Описание языка C++ в AgavaSCADA/AgavaPLC

Данное описание содержит базовое руководство по созданию программ на языке C++ и их использовании в среде разработки Agava.

□

Содержание

1 Дополнительные документы

- Описание базовых классов AgavaSCADA/AgavaPLC.
- Объектная модель AgavaSCADA/AgavaPLC.

2 Основы синтаксиса

Все программы на языке C++ состоят из отдельных действий. Каждое действие заканчивается точкой с запятой:

```
int x;
x = 1;
int y = x + 2;
```

Для повышения удобства работы с кодом можно оставлять в нем комментарии, которые не используются при работе программы и предназначены для улучшения читаемости кода. Перед комментариями ставится «//»:

```
bool x = true;
// bool x = false;
// вторая строка не будет выполняться в программе
```

3 Переменные

Переменные – объекты, хранящиеся в памяти, над которыми и производятся действия в программе.

Для начала работы с переменной необходимо инициализировать ее, указав тип и имя, а затем, при необходимости, присвоить ей значение:

```
bool a = true;
int b = 1;
double c = 1.1;
string d = “строка”;

int f; // переменная инициализирована, значение не присвоено
f = 1; // присвоение значения
```

Ключевое слово auto при инициализации переменной может использоваться вместо типа переменной, для автоматического присвоения типа переменной исходя из инициализируемого значения.

```
auto a = true;
auto b = 1;
```

3.1 Строки

Строка – массив байтов или 16-битных слов. Обычно строки используются для хранения текста, но также могут хранить любые двоичные данные. Текст, хранящийся в строке, заключается в двойные кавычки.

При работе со строками доступны следующие функции:

Функция	Действие
uint length()	Возвращает длину строки
void resize(uint)	Устанавливает длину строки
bool isEmpty()	Возвращает истину, если строка пуста, т. е. длина равна нулю
string substr(uint start = 0, int count = -1)	Возвращает строку с содержимым, начинающимся с start, и количеством байтов, заданным параметром count. Аргументы по умолчанию вернут всю строку как новую строку
void insert(uint pos, const string &in other)	Вставляет другую строку string на позиции pos в исходной строке
void erase(uint pos, int count = -1)	Удаляет количество символов count из строки, начиная с позиции pos
int findFirst(const string &in str, uint start = 0)	Находит первое вхождение значения str в строке, начиная с символа под номером start. Если вхождения не найдено, будет возвращено отрицательное значение
int findFirstOf(const string &in str, uint start = 0)	Находит первое вхождение одного из символов в str в строке, начиная с символа под номером start. Если вхождения не найдено, будет возвращено отрицательное значение
int findLast(const string &in str, int start = -1)	Находит последнее вхождение значения str в строке
int findLastOf(const string &in str, int start = -1)	Находит последнее вхождение одного из символов в str в строке, начиная с символа под номером start. Если вхождения не найдено, будет возвращено отрицательное значение
string formatInt(int64 val, const string &in options = "", uint width=0)	Преобразование int в строку
string formatUInt(uint64 val, const string &in options = "", uint width=0)	Преобразование uint в строку
string formatFloat(double val, const string &in options="", uint width=0, uint precision=0)	Преобразование float и double в строку
int64 parseInt(const string &in str, uint base=10, uint &out byteCount=0)	Преобразование строки в int64
uint64 parseUInt(const string &in str, uint base=10, uint &out byteCount=0)	Преобразование строки в uint64
double parseFloat(const string &in str, uint &out byteCount=0)	Преобразование строки в double

Пример работы с функциями для строк:

```
string str = “ABCDEF”;          // инициализирована строка str

int a = str.length();           // a равно 6
int b = str.findFirst(“CD”, 0); // b равно 2

string str1 = str.erase(1, 3);  // str1 равно “AEF”
```

3.2 Массивы

Массив — свокупный тип данных, который позволяет получить доступ ко всем переменным одного и того же типа данных через использование одного идентификатора. Используется в том случае, если вам необходимо сгруппировать несколько элементов одного типа:

```
array <int> a = {2, 5, 10, 15, 20};  // инициализирован массив a с пятью элементами
int x = a [0];       // x становится равен элементу массива a под номером 0 (2)
int y = a [4];       // y становится равен элементу массива a под номером 4 (20)
arr [3] = 11;        // элемент массива a под номером 3 становится равен 11
int z = arr [3];      // z становится равен элементу массива a под номером 3 (11)
```

Важно помнить, что нумерация членов массива начинается с 0.

При работе с массивами доступны следующие функции:

Функция	Действие
length()	Возвращает длину массива
resize(uint)	Устанавливает новую длину массива
reverse()	Меняет порядок элементов в массиве на обратный
insertAt(uint index, const T& in value)	Вставляет новый элемент в массив по указанному индексу
insertLast(const T& in)	Добавляет элемент в конец массива
removeAt(uint index)	Удаляет элемент по указанному индексу
removeLast()	Удаляет последний элемент массива
removeRange(uint start, uint count)	Удаляет количество элементов count, начиная с элемента номер start
sortAsc ()	Сортирует элементы в массиве в порядке возрастания
sortDesc ()	Сортирует элементы в массиве в порядке убывания
find (const T &in)	Возвращает индекс первого элемента, который имеет то же значение, что и желаемое. Если совпадений не найдено, возвращает отрицательное значение

Пример работы с функциями для массивов:

```
int main()
{
  array<int> arr = {1,2,3}; // 1,2,3
  arr.insertLast(0);        // 1,2,3,0
  arr.insertAt(2,4);        // 1,2,4,3,0
  arr.removeAt(1);          // 1,4,3,0
  arr.sortAsc();            // 0,1,3,4
  int sum = 0;

  for( uint n = 0; n < arr.length(); n++ )
    sum += arr[n];
```

```
   return sum;
}
```

3.3 Перечисления

Перечисление — это тип данных, где любое значение определяется как символьная константа. Оно позволяет представить семейства целочисленных констант (например, коды ошибок) в текстовом виде вместо числового. Использование перечислений помогает улучшить читаемость кода, поскольку не нужно искать в руководстве, что означает числовое значение.

Значения перечислений объявляются путем их объявления в операторе перечисления. Если для константы перечисления не указано конкретное значение, она будет принимать значение предыдущей константы плюс 1. Первая константа получит значение 0, если не указано иное.

```
enum Errors
{
    NoError,         // = 0
    ReadError = 2,    // = 2
    WriteError,       // = 3
}

// теперь к константам перечисления можно обращаться через текстовое значение, например
// int a = NoError;
```

3.4 Типы переменных

Доступны логические, целочисленные, строковые переменные и переменные с плавающей запятой.

Логические переменные

Тип	Мин. значение	Макс. значение
bool	false (0)	true (1)

Целочисленные переменные

Тип	Мин. значение	Макс. значение
int8	- 128	127
int16	- 32 768	32 767
int	- 2 147 483 648	2 147 483 647
int64	- 9 223 372 036 854 775 808	9 223 372 036 854 775 807
uint8	0	255
uint16	0	65 535
uint	0	4 294 967 295
uint64	0	18 446 744 073 709 551 615

Переменные с плавающей запятой

Тип	Диапазон значений	Наименьшее значение
float	± 3.402823466 e+38	1.175494351 e-38
double	± 1.79769313486231 e+308	2.22507385850720 e-308

4 Операторы

Доступны операторы сравнения, логические, побитовые, арифметические, приращений и индексации.

4.1 Операторы сравнения

Оператор	Символ	Пример	Операция
Равно	==	x == y	true, если x равно y
Не равно	!=	x != y	true, если x не равно y
Меньше	<	x < y	true, если x меньше y
Больше	>	x > y	true, если x больше y
Меньше или равно	<=	x <= y	true, если x меньше или равно y
Больше или равно	>=	x >= y	true, если x больше или равно y

4.2 Логические операторы

Оператор	Символ	Пример	Операция
Логическое НЕ	!	!x	true, если x – false и false, если x – true
Логическое И	&&	x && y	true, если x и y – true, в противном случае – false
Логическое ИЛИ	||	x || y	true, если x или y – true, в противном случае – false
Логическое ИСКЛЮЧАЮЩЕЕ ИЛИ (XOR)	^	x ^ y	true, если x или y – true (но не одновременно), в противном случае – false

4.3 Побитовые операторы

Оператор	Символ	Пример	Операция
Побитовое И	&	x & y	Каждый бит в x И каждый соответствующий ему бит в y
Побитовое ИЛИ	|	x | y	Каждый бит в x ИЛИ каждый соответствующий ему бит в y
Побитовое ИСКЛЮЧАЮЩЕЕ ИЛИ (XOR)	^	x ^ y	Каждый бит в x XOR с каждым соответствующим ему битом в y
Побитовый сдвиг влево	<<	x << y	Все биты в x смещаются влево на y бит
Побитовый сдвиг вправо	>>	x >> y	Все биты в x смещаются вправо на y бит

4.4 Арифметические операторы

Оператор	Символ	Пример	Операция
Сложение	+	x + y	Складываем x и y
Вычитание	-	x – y	Вычитаем y из x
Умножение	*	x * y	Умножаем x на y
Деление	/	x / y	Делим x на y
Присваивание	=	x = y	Присваиваем значение y переменной x
Сложение с присваиванием	+=	x += y	Добавляем y к x
Вычитание с присваиванием	-=	x -= y	Вычитаем y из x
Умножение с присваиванием	*=	x *= y	Умножаем x на y
Деление с присваиванием	/=	x /= y	Делим x на y

4.5 Операторы приращений

Операция инкремента увеличивает значение на 1, декремента – уменьшает на 1.

Оператор	Символ	Пример	Операция
Преинкремент	++	++x	Инкремент x, затем вычисление x
Предекремент	--	--x	Декремент x, затем вычисление x
Постинкремент	++	x++	Вычисление x, затем инкремент x
Постдекремент	--	x--	Вычисление x, затем декремент x

4.6 Оператор индексации

Используется для доступа к элементу.

Оператор	Символ	Пример	Операция
Индексация	[ ]	arr [0]	Обращение к нулевому элементу массива arr

5 Блоки выражений и область видимости переменных

Блоки выражений используются в условиях, циклах и функциях. При использовании вложенных блоков, блок, который содержит внутри себя другой блок, называется внешним блоком, а тот, который содержится внутри этого блока — внутренним / вложенным блок. Каждый блок заключается в фигурные скобки.

Внешние блоки не имеют доступа к переменным внутренних блоков.

```
 {
     int a;
     float b;
     {
       float a = 1.0; // Отменить объявление внешней переменной
                      // но только в пределах внутреннего блока
       b = a;         // переменные из внешних блоков все еще доступны
       int c;
     }

     // a снова становится целочисленной переменной, b теперь равна 1.0, c здесь недоступна
   }
```

6 Условия

6.1 Операторы if/else

Позволяют выбрать выполнение нужных действий, в зависимости от выбранного условия, указанного в скобках:

```
 if (x == 1)
 {
   // эти действия выполнятся, если x равно 1
 }
```

```
else
{
  // эти действия выполнятся, если x не равно 1
}
```

6.2 Условное выражение (тернарный оператор)

Более короткая запись оператора выбора if/else:

```
x = 1 ? y = true : y = false;       // если x равно 1, тогда y равен true, иначе y равен false
```

6.3 Оператор switch

Является альтернативой if/else для тех случаев, когда необходимо сделать множественный выбор:

```
switch(x)
{
  case 0:
    // эти действия выполнятся, если x равно 0
    break;  // каждый case всегда должен заканчивать оператором break
  case 1:
    // эти действия выполнятся, если x равно 1
    break;
  case 2:
    // эти действия выполнятся, если x равно 2
    break;
  default:
    // Эти действия выполнятся, если x не равен ни одному из предыдущих значений case
  }
```

7 Циклы

7.1 Цикл while

Цикл выполняется, пока условие в скобках истинно:

```
x = 0;

while (x < 5)
{
    // это действие будет выполняться, пока x меньше 5
    x++;  // при каждом выполнении цикла увеличиваем x на 1
}
```

7.2 Цикл do while

Цикл похож на while, но, в отличии от него, цикл в любом случае выполнится хотя бы один раз (или больше, если при последующих проверках условие будет истинно):

```
x = 10;
```

```
do
{
   // это действие выполнится один раз, так как условие изначально ложно
}
while (x < 5);
```

7.3 Цикл for

Цикл for обычно используется как счетчик и хорошо подходит, когда известно необходимое количество итераций. В скобках через точку с запятой указывается переменная, условие, инкремент / декремент для переменной:

```
for ( int x = 0; x < 5; x ++ )
{
    // при каждом выполнении этих действий переменная x будет увеличивать на 1
    // действия будут выполняться пока x меньше 5
}
```

8 Функции

Функция — это последовательность действия для выполнения определенного задания. Объявление функции похоже на объявление переменной: необходимо указать тип возвращаемого значения (либо void, если функция ничего не возвращает) и название функции. После этого ставятся двойные скобки, в которых находятся аргументы функции (либо пустые скобки, если аргументы отсутствуют). Действия, выполняемые функцией, заключаются в фигурные скобки.

Любой код в операции «Скрипт C++» должен находиться в функции. При использовании нескольких функций в скрипте будет выполняться только первая, остальные должны быть вызваны из нее:

```
void main ()
{
  int x = 1;
  function ();  // вызов функции, программа переходит в void function()
  SetNodeValueAsInt("/Конфигурация/Станция/Сигналы/Constant2", x);
}

void function ()
{
  SetNodeValueAsInt("/Конфигурация/Станция/Сигналы/Constant2", 5);
  // теперь программа возвращается обратно в void main()
}
```

После выполнения данного скрипта значение сигнала Constant1 станет равно 5, а Constant2 равно 1.

Если переменная была объявлена в функции, то она не будет доступна за ее пределами.

8.1 Функции с параметрами

Параметр функции — это переменная, которая используется в функции, и значение

которой предоставляет вызывающий функцию объект.

Параметры указываются при объявлении функции в круглых скобках:

```
void summ (int x, int y)  // в функцию передаются параметры x = 1 и y = 2
{
  int z = x + y;  // z равно 3
}
```

8.2 Возврат значения

Для передачи значения из скрипта можно указать необходимое значение в действии return, указав тип функции такой же, как у возвращаемого значения:

```
bool R_Output (bool x, bool y)
{
  bool z = false;
  z = x && y;

  return z; // z передается на выход скрипта
}
```

9 Специализированные функции

Специализированные функции среды разработки, доступные для использования в скриптах, приведены в таблицах ниже.

9.1 Функции для работы с узлами

Определение функции	Описание
float GetNodeValueAsFloat(string strNodePath)	Получить значение узла в виде float по заданному пути
void SetNodeValueAsFloat(string strNodePath, float fValue)	Установить значение узла в float по заданному пути
int GetNodeValueAsInt(string strNodePath)	Получить значение узла в виде int по заданному пути
void SetNodeValueAsInt(string strNodePath, int iValue)	Установить значение узла в int по заданному пути
string GetNodeValueAsString(string strNodePath)	Получить значение узла в виде string по заданному пути
void SetNodeValueAsString(string strNodePath, string strValue)	Установить значение узла в виде string по заданному пути
bool GetNodeValueAsBool(string strNodePath)	Получить значение узла в виде bool по заданному пути
void SetNodeValueAsBool(string strNodePath, bool bValue)	Установить значение узла в виде bool по заданному пути
bool NodeValueIsError(string strNodePath)	Проверка значения узла на ошибку
void StartNode(string strNodePath)	Запуск узла по заданному пути
void StopNode(string strNodePath)	Остановка узла по заданному пути

9.2 Функции для работы с окнами

Определение функции	Описание
void CloseWindow(string strNodePath, int iDisplay)	Закрыть окно по заданному пути на дисплее с номером iDisplay
void ShowWindow(string strNodePath, int iDisplay)	Отобразить окно по заданному пути на дисплее с номером iDisplay. Координаты - абсолютные, установленные в свойствах окна.
int LoadComposition(string strWindowPath, string strCompositionPath, int iDisplay)	Загрузить композицию strCompositionPath в окно strWindowPath на дисплее с номером iDisplay AS 1.6.25+

9.3 Функции для работы с временем

Определение функции	Описание
int GetCurrentTime()	Получение текущего времени в UNIX формате (количество секунд с 01 января 1970 года)
uint64 GetCurrentTimeMs()	Получение текущего времени в UNIX формате (количество миллисекунд с 01 января 1970 года) (AS 1.2.55+, AS 1.5.27+)
int GetLocalTime()	Получение текущего времени в UNIX формате (количество секунд с 01 января 1970 года) с учетом часового пояса
int GetHours(int iTime)	Получение текущего часа из времени в формате UNIX
int GetMinutes(int iTime)	Получение текущих минут из времени в формате UNIX
int GetSeconds(int iTime)	Получение текущих секунд из времени в формате UNIX

9.4 Функции для работы с событиями

```
int StoreMessage(int iMessageLevel, string strMessage)
```

Назначение	Генерация информационного события (типа "сообщение"). Событие помещается в группу "Системные".
Возвращаемое значение	0 - успешно, иначе код ошибки
Аргументы	int iMessageLevel - уровень сообщения (1-наивысший): 1=FATAL, 2=ERROR, 3=WARNING, 4=NOTICE, 5=INFO, 6=TRACE, 7=DEBUG string strMessage - текст сообщения
Примечания	функция доступна с версии 1.2.34

```
int StoreMessage(int iMessageLevel, string strMessage, string strGroup)
```

Назначение	Генерация информационного события (типа "сообщение"). Событие помещается в указанную при вызове группу.
Возвращаемое значение	0 - успешно, иначе код ошибки
Аргументы	int iMessageLevel - уровень сообщения (1-наивысший): 1=FATAL, 2=ERROR, 3=WARNING, 4=NOTICE, 5=INFO, 6=TRACE, 7=DEBUG string strMessage - текст сообщения string strGroup - имя группы, в которую должно помещаться событие.
Примечания	функция доступна с версии 1.6.20

9.5 Функции для работы со строками

```
string EncodeMD5(string strText)
```

Назначение	Кодирование строки в MD5 хэш
Возвращаемое значение	MD5 хэш параметра
Аргументы	string strText - исходная строка
Примечания	функция доступна с версии 1.2.34

9.6 Другие функции

Проверка на nan

```
bool isnan(double)
```

10 Класс file

Класс доступен начиная с версии 1.4.5 AgavaSCADA/AgavaPLC.

10.1 Методы

```
int open(const string& filename, const string& mode)
```

Назначение	Открытие файла
Возвращаемое значение	Результат выполнения операции: 0 - успешно, -1 - ошибка.
Аргументы	const string& filename - путь до файла const string& mode - режим открытия файла.

```
int close()
```

Назначение	Закрытие файла
Возвращаемое значение	Результат выполнения операции. 0 - успешно, -1 - ошибка.
Аргументы	-

```
int getSize() const
```

Назначение	Получение размера файла
Возвращаемое значение	Размер файла в байтах
Аргументы	-

```
bool isEndOfFile() const
```

Назначение	Получение признака конца файла
Возвращаемое значение	true - обнаружен конец файла.
Аргументы	-

```
string readString(uint length)
```

Назначение	Чтение строки
Возвращаемое значение	Считанная строка
Аргументы	uint length - длина считываемой строки

```
string readLine()
```

Назначение	Чтение строки
Возвращаемое значение	Считанная строка
Аргументы	-
Примечания	Считывание производится до тех пор, пока не будет обнаружен символ '\n' или конец файла

```
int64 readInt(uint)
```

Назначение: чтение целого числа со знаком.

Возвращаемое значение: считанное значение.

Аргументы: отсутствуют.

Примечания: отсутствуют.

```
uint64 readUInt(uint)
```

Назначение: чтение целого беззнакового числа.

Возвращаемое значение: считанное значение.

Аргументы: отсутствуют.

Примечания: отсутствуют.

```
float readFloat()
```

Назначение: чтение целого вещественного числа одинарной точности.

Возвращаемое значение: считанное значение.

Аргументы: отсутствуют.

Примечания: отсутствуют.

double readDouble()

int writeString(const string &in)

int writeInt(int64, uint)

int writeUInt(uint64, uint)

int writeFloat(float)

int writeDouble(double)

int getPos() const

int setPos(int)

int movePos(int)

10.2 Поля

bool mostSignificantByteFirst.

11 Класс datetime

Класс доступен начиная с версии 1.6+.

11.1 Конструкторы

11.1.1 Конструктор по умолчанию

```
datetime dt();
```

Создает объект с текущей датой и временем.

11.1.2 Конструктор копирования

```
datetime dt(other);
```

Создает копию существующего объекта datetime.

11.1.3 Конструктор с параметрами

```
datetime dt(year, month, day, hour = 0, minute = 0, second = 0);
```

Создает объект с указанной датой и временем. **Примеры:**

```
// Только дата
datetime newYear = datetime(2024, 1, 1);

// Дата и время
datetime meeting = datetime(2024, 12, 25, 14, 30, 0);

// Полная спецификация
datetime exact = datetime(2024, 6, 15, 9, 45, 30);
```

11.2 Методы

`uint get_year() const`

Возвращает год (4 цифры).

`uint get_month() const`

Возвращает месяц (1-12).

`uint get_day() const`

Возвращает день месяца (1-31).

`uint get_hour() const`

Возвращает часы (0-23).

`uint get_minute() const`

Возвращает минуты (0-59).

`uint get_second() const`

Возвращает секунды (0-59).

Пример:

```
datetime dt = datetime(2024, 12, 25, 14, 30, 45);

uint year = dt.year;    // 2024
uint month = dt.month;  // 12
uint day = dt.day;      // 25
uint hour = dt.hour;    // 14
uint minute = dt.minute; // 30
uint second = dt.second; // 45
```

`bool setDate(year, month, day)`

Устанавливает дату. Возвращает `true` при успехе, `false` при неверной дате.

Пример:

```
datetime dt;
bool success = dt.setDate(2024, 2, 29); // true (високосный год)
bool invalid = dt.setDate(2023, 2, 29); // false (не високосный)
```

`bool setTime(hour, minute, second)`

Устанавливает время. Возвращает `true` при успехе, `false` при неверном времени.

Пример:

cpp

```
datetime dt;
bool success = dt.setTime(23, 59, 59); // true
bool invalid = dt.setTime(25, 0, 0);   // false (часы > 23)
```

11.3 Арифметические операции

Вычитание дат

```
int64 difference = dt1 - dt2;
```

Возвращает разницу в секундах.

```
datetime start = datetime(2024, 1, 1, 0, 0, 0);
datetime end = datetime(2024, 1, 2, 0, 0, 0);
int64 diff = end - start; // 86400 секунд (1 день)
```

Сложение с секундами

```
datetime result = dt + seconds;
datetime result = seconds + dt; // обратный порядок
dt += seconds;
```

Пример:

```
datetime dt = datetime(2024, 1, 1, 10, 0, 0);

// Добавить 2 часа
datetime later = dt + 7200;
dt += 7200;

// Обратный порядок
datetime same = 7200 + dt;
```

Вычитание секунд

```
datetime result = dt - seconds;
datetime result = seconds - dt; // обратный порядок
dt -= seconds;
```

Пример:

```
datetime dt = datetime(2024, 1, 1, 10, 0, 0);

// Вычесть 30 минут
datetime earlier = dt - 1800;
dt -= 1800;
```

11.4 Операции сравнения

`bool opEquals(other)`

Проверка равенства дат.

`bool opLess(other)`

Проверка что дата раньше.

`int opCmp(other)`

Сравнение с возвратом -1 (меньше), 0 (равно), 1 (больше).

Пример:

```
datetime dt1 = datetime(2024, 1, 1);
datetime dt2 = datetime(2024, 1, 2);

if (dt1 < dt2) {
    // dt1 раньше dt2
}

if (dt1 == dt2) {
    // даты равны
}

int result = dt1.opCmp(dt2); // -1
```

11.5 Практические примеры

Пример 1: Таймер выполнения

```
datetime startTime;

// Выполняем некоторую операцию
for (int i = 0; i < 1000000; i++)
{
    // какая-то работа
}

datetime endTime;
int64 elapsed = endTime - startTime;
print("Операция заняла: " + elapsed + " секунд");
```

Пример 2: Расчет дат событий

```
// Дата начала проекта
datetime projectStart = datetime(2024, 1, 15);

// Напоминание за 3 дня до дедлайна
datetime deadline = datetime(2024, 3, 1);
datetime reminder = deadline - 3 * 86400; // 3 дня в секундах

// Продление срока на 2 недели
datetime extendedDeadline = deadline + 14 * 86400;
```

Пример 3: Проверка рабочего времени

```
datetime currentTime;

// Создаем время начала и конца рабочего дня
datetime workStart = datetime(currentTime.year, currentTime.month,
                              currentTime.day, 9, 0, 0);
datetime workEnd = workStart + 8 * 3600; // 8 часов

if (currentTime >= workStart && currentTime <= workEnd)
{
    print("Рабочее время");
```

```
}
else
{
    print("Вне рабочего времени");
}
```

Пример 4: Валидация даты

```
bool createAppointment(uint year, uint month, uint day, uint hour)
{
    datetime appointment;
    // Проверяем корректность даты
    if (!appointment.setDate(year, month, day)) {
        print("Неверная дата!");
        return false;
    }
    // Проверяем корректность времени
    if (!appointment.setTime(hour, 0, 0)) {
        print("Неверное время!");
        return false;
    }
    // Проверяем что дата не в прошлом
    datetime now;
    if (appointment < now) {
        print("Дата не может быть в прошлом!");
        return false;
    }
    print("Встреча создана: " + appointment.year + "-" + appointment.month + "-" + appointment.day);
    return true;
}
```

Пример 5: Разбивка интервала времени

```
datetime start = datetime(2024, 1, 1, 0, 0, 0);
datetime end = start + 365 * 86400; // +1 год

int64 totalSeconds = end - start;
int64 days = totalSeconds / 86400;
int64 hours = (totalSeconds % 86400) / 3600;
int64 minutes = (totalSeconds % 3600) / 60;
int64 seconds = totalSeconds % 60;

print("Интервал: " + days + " дней, " + hours + " часов, " + minutes + " минут, " + seconds + "
секунд");
```

11.6 Пример использования скрипта для решения задачи

Приведенный ниже пример демонстрирует один из вариантов решения реальной практической задачи. Для реализации необходимого алгоритма работы задействованы узлы, взаимодействующие с входами / выходами контроллера или хранящие значения, и скрипт C++.

На вход скрипта подаются значения некоего сигнала (Signal) и уставки (Constant). В том случае, если сигнал больше уставки, замыкается релейный выход (out0). Соотношение между сигналом и уставкой переводится в проценты и записывается в переменную Percent. Если это соотношение становится меньше определенного значения, то соответствующее сообщение записывается в переменную Alert.

Текст скрипта:

```
bool main (float signal_val, float constant_val)  // функция получает значения сигнала и уставки
{
  float ratio;             // переменная, в которой будет храниться соотношение сигнала и уставки
  ratio = signal_val / constant_val; // вычисление соотношения

  percent_write(ratio);             // вызов функции записи в узлы, с передачей в нее соотношения

  if ( signal_val > constant_val)
  {
     return true;                   // если сигнал больше уставки, на выход скрипта приходит true
  }
  else
  {
     return false;               // если сигнал не больше уставки, на выход скрипта приходит false
  }
}

void percent_write (double ratio) // функция записи в узлы
{
  string message = "Значение в пределах нормы"; // текстовое сообщение по умолчанию
  double percent = ratio * 100;                 // вычисляется процентное соотношение

  if ( percent < 50 )
  {
    message = "Значение ниже нормы!";           // если процентное соотношение меньше 50
                                                // то текст сообщения меняется

  }

  // запись текстового сообщения в нужный узел
  SetNodeValueAsString("/Конфигурация/Станция/Сигналы/Alert", message);

  // запись процентного соотношения в нужный узел
  SetNodeValueAsFloat("/Конфигурация/Станция/Сигналы/Percent", percent);

}
```

12 Приоритет операций

В выражениях первым всегда вычисляется оператор с наивысшим приоритетом. Унарные операторы имеют более высокий приоритет, чем другие операторы. Постоператоры имеют более высокий приоритет, чем преоператоры.

В этой таблице показаны доступные унарные операторы, в порядке убывания приоритета.

Символ	Операция
::	Оператор разрешения области видимости
[]	Оператор индексации
++ --	Постинкремент и декремент
.	Доступ
++ --	Преинкремент и декремент
!	Логическое НЕ
+ -	Унарный положительный и отрицательный
~	Побитовое дополнение
@	Ссылка

Эта таблица показывает приоритет бинарных и тернарных операторов в порядке убывания.

Символ	Операция
**	Экспонента
* / %	Умножить, разделить, по модулю
+ -	Сложить и вычесть
<< >> >>>	Сдвиг влево, сдвиг вправо и арифметический сдвиг вправо
&	Битовое И
^	Битовый XOR
|	Битовое ИЛИ
<= < >= >	Сравнение
== != is !is xor ^^	Равенство, идентичность и логическое исключающее ИЛИ
and &&	Логическое И
or ||	Логическое ИЛИ
?:	Условие
= += -= *= /= %= **= &= |= ^= <<= >>= >>>=	Присваивание и составные присваивания

13 Зарезервированные ключевые слова

Это ключевые слова, которые зарезервированы языком. Они не могут использоваться какими-либо идентификаторами, определенными скриптом.

and	double	if	not	this
abstract	else	import	null	true
auto	enum	in	or	try
bool	explicit	inout	out	typedef
break	external	int	override	uint
case	false	interface	private	uint8
cast	final	int8	property	uint16
catch	float	int16	protected	uint32
class	for	int32	return	uint64
const	from	int64	set	void
continue	funcdef	is	shared	while
default	function	mixin	super	xor
do	get	namespace	switch	

14 Ссылки

Источник — https://docs.kb-agava.ru/index.php?title=Описание_языка_C%2B%2B_в_AgavaSCADA/AgavaPLC&oldid=3306