UNIVERSIDAD DE SAN BUENAVENTURA CALI

FACULTAD DE INGENIERIAS - PROGRAMA DE ING. SISTEMAS

TOMAS MANCERA – 98649
CARLOS PEREZ – 1155961
SEBASTIAN LOPEZ – 97500
JUAN MANUEL PADILLA – 97196

1.1

```cpp
#include <fstream>
#include <iostream>
#include <sstream>
#include "crearGenerados.h"
// Esta funcion nos permite crear las funciones de acuerdo a lo que quiera el usuario
void menuFuncion(std::stringstream &flujoFuncionesCpp, std::stringstream &flujoFuncionesPy, std::stringstream &flujoFuncionesHtml)
{
  bool bucle(true);
  bool condicion(true);
  std::string nombreFuncion("");
  std::string tipoFuncionCpp;
  std::string variableFuncionCpp;
  std::string retornoFuncionCpp;
  std::string variableFuncionPy;
  std::string retornoFuncionPy;
  std::stringstream flujoParametrosCpp("");
  std::stringstream flujoParametrosPy("");
  std::stringstream flujoDescripcionFuncion("");
  std::string tipoDatoParametro("");
  int opcion(0);
  int opcionParametro(0);
  int opcionParametroDos(0);
  int opcionOtroParametro(0);
  std::string nombreParametro;
  std::string nombreOtroParametro;
  std::string comaParametros("");
  std::string descripcionFuncion("");


```

Se tiene una función void llamada “menuFuncion” que nos permite crear las funciones que el usuario desea. A continuación se evidenciará las opciones que tiene un usuario en nuestro generador de código al construir su función.

1.2

```cpp
  do
  {
    std::cout << "Elige el tipo de funcion que deseas...\n\n";
    std::cout << "1.void\n2.int\n3.double/float\n4.bool\n5.string\n";
    std::cin >> opcion;
    switch (opcion)
    {
    case 1:
      tipoFuncionCpp = "void ";
      variableFuncionCpp = "";
      retornoFuncionCpp = "";
      variableFuncionPy = "";
      retornoFuncionPy = "return";
      break;
    case 2:
      tipoFuncionCpp = "int ";
      variableFuncionCpp = "int aux = 0;";
      retornoFuncionCpp = "return aux;";
      variableFuncionPy = "aux = 0";
      retornoFuncionPy = "return aux";
      break;
    case 3:
      tipoFuncionCpp = "double ";
      variableFuncionCpp = "double aux = 0.0;";
      retornoFuncionCpp = "return aux;";
      variableFuncionPy = "aux = 0.0";
      retornoFuncionPy = "return aux";
      break;
```

1.3

```
58      case 4:
59        tipoFuncionCpp = "bool ";
60        variableFuncionCpp = "bool aux = false;";
61        retornoFuncionCpp = "return aux;";
62        variableFuncionPy = "aux = False";
63        retornoFuncionPy = "return aux";
64        break;
65      case 5:
66        tipoFuncionCpp = "std::string ";
67        variableFuncionCpp = "std::string aux = "
68                             "\"  \";";
69        retornoFuncionCpp = "return aux;";
70        variableFuncionPy = "aux = "
71                            "\"  \"";
72        retornoFuncionPy = "return aux";
73        break;
74      }
75      if (opcion >= 1 && opcion <= 6)
76      {
77        std::cout << "\nIngresa el nombre para tu funcion...\n";
78        std::cin >> nombreFuncion;
79        // solicita al usuario la descripcion para la funcion que esta creando
80        std::cout << "Ingresa la descripcion para tu funcion...\n";
81        std::cin.ignore();
82        getline(std::cin,descripcionFuncion);
```

1.4

```
84        while (bucle == true)
85        {
86          std::cout << "Quieres ingresar parametros para tu funcion?...\n";
87          std::cout << "1. Si\n2. No\n";
88          std::cin >> opcionParametro;
89          if (opcionParametro == 2 || opcionParametro == 1)
90          {
91            bucle = false;
92          }
93        }
94      }
95
96      switch (opcionParametro)
97      {
98      case 1:
99        while (condicion == true)
100       {
101         do
102         {
103           std::cout << "Elige el tipo de dato para tu "
104                        "parametro...\n1. char\n2. int\n3. double/float\n4. bool"
105                        "\n5. string\n";
```

1.5

```
107             std::cin >> opcionParametroDos;
108             if (opcionParametroDos >= 1 && opcionParametroDos <= 5)
109             {
110
111               switch (opcionParametroDos)
112               {
113               case 1:
114                 tipoDatoParametro = "char ";
115                 break;
116
117               case 2:
118                 tipoDatoParametro = "int ";
119                 break;
120
121               case 3:
122                 tipoDatoParametro = "double ";
123                 break;
124
125               case 4:
126                 tipoDatoParametro = "bool ";
127                 break;
128
129               case 5:
130                 tipoDatoParametro = "std::string ";
131                 break;
132               }
```

1.6

```
134               std::cout << "\nIngresa el nombre para tu parametro...\n";
135               std::cin >> nombreParametro;
136             }
137           } while (opcionParametroDos < 1 || opcionParametroDos > 6);
138
139           flujoParametrosCpp << comaParametros << tipoDatoParametro << nombreParametro;
140           flujoParametrosPy << comaParametros << nombreParametro;
141
142           do
143           {
144             std::cout << "\nDesea ingresar mas parametros a su funcion?...\n";
145             std::cout << "1. Si\n2. No\n";
146             std::cin >> opcionOtroParametro;
147             if (opcionOtroParametro == 1)
148             {
149               comaParametros = ",";
150             }
151             if (opcionOtroParametro == 2)
152             {
153               condicion = false;
154             }
155           } while (opcionOtroParametro < 1 || opcionOtroParametro > 2);
156         }
157       }
158       break;
```

En la imagen 1.2 hasta la imagen 1.6 Se continua con ciclos do while que permite dar la variedad de opciones que el usuario puede tomar en el generador de código. Contiene todos los tipos de funciones, si la función recibe parámetros y su cantidad, como también el valor y tipo de dato que estos contienen. Solicita el nombre de la función que el usuario desea colocar. Cuenta con diferentes condicionales para ser especifico con las decisiones que tome el usuario para ejecutar su código. Aspecto importante es que contiene la sintaxis marcada para cada lenguaje en cada opción elegida.

1.7

```
160   } while (bucle == true);
161
162   // ordenamiento de los flujos respectivos para funciones
163   flujoFuncionesCpp << "\n//" << descripcionFuncion << "\n\n" << tipoFuncionCpp << nombreFuncion << "("
164                     << flujoParametrosCpp.str() << "){\n"
165                     << variableFuncionCpp << "\n"
166                     << retornoFuncionCpp << "\n}" << std::endl;
167   flujoFuncionesPy << "  " <<"\n#"<<descripcionFuncion << "\n\n" <<"def " << nombreFuncion << "(" << flujoParametrosPy.str()
168                    << "):\n"
169                    << "  "
170                    << "\"\"\"" << nombreFuncion << "\"\"\"\n"
171                    << "  " << variableFuncionPy << "\n"
172                    << "  " << retornoFuncionPy << std::endl;
173   flujoFuncionesHtml << "<tr>\n";
174   flujoFuncionesHtml << "<td bgcolor = #EFF5F5><center>" << tipoFuncionCpp
175                      << "</center></td>\n";
176   flujoFuncionesHtml << "<td bgcolor = #EFF5F5><center>" << nombreFuncion
177                      << "</center></td>\n";
178   flujoFuncionesHtml << "<td bgcolor = #EFF5F5><center>" << descripcionFuncion
179                      << "</center></td>\n";
180   }
```

Pasamos al ordenamiento de los flujos respectivos para las funciones. En este caso se evidencia en el código cuando la función se mueve por los archivos CPP (C++) PY (Python) y el archivo de texto HTML.

1.8

```
182   // MENU VARIABLES
183   void menuVariables(std::stringstream &flujoVariablesCpp,std::stringstream &flujoVariablesPy, std::stringstream &flujoVariablesHtml)
184   {
185     std::string tipoVariableCpp("");
186     std::string valorDefectoCpp("");
187     std::string valorDefectoPy("");
188     std::string nombreVariable("");
189     std::string variableSeleccionada("");
190     std::string descripcionVariable("");
191     int opcion(0);
192     int tipoVariableEleccion(0);
```

Seguimos con una función void tal cual como en la imagen 1.1 pero en este caso es para las variables que el usuario desea solicitar en el generador de código.

1.9

```
193     do
194     {
195       std::cout << "Asistente de generador de codigo\n";
196       std::cout << "Selecciona el Tipo de variable que deseas...\n";
197       std::cout << "1. char/string\n2. int\n3. double\n4. bool\n5. string\n";
198       std::cin >> opcion;
199       std::cout << "Ingresa el nombre que quieres para la variable...\n";
200       std::cin >> nombreVariable;
201
202       switch (opcion)
203       {
204       case 1:
205         tipoVariableCpp = "char ";
206         valorDefectoCpp = "'a';";
207         valorDefectoPy = "\" \"";
208         break;
209       case 2:
210         tipoVariableCpp = "int ";
211         valorDefectoCpp = "0;";
212         valorDefectoPy = "0";
213         break;
214       case 3:
215         tipoVariableCpp = "double ";
216         valorDefectoCpp = "0.0;";
217         valorDefectoPy = "0.0";
218
```

2.0

```
219         break;
220       case 4:
221         tipoVariableCpp = "bool ";
222         valorDefectoCpp = "false;";
223         valorDefectoPy = "False";
224         break;
225       case 5:
226         tipoVariableCpp = "std::string ";
227         valorDefectoCpp = "\" \";";
228         valorDefectoPy = "\" \"";
229
230         break;
231       }
232     } while (opcion < 1 || opcion > 5);
233     std::cout << "Ingresa la descripcion de la variable...\n";
234     std::cin.ignore();
235     getline(std::cin,descripcionVariable);
```

2.1

```
236     // pregunta al usuario si la variable es in/aux/out
237     do
238     {
239       std::cout << "Tipo variable seleccionada...\n";
240       std::cout << "1.in\n2.aux\n3.out\n";
241       std::cin >> tipoVariableEleccion;
242
243       switch (tipoVariableEleccion)
244       {
245       case 1:
246         variableSeleccionada = "in";
247         break;
248       case 2:
249         variableSeleccionada = "aux";
250         break;
251       case 3:
252         variableSeleccionada = "out";
253         break;
254       }
255
256     } while (tipoVariableEleccion < 1 || tipoVariableEleccion > 3);
257
```

En la imagen 1.9 hasta la imagen 2.1 se evidencia mediante un ciclo do while la variedad de opciones que el usuario tiene para escoger a la hora de solicitar una variable al generador de código, similar a lo explicado con anterioridad en las imágenes 1.2 hasta 1.6 salvando distancias en sus diferencias, tales como, al preguntarle al usuario si la variables es in / aux / out. también se tiene la sintaxis marcada para cada lenguaje solicitado (C++ y Python).

2.2

```
258     // ordenamiento de los flujos respectivos para las variables
259     flujoVariablesCpp << tipoVariableCpp << nombreVariable << " = "
260                       << valorDefectoCpp << std::endl;
261     flujoVariablesPy << "  " << nombreVariable << " = " << valorDefectoPy
262                      << std::endl;
263     flujoVariablesHtml << "<tr>\n";
264     flujoVariablesHtml << "<td bgcolor = #EFF5F5><center>" << variableSeleccionada
265                        << "</center></td>\n";
266     flujoVariablesHtml << "<td bgcolor = #EFF5F5><center>" << tipoVariableCpp
267                        << "</center></td>\n";
268     flujoVariablesHtml << "<td bgcolor = #EFF5F5><center>" << nombreVariable
269                        << "</center></td>\n";
270     flujoVariablesHtml << "<td bgcolor = #EFF5F5><center>" << descripcionVariable
271                        << "</center></td>\n";
272     flujoVariablesHtml << "</tr>\n";
273   }
```

Pasamos al ordenamiento de los flujos respectivos para las variables. En este caso se evidencia en el código cuando la función se mueve por los archivos CPP (C++) PY (Python) y el archivo de texto HTML.

2.3

```
275  void menuPrincipal()
276  {
277    std::stringstream flujoVariablesCpp;
278    std::stringstream flujoFuncionesCpp;
279    std::stringstream flujoVariablesPy;
280    std::stringstream flujoFuncionesPy;
281    std::stringstream flujoVariablesHtml;
282    std::stringstream flujoFuncionesHtml;
283    int opcion(0);
284    do
285    {
286      std::cout << "Asistente de generador de codigo\n";
287      std::cout << "Digita acorde a tu opcion\n";
288      std::cout << "1. Funcion\n2. Variables\n3. Terminar\n";
289      std::cin >> opcion;
290      if (opcion == 1)
291      {
292        menuFuncion(flujoFuncionesCpp, flujoFuncionesPy, flujoFuncionesHtml);
293      }
294      if (opcion == 2)
295      {
296        menuVariables(flujoVariablesCpp, flujoVariablesPy, flujoVariablesHtml);
297      }
298    } while (opcion != 3);
299    crearGenerados::generarCpp(flujoFuncionesCpp, flujoVariablesCpp);
300    crearGenerados::generarPy(flujoFuncionesPy, flujoVariablesPy);
301    crearGenerados::generarHtml(flujoFuncionesHtml, flujoVariablesHtml);
302  }
303
```

En la imagen 2.3 se tiene la función del menú principal que cuenta con los flujos tanto para funciones como variables y a su vez en los diferentes tipos de lenguaje. Cuenta con un ciclo de tipo do while donde le pregunta al usuario que desea crear (funciones o variables). Donde mediante condicionales invocamos las funciones de “menuFuncion” “menuVariables” definidas y explicadas con anterioridad. Luego se tiene la opción número tres (3) que da por terminado el procesos del generador de código y procede a generar los archivos en sus diferentes tipos ya predeterminados, los cuales son C++, Python y HTML.

2.4

```
304  bool llamadoSistema(std::string comando) {
305      bool resultado(false);
306      int llamado(0);
307      std::cout << "Ejecutando el comando: " << comando << std::endl;
308      llamado = system(comando.c_str());
309      if (llamado != 0) {
310          resultado = false;
311          std::cerr << "Fallo en ejecución de comando!" <<  std::endl;
312          return resultado;
313      }
314      return true;
315  }
```

2.5

```
317  bool compilarEjecutarCpp() {
318      bool resultado(false);
319      std::string compilar = "g++ -o generado.exe generado.cpp";
320      std::string ejecutar = "./generado.exe";
321
322      if (!llamadoSistema(compilar)) { return resultado;}
323      if (!llamadoSistema(ejecutar)) { return resultado;}
324
325      return true;
326  }
```

2.6

```
328  bool ejecutarPython() {
329      bool resultado(false);
330      std::string ejecutar = "python generado.py";
331      if (!llamadoSistema(ejecutar)) { return resultado;}
332      return true;
333  }
```

2.7

```
336  int flujoControl() {
337
338      if (!compilarEjecutarCpp()) { return -1; }
339      if (!ejecutarPython()) { return -1; }
340      return 0;
341  }
```

En la imagen 2.4 hasta la imagen 2.6 se tiene funciones de tipo bool que nos ayudan a compilar el código para cada lenguaje C++ y Python, recordemos que se usa una consola Linux para compilar lo cual hace necesario utilizar los comandos de Linux para compilar y ejecutar el código, razón por la cual se tienen dos (2) string "g++ -o generado.exe generado.cpp" para ejecutar y "./generado.exe para compilar esto en C++. Para Python se tiene un (1) string "Python generado.py" para ejecutar.

En la imagen 2.7 se invocan las dos funciones explicadas en el párrafo anterior mediante la función de "flujoControl"

2.8

```
344  int main ()
345  {
346    menuPrincipal();
347    if (flujoControl()!=0) {
348      std::cerr<<"\nLo sentimos, ha habido un fallo en el proceso de generación de código\n";
349      return -1;
350    }
351    std::cout<<"\nGracias por usar nuestro generador de código fuente multilenguaje y documentación\n";
352    return 0;
353  }
```

Por ultimo llegamos al "int main" donde invocamos lo declarado anteriormente y lanzamos dos mensajes dependiendo la situación, un mensaje de agradecimiento al usuario por usar nuestro software de generador de código en caso de que todo haya marchado bien, de lo contrario arrojará un mensaje de lo sentimos y dejamos saber que hubo problemas con el proceso de generación de código.