGObject, SVGAttribute, Selecttion 등의 Class를 정의했고, Object들을 연결리스트로 묶어 관리하기 위해 SVGObjectList 등의 list Class 역시 정의하여 사용하였다.

CSV Reader의 경우 ‘, ‘를 통해 내용을 구분 짓는 파일이고, 본 프로그램에 사용된 help.csv 파일은 각 기능의 이름, 사용법, 조건, 정의가 ‘,’로 나눠져 적혀있다.

이 프로그램의 구조도는 아래와 같다.

**2. Class Diagram 및 함수, 알고리즘 설명.**

이 프로그램의 Class Diagram은 아래와 같다.



각 Class의 메서드 중 get, set 함수는 private에서 숨겨져 있는 멤버 변수들의 값을 출력하고, 입력하는 함수이다. 단순한 함수이므로 get, set 함수들의 설명은 생락한다.

각 Class에 있는 prenode와 nextnode는 쌍방향 연결 리스트를 구현하기 위해 이전 노드와 다음 노드를 가리키는 포인터변수이다.

또한 네 개의 List Class의 멤버 변수는 모두 동일한 이름을 가지고 있는데 apHead, apTail,은 각각 제일 먼저 list에 들어온 object와 제일 나중에 list에 들어온 object를 가리키며, aSize는 list에 들어있는 object들의 개수를 나타낸다.

**1) SVGObject Class**

멤버 변수:

apParent; //부모 svg 객체 포인터

apChildList; // 자식 SVG 객체들에 대한 리스트 포인터

apAttributeList; // 객체의 Attribut들의 리스트 포인터

SVGObject \* prenode;//같은 세대 SVGObject list 에서 앞에 위치하는 object

SVGObject \* nextnode; //같은 세대 SVGObject list에서 뒤에 위치하는 object

**2) SVGObjectList Class**

멤버 변수:

SVGObject \* apHead;// list의 head를 가리키는 포인터

SVGObject \* apTail; // list의 tai을 가리키는 포인터

int aSize; //리스트의 크기

**3) SVGAttribute Class**

멤버 변수:

char \* aName; // 객체에 들어갈 데이터의 이름

string aValue; // 데이터의 값

float x; // tattr을 통해 들어올 transform의 x값

float y; //tattr을 통해 들어올 transform의 y값

SVGAttribute \* prenode;

SVGAttribute \* nextnode;

**4) SVGAttributeList Class**

멤버 변수:

SVGAttribute\* apHead; // list의 head를 가리키는 포인터

SVGAttribute\* apTail; // list의 tail을 가리키는 포인터

int aSize; // list의 크기를 나타내는 변수

**5) Selection Class**

멤버 변수:

SVGObject\* apSVGObject; // 선택된 객체

Selection \* prenode; // 리스트에서 이전 셀렉션

Selection \* nextnode; // 리스트에서 다음 셀렉션

**6) SelectionList Class**

멤버 변수:

Selection\* apHead ; // list에서 가장 앞에있는 객체의 주소

Selection \* apTail; // 가장 뒤에 있는 객체의 주소

int aSize;

**7) help Class**

기본 class 외에, csv파일을 읽어, 각 명령어에 대한 도움말을 저장할 class를 구현하였다.

멤버 변수:

char \* command; // 명령

char \* usage; // 용도

char \* condition; //명령 사용가능한 상태

char \* description; // 명령에 대한 기술

help \* prenode;

help \* nextnode;

**8) helplist Class**

help Class를 linked list로 관리하기 위해 구현한 class이다.

멤버 변수:

help\* apHead; // 리스트의 가장 앞에 있는 객체

help \* apTail; //가장 뒤에 있는 객체

int aSize; // 리스트의 크기

**대표 메서드들**

**1) List Object 들에서 add, remove 함수**

|  |
| --- |
| void SVGObjectList::AddSvgObject(SVGObject\* pObject)  {  if (aSize == 0)  {  apHead = pObject;  apTail = pObject;  pObject->set\_prenode(NULL);  pObject->set\_nextnode(NULL);  // 처음 들어온 Object를 첫 주소로 하고, 들어온 object의 next node를 초기화  aSize++; // node 갯수 증가  }  else  {  pObject->set\_prenode(apTail); // tail을 object의 prenode로 연결  apTail->set\_nextnode(pObject); // tail의 next node를 object로 연결  apTail = pObject; // tail을 최근에 들어온 object로 변경  pObject->set\_nextnode(NULL);  aSize++; // node 갯수 증가  }  } |

add 함수의 원리는 간단하다. 추가하고자 하는 object의 포인터를 받아. 먼저 리스트에 항목이 하나도 없는 경우 리스트의 시작을 가리키는 포인터와 끝을 가리키는 포인터에 받은 object의 포인터를 저장한다.

이미 리스트에 object가 있는 경우 tail에 object를 연결한다.

|  |
| --- |
| void SVGObjectList::RemoveSvgObject(SVGObject \* pObject)  {  if (pObject == apHead)  {  apHead = apHead->get\_nextnode();  if (aSize == 1)  {  apTail = NULL;  //head를 제거하는 경우에 리스트 안에 객체가 하나만 있으면, apTail=apHead이므로 tail도 초기화  }  aSize--;    }  else if (pObject == apTail)  {  apTail = pObject->get\_prenode(); // tail을 앞의 걸로 바꿔줌  apTail->set\_nextnode(NULL); // 새로운 TAIL의 NEXT NODE를 초기화해줌  pObject->set\_prenode(NULL); // pObject의 pre node를 초기화 해줌  aSize--;  }  else if (pObject != apHead && pObject != apTail && pObject->get\_nextnode() != NULL && pObject->get\_prenode() != NULL)  //HEAD와 TAIL이 아니면서, LINK 되어 있는 경우  {  pObject->get\_prenode()->set\_nextnode(pObject->get\_nextnode());  // object의 prenode의 nextnode를 object의 nextnode로 연결  pObject->get\_nextnode()->set\_prenode(pObject->get\_prenode());  // object의 nextnode의 prenode를 object의 prenode로 연결  pObject->set\_prenode(NULL);  pObject->set\_nextnode(NULL); // pObject의 연결을 모두 끊음  aSize--;  }  } |

head인 경우와 tail인 경우, 둘다 아닌 경우 세 개로 나눠 각자 맞춰서 object를 제거한다. 자세한 알고리즘은 주석을 보면 알 수 있다.

**2) List Object 들에서 Clear 함수**

|  |
| --- |
| void SVGObjectList::Clear()  {  SVGObject \*temp;  int i = aSize;  for (; apTail != NULL;)  {  temp = apTail;  RemoveSvgObject(temp);  i = aSize;  }  } |

리스트의 끝에서부터 셀렉션을 끝에서부터 하나씩 제거해간다.

**3) Clear\_all() 함수**

|  |
| --- |
| void SelectionList::Clear\_all()  {  Selection \*temp;  int i = aSize;  for (; apTail != NULL;)  {  temp = apTail;  temp->get\_abSVGObject()->get\_apParent()->get\_apChildList()->RemoveSvgObject(temp->get\_abSVGObject());  // parent의 childlist로 접근하여 세대간 연결리스트의 연결을 끊는다.  delete temp->get\_abSVGObject();  RemoveSelection(temp);  i = aSize;  }  } |

리스트의 끝에서부터 셀렉션을 끝에서부터 하나씩 제거해간다. clear() 함수에서 selection을 제거하는 동시에 object를 제거한다.

**4) 깊은 복사 함수**

|  |
| --- |
| void SVGObject::set\_atagname(char \* a)  {  aTagName = new char[strlen(a)+1];  strcpy(aTagName, a);  in\_name++;  } |

단순히 문자열의 시작주소만 받으면 얕은 복사가 일어나, 매개변수로 받은 변수의 값이 변할 경우, 멤버 변수의 값도 변한다. 따라서 멤버 변수에 문자열을 동적할당하고, strcpy 함수를 이용해, 저장한 문자열만 옮긴다.

**main의 함수들**

**1) object가 selection list에 있는지 확인하는 함수**

|  |
| --- |
| int if\_in\_sellist(SVGObject \* object, SelectionList \* list)  {  int n=0;  Selection \* selection = list->get\_head(); // list의 head를 받아 저장한다.  SVGObject\* obj\_in\_list;  for (; selection!= NULL;)  {  obj\_in\_list = selection->get\_abSVGObject();  // 현재 셀렉션에 저장된 객체를 받는다.  if (object == obj\_in\_list)  {  n++; // 셀렉션에 해당하는 오브젝트가 있으면 글을 높인다.  }  selection = selection->get\_nextnode();    }  return n;  } |

셀렉션 리스트에서 head부터 tail까지 셀렉션을 옮겨가며 셀렉션 안에 저장되어 있는 객체와 매개변수로 받은 객체를 비교하고 같으면 1을 반환하는 원리이다.

**2) 파일로 출력될 html 코드를 표준 출력해주는 함수**

|  |
| --- |
| void cout\_html\_code(int ifin, SVGObject \* object, int generation, SelectionList \*selectionlist)  {  for (int i = 0; i < generation; i++)  {  cout << "\t";  }  if (ifin == 1)  {  cout << "\*";  }  cout << "<" << object->get\_name();  SVGAttribute \* temp; // object 내의 attribute를 본다  if (object->get\_atrlist() != NULL) //attribute가 있는 경우 출력한다.  {  temp = object->get\_atrlist()->get\_head();  if (temp != NULL)  {  while (temp != NULL)  {  cout << " ";  if (temp->get\_aName() != NULL)  {  cout << temp->get\_aName();  }  if (temp->get\_aValue().compare("")!=0)  {  cout << "=" << '"' << temp->get\_aValue() << '"';  }  if (!strcmp(temp->get\_aName(), "transform"))  {  cout << fixed;  cout.precision(6);    cout << "=translate(" << temp->get\_x() <<","<< temp->get\_y() << ')' << '"';  }  temp = temp->get\_nextnode();  }  }  }  cout << ">";  if (object->get\_apChildList() != NULL)// child가 있는경우 child를 호출한다.  {  cout << "\n";  generation++; // child로 넘어갔으므로 세대에 1을 더한다.  cout\_html\_code(if\_in\_sellist(object->get\_apChildList()->get\_Head(), selectionlist), object->get\_apChildList()->get\_Head(), generation, selectionlist);  // 현재 객체의 자식세대 중 가장 앞에 있는 객체를 매개변수로하여 재귀함수를 호출한다.  generation--;  // 자식세대의 호출이 끝났으므로, 세대를 내린다.  }  if (object->get\_apChildList() == NULL)  // child가 없는, 마지막세대인 경우 html code를 닫는다.  {  cout << "</" << object->get\_name() << ">";  if (object->get\_nextnode() == NULL)  {  cout << "\n";  }  }  if (object->get\_apChildList() != NULL)  // 부모세대로 호출이 넘어오면, svg 객체를 닫는다.  {  for (int i = 0; i < generation; i++)  {  cout << "\t";  }  cout << "</" << object->get\_name() << ">"; //</객체이름>을 출력  if (object->get\_nextnode() == NULL) // 같은세대에서 마지막인 경우 엔터를 침  {  cout << "\n";  }  }  if (object->get\_nextnode() != NULL) // 같은 세대에 다음 노드가 있는 경우 다음노드로 함수 호출  {  cout << "\n";  cout\_html\_code(if\_in\_sellist(object->get\_nextnode(), selectionlist), object->get\_nextnode(), generation, selectionlist);    }        } }  if (object->get\_apChildList() != NULL)  {  for (int i = 0; i < generation; i++)  {  cout << "\t";  }  cout << "</" << object->get\_name() << ">";  if (object->get\_nextnode() == NULL)  {  cout << "\n";  }  }  if (object->get\_nextnode() != NULL)  {  cout << "\n";  cout\_html\_code(if\_in\_sellist(object->get\_nextnode(), selectionlist), object->get\_nextnode(), generation, selectionlist);    }  } |

재귀함수를 이용하여 XML 코드의 형태로 연결리스트를 출력했다. if\_in\_sellist 함수의 반환값을 받아 셀렉션 안에 있는 경우 \*를 출력하고, 자식 객체가 있는 경우 자식 객체를 매개변수로하여 함수를 다시 호출하여 포함관계대로 출력되도록 하였다.

attribute를 출력할 때 이름을 보고 cattr에서 저장한 것인지 tattr에서 저장한 것 인지를 구분하여 출력하였다.

**3) 자신의 자녀 객체와, 같은 리스트에 있는 객체를 모두 제거하는 함수**

|  |
| --- |
| int remove(SVGObject\* object\_remove, SVGObject \* root) // 자신 하위(child)와 뒤(next node)에 있는 object 모두 제거하는 함수, list->head()  {  SVGObject \* temp;  if (object\_remove == root) // li  {  //root 이므로 오류  return 0;  }  else  {  if (object\_remove->get\_apChildList() != NULL) // child가 있는 경우  {  temp = object\_remove->get\_apChildList()->get\_Head();  remove(temp, root); // child list의 head로 remove 호출  }  if (object\_remove->get\_apChildList() == NULL)// child가 없는경우  {  if (object\_remove->get\_nextnode() != NULL) //next node가 있는 경우  {  remove(object\_remove->get\_nextnode(), root); // 다음 노드로 remove 호출  }  if(object\_remove->get\_nextnode() == NULL) //child도, next node도 없는 경우  {  object\_remove->get\_apParent()->get\_apChildList()->RemoveSvgObject(object\_remove);  //parent의 apChildlist로 접근하여 object를 제거한다.  if (object\_remove->get\_apParent()->get\_apChildList()->get\_aSize() == 0)  //apChildlist에 아무것도 없을 경우, apChildlist를 해제한다.  {  object\_remove->get\_apParent()->set\_apChildList(NULL);  }  }  }  }  } |

재귀함수를 이용하여 자신과 같은 세대 및 아래세대의 Object 들을 모두 제거하는 함수를 구현했다. object의 삭제는 child도 nextnode도 없는 경우 이루어지고, 이에 해당하지 않는 경우는 child 또는 nextnode로 함수를 재호출했다.

**4) 특정 셀렉션안의 객체에 attribute를 추가하는 함수**

|  |
| --- |
| void add\_attribute  (Selection\* current\_selection, char\* new\_tagname, string value, float x = 0, float y = 0)    {  SVGObject \* current\_object;  SVGAttributeList \* new\_attributelist;  SVGAttribute \* current\_attribute;  SVGAttribute \* new\_attribute;  current\_object = current\_selection->get\_abSVGObject();  if (current\_object->get\_atrlist() == NULL)  {  new\_attributelist = new SVGAttributeList; // attributelist 할당  current\_object->set\_atrlist(new\_attributelist);  // attribute list를 objet 안의 apattributelist에 대입  // current\_object 안에 attribute list가 없는 경우 이를 할당해 준다.  current\_attribute = new SVGAttribute;  //그리고 current\_attribute를 할당해 준다.  current\_attribute->set\_aName(new\_tagname);  if (!strcmp(new\_tagname, "transform"))  {  current\_attribute->set\_x(x);  current\_attribute->set\_y(y);  }  else  {  current\_attribute->set\_aValue(value);  }  current\_object->AddAttribute(current\_attribute);  // current\_attribute를 생성해주고, apAttributelist에 대입한다.  }  else //current\_object 안에 attribute list가 있는 경우  {  current\_attribute = current\_object->get\_atrlist()->get\_head();  if (current\_attribute != NULL) // attribute list 안에 atribute 항목이 있는 경우  {  int count = 0;  while (current\_attribute != NULL)  {  if (!strcmp(current\_attribute->get\_aName(), new\_tagname))  // attribute 항목 중 해당하는 값의 종류를 가진 값이 있는경우  {  if (!strcmp(new\_tagname, "transform"))  {  current\_attribute->set\_x(x);  current\_attribute->set\_y(y);  }  else  {  current\_attribute->set\_aValue(value);  }  count++;  }  current\_attribute = current\_attribute->get\_nextnode();  }  if (count == 0) // 같은 이름을 가진 attribute가 없는 경우, attrubute를 할당해준다.  {  current\_attribute = new SVGAttribute; //그리고 current\_attribute를 할당해 준다.  current\_attribute->set\_aName(new\_tagname);  if (!strcmp(new\_tagname, "transform"))  {  current\_attribute->set\_x(x);  current\_attribute->set\_y(y);  }  else  {  current\_attribute->set\_aValue(value);  }  current\_object->AddAttribute(current\_attribute);  // current\_attribute를 생성해주고, apAttributelist에 대입한다.  }  }  else // attributelist는 있지만 attribute 항목은 없는 경우  {  current\_attribute = new SVGAttribute; //그리고 current\_attribute를 할당해 준다.  current\_attribute->set\_aName(new\_tagname);  if (!strcmp(new\_tagname, "transform"))  {  current\_attribute->set\_x(x);  current\_attribute->set\_y(y);  }  else  {  current\_attribute->set\_aValue(value);  }  current\_object->AddAttribute(current\_attribute);  // current\_attribute를 생성해주고, apAttributelist에 대입한다.  }  }  } |

selection의 주소, attribute의 이름과 값을 받아 selection에 들어있는 object에 Attribute를 저장한다. cattr에서 부르는 경우 데이터 값을 string value에 전달해주고, tattr에서 부르는 경우 attribute이름을 transform으로 하고, 데이터 값은 x와 y가 float 형태로 전달해주어, 서로 다른 두 가지 기능을 수행하도록 하으며. 이미 동일한 이름을 가진 attribute가 있는 경우 데이터만 바꾸고, 없는 경우 새로 동적할당 하여 대입하도록 하였다.

**5) csv파일을 읽어 도움말을 출력해주는 함수**

|  |
| --- |
| void get\_helpfile(helplist\* helplist)  {  helplist->Clear();  ifstream ReadFile("help.csv");  string line;  help\* new\_help;  char \* sentence;  char \* command;  char \* usage;  char \* condition;  char\* description;  int len\_sentence;  getline(ReadFile, line);  while (ReadFile.peek() != EOF)  {  getline(ReadFile, line);  const char \* x = line.c\_str();  int f = strlen(x);  sentence = new char[f + 1];  strcpy(sentence, line.c\_str());  int j = 0;  //while (1)  {  int count = 0;  int length = strlen(sentence) + 1;  for (int i = 0; i < length; i++)  // CSV 파일을 읽는 부분이다.  // 문장을 받아 한 글자씩 읽으면서, ,가 나오면 내용을 \0으로 바꾼다.  // 그러면 , 단위로 string이 나뉘게 되는데 이때 , 의 위치를 저장한다.  // ,부터 ,까지 문자열을 나눠서 저장한다.  {  if (sentence[i] == ',' && count == 0)  {  sentence[i] = '\0';  command = new char[i - j + 1];  strcpy(command, &sentence[j]);  j = i + 1;  count++;  }  if (sentence[i] == ',' && count == 1)  {  sentence[i] = '\0';  usage = new char[i - j + 1];  strcpy(usage, &sentence[j]);  j = i + 1;  count++;  }  if (sentence[i] == ',' && count == 2)  {  sentence[i] = '\0';  condition = new char[i - j + 1];  strcpy(condition, &sentence[j]);  j = i + 1;  count++;  }  if (count == 3)  {  description = new char[length + 1 - j];  strcpy(description, &sentence[j]);  new\_help = new help(command, usage, condition, description);  // 새로운 헬프 객체를 만든다.  helplist->AddHelp(new\_help);  // 새로운 헬프 객체를 리스트에 저장한다.  count = 0;  delete[]command;  delete[]usage;  delete[]condition;  delete[]description; // 각 순간 할당했던 문장들을 해제한다.  }  }  delete[]sentence;  }  }    } |

이 함수의 핵심 알고리즘은 csv 파일에서 ,를 기준으로 하여 문자열을 저장하는 것이다. 본 프로그램의 경우, csv 파일의 한 문장을 읽은 뒤 string에 저장하고, 저장된 문장을 한 글자씩 읽으며 ,를 골라내 \0으로 바꾸었다. 문자열을 다룰 때 \0이 기준이된다는 점을 활용하였고, \0의 위치를 int형 변수 j에 담아 놓고, j부터 다음 \0까지의 문자열을 복사하여 순서대로 동적할당된 command, usage, condition, description에 저장했다. 그 다음 저장된 데이터들을 동적할당된 help로 보내고, helplist에 추가하여, csv파일로부터 동작 정보를 얻어냈다.

**int main() 안의 핵심 알고리즘**

**1) 사용자 입력을 받는 알고리즘**

|  |
| --- |
| cout << "<SVGMaker> ";  c=getchar();  if(c=='\n') // 엔터를 치는 경우 처음으로 돌아간다.  {  continue;  }  else  {  command[0] = c;  cin>> &command[1];  // 처음 받은 글자를 첫 글자로 하고, 남은 문자를 받는다.  } |

초기에는 모든 STL을 사용하면 안 되는 줄 알아, string과 stringstream 없이 문자열로만 구현하였다. int main()이 호출되면 제일 먼저 문자를 하나 입력받는다. 만약 엔터일 경우 반복문의 처음으로 돌아가고 아닌 경우, 처음받은 글자를 첫 글자로 하는 문자열을 생성한다.

**2) 사용자 입력 예외를 판단하는 알고리즘**

|  |
| --- |
| if (getchar() == ' ')  {  cin >> value;  num = atoi(value.c\_str());  if (num==0)  {  get\_helpfile(&helplist);  select\_help(&helplist, command);  continue;  }    } |

데이터 형이 정해진 값을 받는 경우 예외 판단을 string to 함수들을 활용하여 구현했다. 일단 데이터는 모두 string으로 받고 atoi나 atof를 사용하여 적절한 자료형으로 변환해주었는데, 위의 함수들이 자료형 변환이 불가능할 때 0 을 반환함을 활용하여 예외처리를 해주었다.

**3) append 핵심 알고리즘**

|  |
| --- |
| Selection \* s\_parent = selectionlist.get\_tail();  // 부모세대 selection  SVGObject \* parent = (\*s\_parent).get\_abSVGObject();  // parent object의 주소  char \* name = parent->get\_name();  Selection \* s\_child=new Selection;  // 자녀세대 selection  SVGObject \*child = new SVGObject;  // 자녀세대 object 동적할당  (\*child).set\_atagname(new\_tagname);  //object 이름 대입  (\*child).set\_apParrent(parent);  //자녀세대 object의 parent에 대입    if (parent->get\_apChildList() == NULL)  {  (\*parent).set\_apChildList(new SVGObjectList);  // childlist 동적할당.  }    (\*parent).AppendChild(child);  //할당한 object를 parent의 apChildList에 대입.  selectionlist.Clear();// selection list clear  (\*s\_child).set\_selction(child);//child selection 형성    selectionlist.AddSelection(s\_child);  //slelctionlist에 s\_child 추가  for (int j = 0; j < num-1; j++)  {  s\_child = new Selection;  // 자녀세대 selection 동적할당  child = new SVGObject;  // 자녀세대 object 동적할당  (\*child).set\_atagname(new\_tagname); //object 이름 대입    (\*child).set\_apParrent(parent);  //자녀세대 object의 parent에 대입  (\*parent).AppendChild(child);  //ob parent의 apChildList에대입  (\*s\_child).set\_selction(child);  //child selection 형성  selectionlist.AddSelection(s\_child);  //slelctionlist에 s\_child 추가  } |

Attrib 사용자가 append 명령어를 통해 추가할 object와 그 개수를 입력해주면, 제일 먼저 해당하는 selection의 apChildList가 null이 아닌지 확인하고, null인 경우 동적할당해준다. 그다음, object를 할당하고 명령어를 통해 입력받은 이름을 object에 대입한뒤 apChildlist에 옮겨준다. for 문을 이용하여 입력받은 개수만큼 반복한다.

**4) select 핵심 알고리즘**

|  |
| --- |
| Selection\* current\_selection = selectionlist.get\_tail();  SVGObject\* temp = current\_selection->get\_abSVGObject();  Selection\* new\_selection; // 동적할당할 selection    if (temp->get\_apChildList() == NULL)    {  get\_helpfile(&helplist);  select\_help(&helplist, command);  continue;  }    temp = temp->get\_apChildList()->get\_Head();  int count = 0;  if (num == -1)  {    while ( temp!=NULL)  {  if (!strcmp(temp->get\_name(), new\_tagname))  {  new\_selection = new Selection;  new\_selection->set\_selction(temp);  selectionlist.AddSelection(new\_selection);  count++;  }  temp = temp->get\_nextnode();  }  if (count > 0)  {  selectionlist.RemoveSelection(selectionlist.get\_head());  cout << count << " item(s) is selected.\n";  }  // 이름이 동일한 object가 있는 경우  else  {  get\_helpfile(&helplist);  select\_help(&helplist, command);  continue;  }  }  else if (num > 0)  {  int count2 = 0;  while (temp != NULL)  {  if (!strcmp(temp->get\_name(), new\_tagname))  {  count++;  }  if (count == num)  {  new\_selection = new Selection;  new\_selection->set\_selction(temp);  selectionlist.AddSelection(new\_selection);  count2++;  }  else { temp = temp->get\_nextnode(); }  }    if (count2 == 1)  { selectionlist.RemoveSelection(selectionlist.get\_head());  cout << 1 << " item(s) is selected.\n";  }  else {  get\_helpfile(&helplist);  select\_help(&helplist, command);  }  }  } |

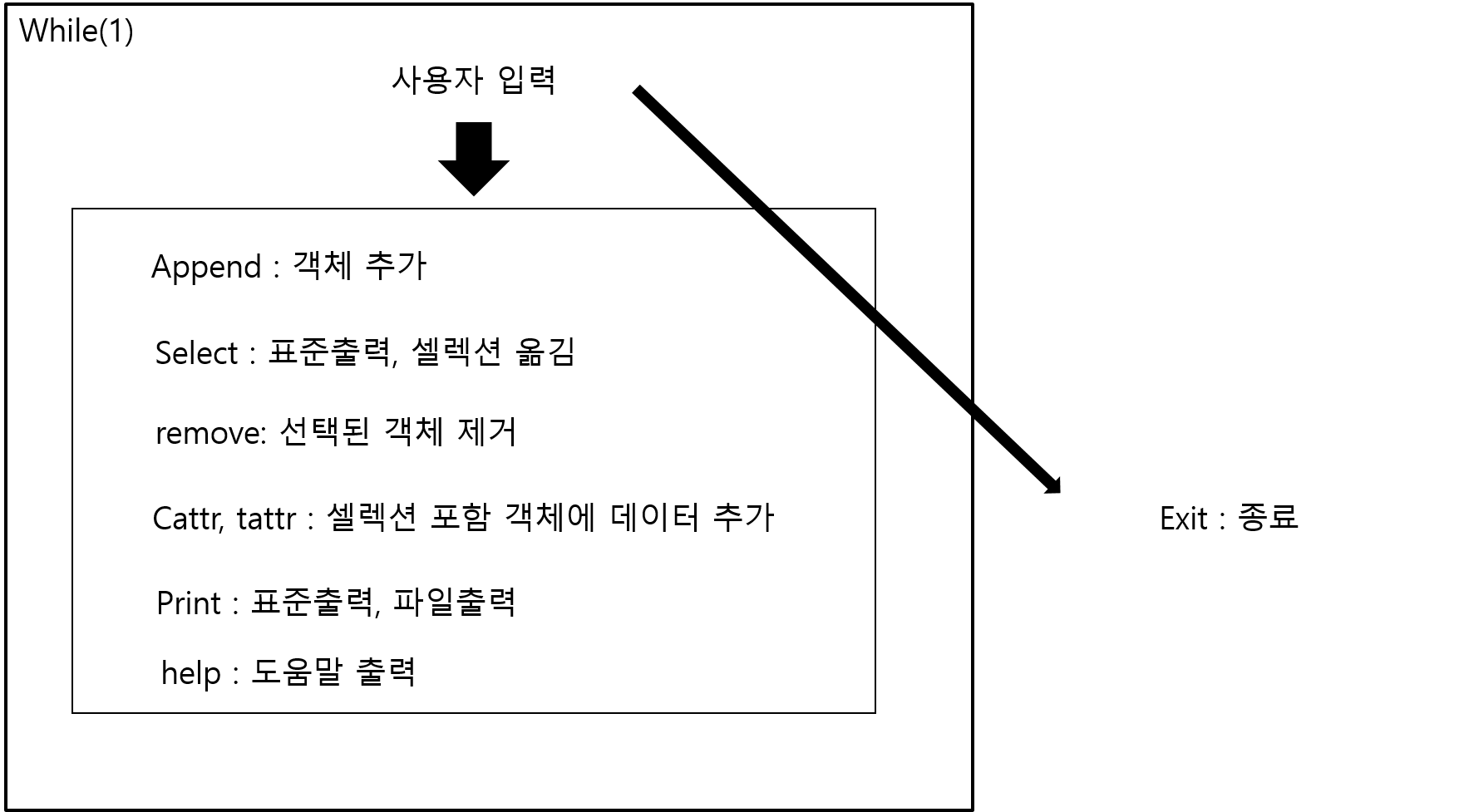
예외 처리 방식은 앞에서 설명한 예외 처리방식과 유사하다. 본 함수는 사용자로부터 object name과 숫자를 입력받아 현재 셀렉션 아래에 동일한 이름을 가진 object로 셀렉션을 옮기는 기능을 수행한다. 먼저 num의 초기값을 -1로 잡고, atoi로 string을 받아 옮겨준다. 옮겨진 후 num의 값이 -1이라면 숫자를 입력받지 않은 것이기 때문에 해당하는 이름을 가진 모든 객체를 리스트에 추가하고, 0이라면 숫자가 아닌 값을 입력받은 것이기 때문에 오류를 출력, 0보다 큰 정수라면 값을 제대로 입력 받은 것이기 때문에 n번째를 리스트에 추가한다.

**5) remove 핵심 알고리즘**

|  |
| --- |
| while (temp!=NULL)  {  if (temp->get\_abSVGObject()->get\_apChildList() != NULL)  {  remove(temp->get\_abSVGObject()->get\_apChildList()->get\_Head(), &root);  } // 하위 모두 제거  temp = temp->get\_nextnode();  }  selectionlist.Clear\_all();  if (parent->get\_apChildList()->get\_Head() == NULL)  {  parent->set\_apChildList(NULL);  }  selection = new Selection;  selection->set\_selction(parent);  selectionlist.AddSelection(selection); |

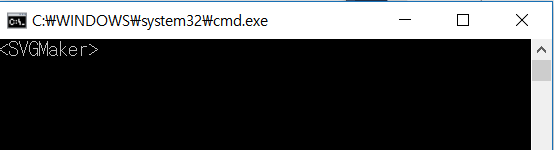
remove의 핵심알고리즘은 remove 함수이고, main의 remove 알고리즘은 단순하다. 리스트 안의 각 object의 자식 객체중 제일 처음인 객체를 매개변수로 하여 remove를 호출하여 아래의 object를 모두 제거하고, 자기자신을 제거하는 과정을 반복한다.

**3. 프로그램 구조**

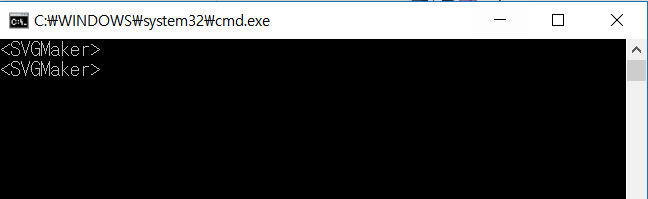


**4. 프로그램의 실행 및 예제**

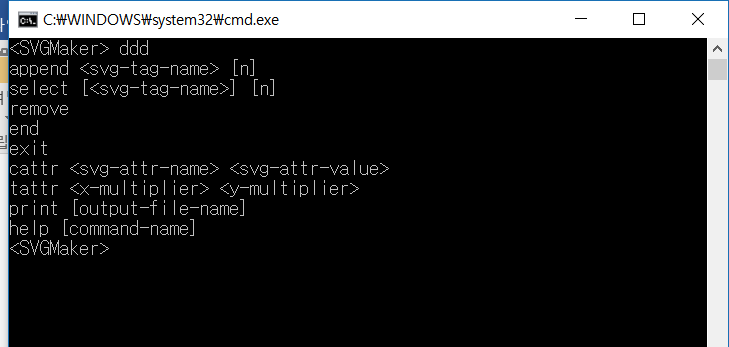
- 프로그램을 실행하면 사용자에게 입력을 요구한다.



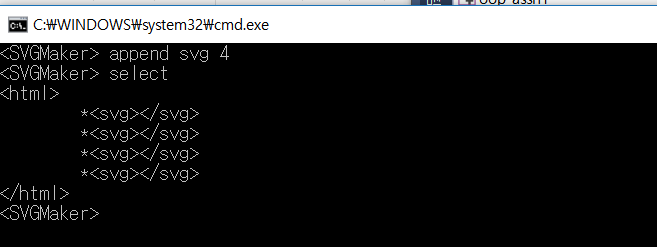
- 엔터를 입력하면 재입력을 요구한다.

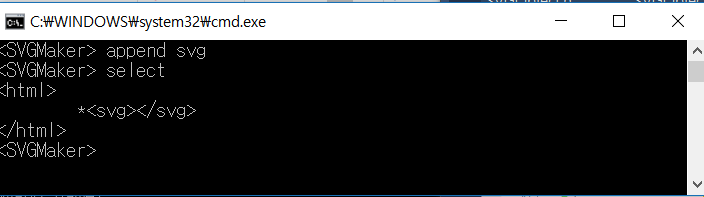


- 없는 명령어 입력시 도움말을 출력한다.



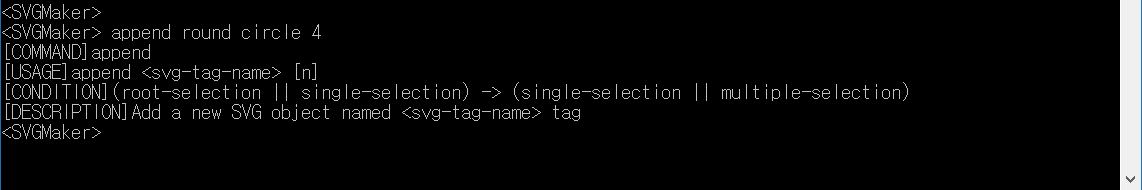
-append 이름 입력시 해당하는 이름을 가진 객체가 하나 추가되고, append 이름 숫자 입력시 n개가 추가된다. 그리고 selection이 이동한다.





-multi-selection, 맞지 않는 명령어가 입력된 경우 오류를 출력한다.

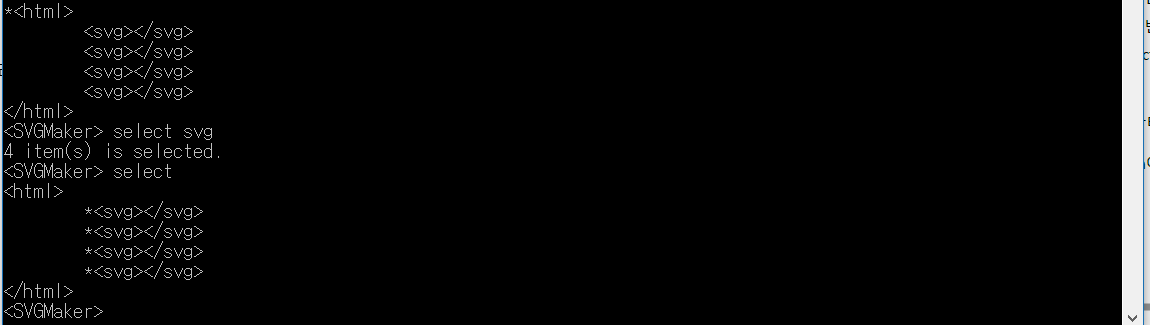


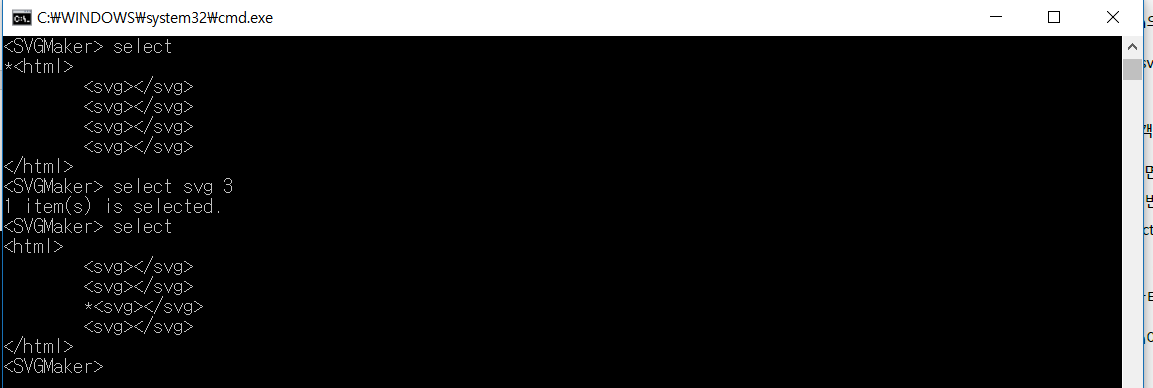


- select 만 입력될 경우 현재 리스트를 표준출력하고, 현재 셀렉션에 있는 객체는 \*를 붙여 표시해준다.

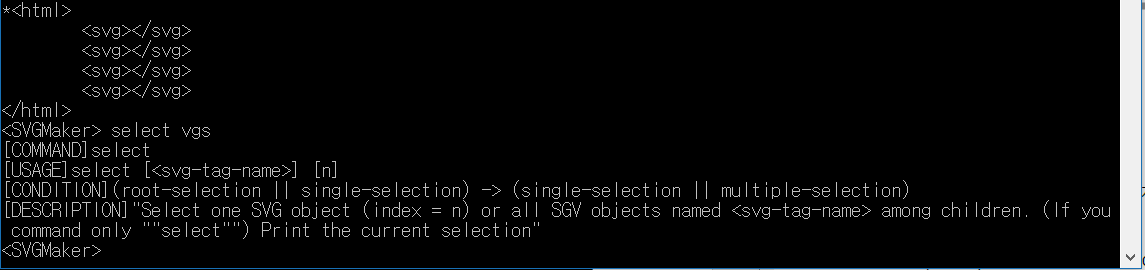


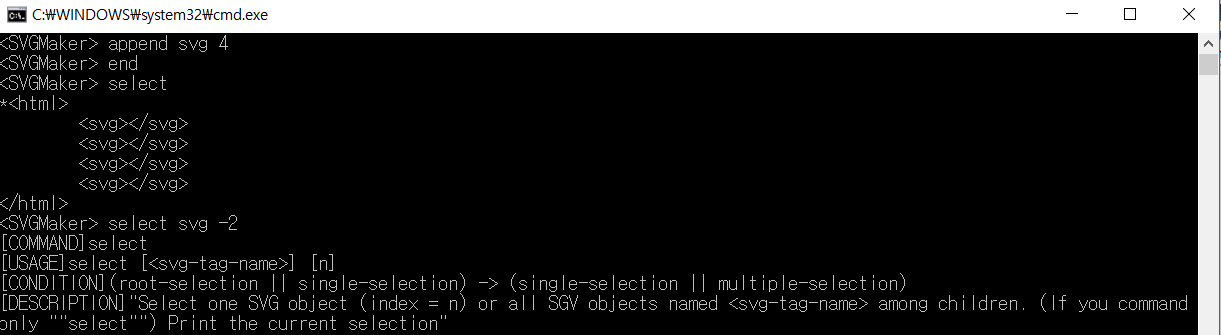
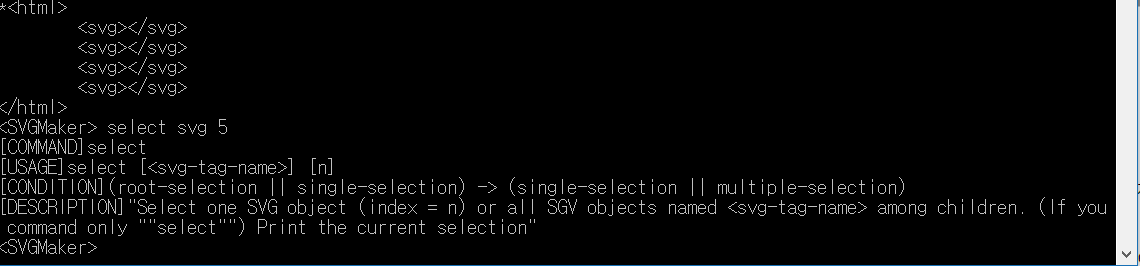
-select와 이름 그리고 추가로 숫자를 입력하는 경우 주어진 이름과 순서에 해당하는 object로 셀렉션을 옮긴다. 그리고 몇 개의 object가 선택되었는지 출력해준다.



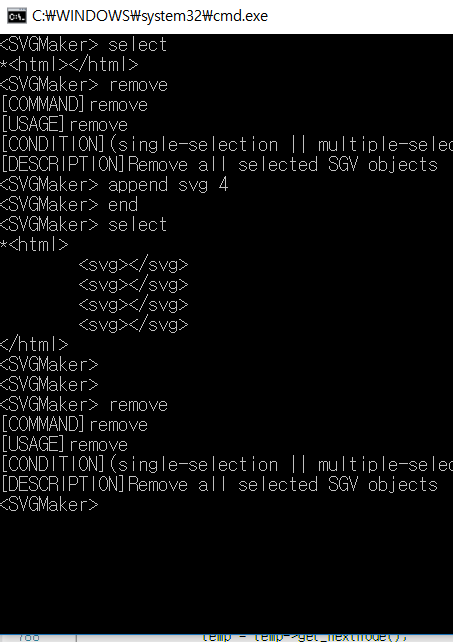


- 없는 object를 찾거나, 존재하는 개수 이상의 숫자를 입력하는 등의 조건에는 에러를 출력하고, selection을 유지한다.





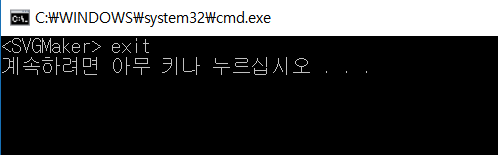
- remove를 입력하면 해당 selection 아래에 있는 child와 자신을 모두 삭제하고, 삭제 후에는 부모를 셀렉션리스트에 넣는다. 만약 root-selection일 경우, 에러를 출력한다.

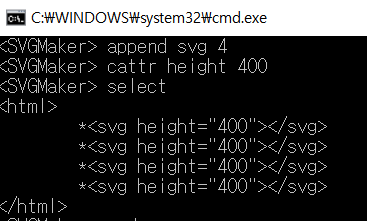
- end를 입력하면 현재 셀렉션의 부모 노드로 셀렉션을 옮기고, 현재 셀렉션이 root이면 오류를 출력한다.



- exit가 입력되면 프로그램이 종료된다.



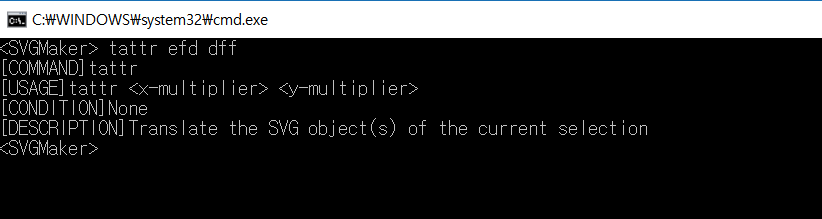
- cattr, attribute 이름, 값을 입력해주면 해당 selection에 attribute를 추가하거나 수정한다. 동일한 이름을 가진 attribute가 있으면 수정해주고, 없으면 새로 생성한다.



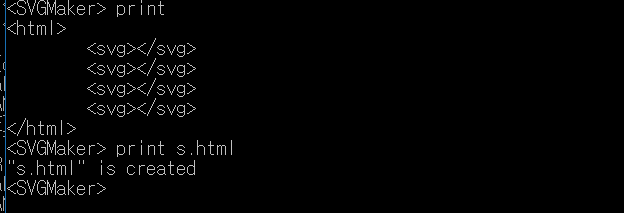
-tattr과 float 두 개를 입력해 주면, transform=”translate(x,y)” 형식의 attribute를 저장해준다. 이미 transform 형식의 데이터가 있는 경우 값을 바꿔주기만 한다.



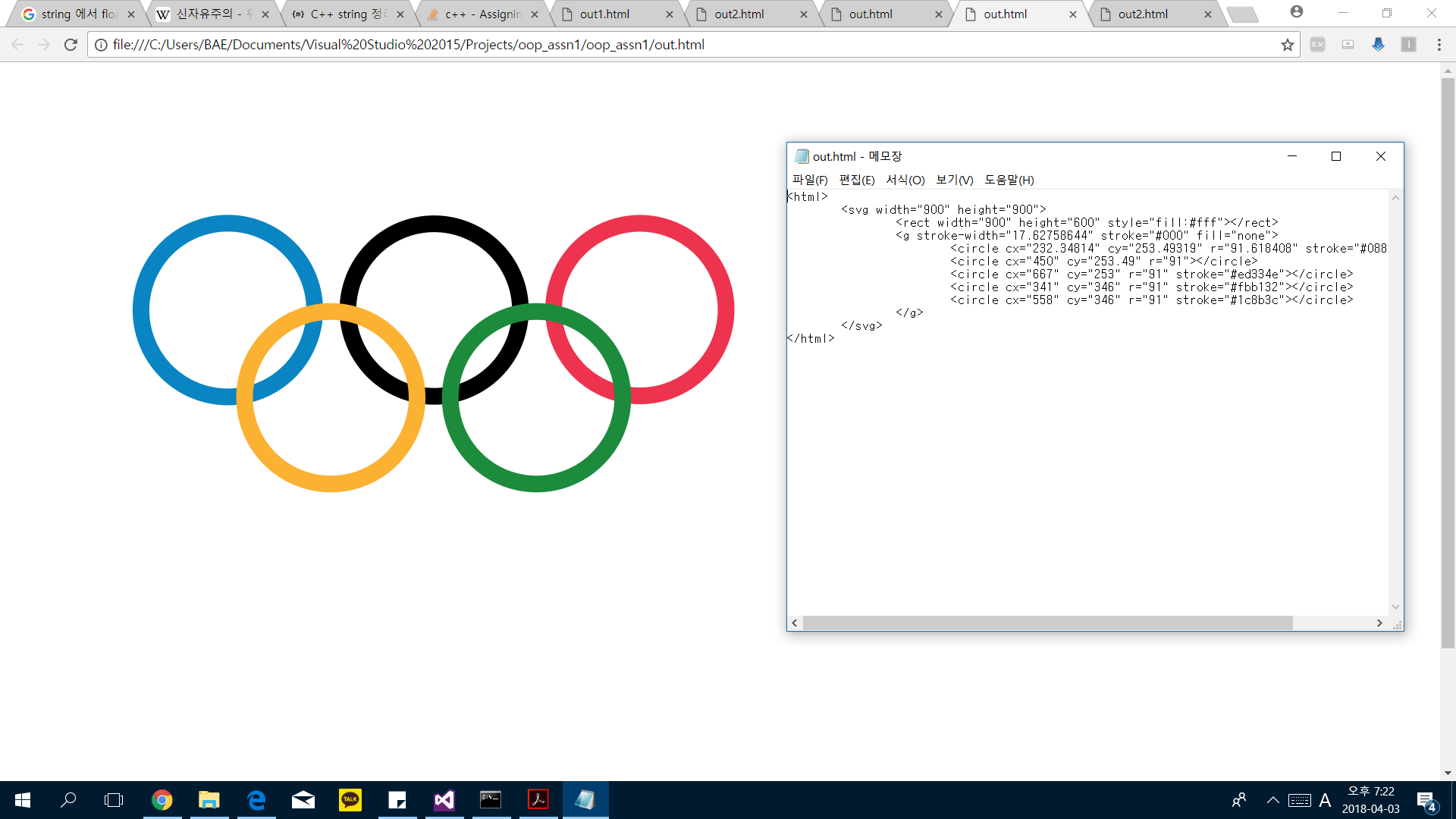
- tattr에 형식에 맞지 않는 값들이 들어오면 오류를 출력한다.



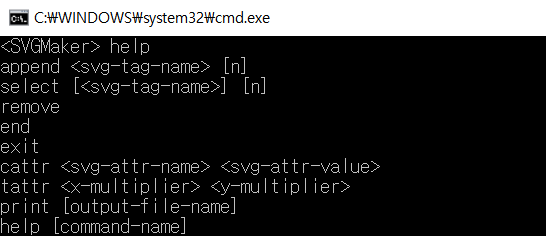
- print 만 입력하면 현재 트리를, print, 파일이름을 입력하면 해당하는 파일을 생성한다.



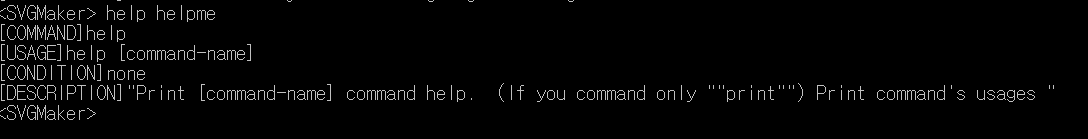
생성된 파일을 chrome으로 열어 보았다.



-help만 입력하면 저장된 도움말의 usage 부분을, help와 명령어 이름을 입력하면 해당 명령어의 정보를 모두 제공한다. 없는 명령어 이름이 입력되면 오류를 출력한다.







**5. 토론**

- string과 관련된 함수를 이용하여 각종 예외조건들을 고려해주었다.

- 세겹의 linked list를 이용하여 tree를 구성하였다.

- CSV file reader를 배열을 통해 구현하였다.

\_ 재귀함수를 이용해 각종 난해한 출력함수들을 구현하였다.

**6. 결론**

- CSV file reader와 XML file 생성프로그램을 구현하였다.

**7. 개선방향**

- stl을 아예 사용하지 말아야하는 줄 알고 처음에 stringstream과 string을 안 쓰고 짜다가, 마무리쯤부터 사용하기 시작했는데, 미리 사용했다면 코드도 훨씬 단순해졌을 것이고, 처리해 줄 수 있는 예외도 많았을 것이다. 또한 상속을 사용하면 불필요한 get, set 함수의 사용을 줄일 수 있어 코드가 더욱 단순해질 것이다.

buffer 초기화를 위해 rewind(stdin)을 사용했는데, 이 함수는 windows 환경에서는 잘 작동하지만, 리눅스 상에서는 작동하지 않는다. 다른 함수를 찾아 buffer 초기화를 해보려했지만 마땅한 함수를 찾지 못하였다. 다른 알고리즘을 생각하여 buffer 초기화를 이루면 좋을 것 같다.