PERFECTION IN AUTOMATION
A MEMBER OF THE ABB GROUP

TM248
ANSI C

버전 정보

버전	날짜	수정내역	번역	검수
1.0	2017.10.12	TM248TRE.25-ENG 3.2, 3.5 그림 변경 4.5 설정제거 – (wlway overwrite datatype information) 5.1 아쿠아리즘 그림 변경 설명 추가, Bottle counter 그림 변경 설명 추가		

Table 1: Versions

선행 및 필요 조건

교육 자료	TM210 – The Basics of Automation Studio TM211 – Automation Studio Online Communication TM213 – Automation Runtime TM223 – Automation Studio Diagnostics
소프트웨어	Automation Studio
하드웨어	–

II 목차

1 Introduction 1
1.1 Objectvies 2

2 THE CHARACTERISTICS OF ANSI C 3
2.1 General 3
2.2 History 3
2.3 Features 3
2.4 Possibilities 3

3 ANSI C IN AUTOMATION STUDIO 4
3.1 Elements of a C task 4
3.2 Creating a C task 5
3.3 C Language extensions 5
3.3.1 INIT 5
3.3.2 CYCLIC 6
3.3.3 EXIT 6
3.4 Header file 7
3.4.1 Rules for the “include” statement 8
3.5 Compiler settings 9

4 Variables 10
4.1 Data types 10
4.2 Scope 11
4.3 Remanent variables 12
4.4 Constants 12
4.5 Structures 12

5 Function Calls 15
5.1 Definition of a function 15

6 B&R Function Blocks 19
6.1 General information 19
6.2 Calling function blocks 19

7 COMPILER 22

8 Summary 23

9 Appendix 24
9.1 Operators 24
9.1.1 산술 연산자(Arithmetic operators) 24
9.1.2 비교 연산자(Comparison operators) 24
9.1.3 비트 연산자(Bitwise operators) 24
9.1.4 논리 연산자(Logic operators) 25
9.2 Commands 25
9.2.1 IF-Else 25
9.2.2 Else-IF 25
9.2.3 Switch 26
9.2.4 While loops 27
9.2.5 For loops 27

9.2.6 Do-While 27
9.2.7 Break 28
9.2.8 Continue 28
9.2.9 Goto 28

1 Introduction

많은 개발자들이 선호하는 ANSI C 는 광범위하게 사용되는 프로그래밍 언어입니다. 이 언어에서 제공되는 광범위한 가능성들은 PC-based 와 embedded system 에서 사용되는 중요한 이유입니다.

Figure 1 ANSI C

다음의 장에서는 Automation Studio 에서 B&R 의 확장과 C-task 들의 통합에 대한 정보를 제공할 것입니다.

당신은 들어가기에 앞서 프로그래밍 언어인 ANSI C 에 익숙해져 있어야만 합니다.

1.1 Objectvies

교육 자료에서, 참가자들은 Automation Studio 에서 어떻게 ANSI C 를 사용하는지 배우게 될 것입니다.

여기서는 프로그래밍 언어의 명령 구성들과 문법들에 관한 사항들은 설명하지 않을 것입니다. 그러나 이러한 주제는 많은 참고 서적들을 이용하실 수 있습니다.

참가자들은 B&R 의 확장에 관한 개요를 알게 될 것이며, 이것은 ANSI C 에서 그들의 control task 을 프로그램 할 수 있도록 할 것입니다.

기본적인 예제들은 어떻게 C-task 들을 생성하는지 보여주기 위해서 사용될 것입니다.

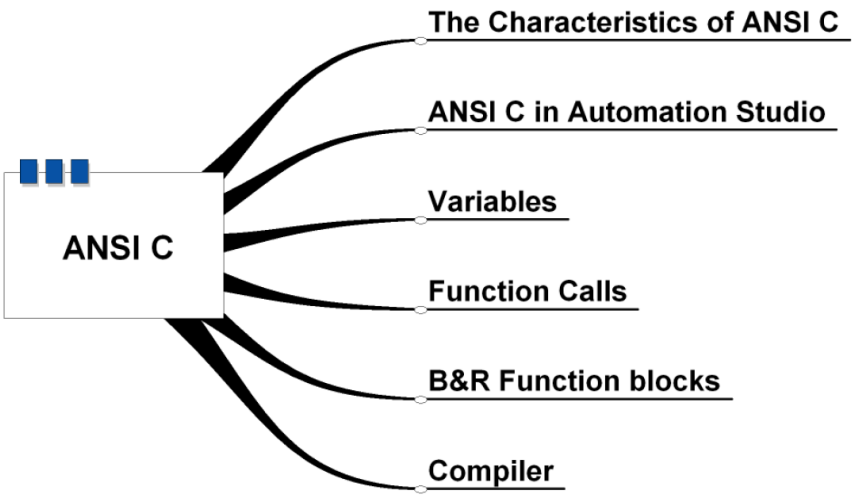


Figure 2 Overview

2 THE CHARACTERISTICS OF ANSI C

2.1 General

ANSI C 는 기본 형태의 일반적인 어플리케이션, 최신 명령어와 데이터 구조 그리고 개발자의 넓은 선택을 위한 프로그래밍 언어입니다.

그렇지만, 일반적인 유용성은 많은 task 들에서 C 를 편리하고 효과적으로 만듭니다.

2.2 History

컴퓨터 세계의 혁명은 Kernigham 과 Ritchie 이 1978 년에 집필한 “Programming in C" 이래로 시작되었습니다.

증가하는 C 의 인기와 시간이 지남에 따라 변화된 언어는 더욱 정교하고 현대적인 언어에 대한 정의가 필요하게 되었습니다.

1983 년 American National Standards Institute (ANSI)는 C 언어의 시대정신에 위배되지 않고 기계독립적인 개발을 위해서 위원회를 결성하였습니다.

그 결과가 C 에 대한 ANSI 표준입니다.

2.3 Features

ANSI C 는 다음과 같은 특징들을 지닙니다.

- 범용 프로그래밍언어(Universal programming language)
- 어떤 하드웨어에도 제한이 없음(Not bound to any hardware)
- 최신 명령 구조(if, while, switch 등) (Modern command structures)
- 연산자의 폭넓은 선택(Large selection of operators)
- 효과적인 코드(Effective code)

2.4 Possibilities

Automation Studio 는 다음의 기능들을 지원합니다.

- 디지털/아날로그 입출력(Digital and analog inputs and outputs)
- 논리 연산자(Logic operations)
- 논리 비교 표현(Logic comparison expressions)
- 산술 연산(Arithmetic operations)
- 결정(Decisions)
- 스텝 시퀀스(Step sequencers)
- 루프(Loops)
- 동적 변수의 사용(Use of dynamic variables)
- 진단 도구(Diagnostic tools)
- 표준 ANSI C 라이브러리 (math.h,string.h 등) (Standard ANSI C libraries)
- B&R 라이브러리(libraries)

3 ANSI C IN AUTOMATION STUDIO

3.1 Elements of a C task

C task 는 적어도 하나의 *.c 파일로 존재합니다. 추가적인 소스파일(*.c)과 헤더파일(*.h) 또한 추가될 수 있습니다.

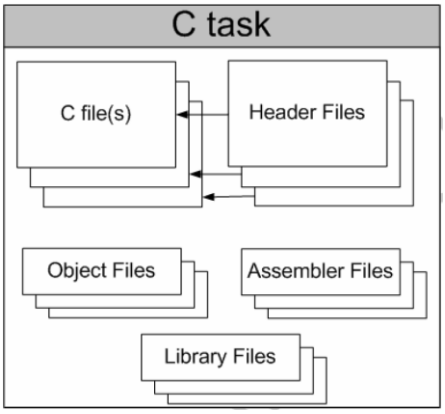


Figure 3 Elements of a C task

File type	Update	Description
Source file	*.c	변수 정의, 함수들
Header file	*.h	함수들과 변수정의의 프로토타입은 다른 C task 에서 사용됩니다. 이러한 헤더파일들은 C source 에서 반드시 #include 명령어를 사용하여 구현되어야 합니다. (예: #include<plctypes.h>)
Libraries	*.a	대상 라이브러리
Assembler source	*.s	어셈블러 소스 텍스트 (Assembler source text)
Object files	*.o	대상 파일

3.2 Creating a C task

새로운 ANSI C program (C language)를 선택하십시요. 다음 윈도우는 task 가 할당된 후의 모습입니다.

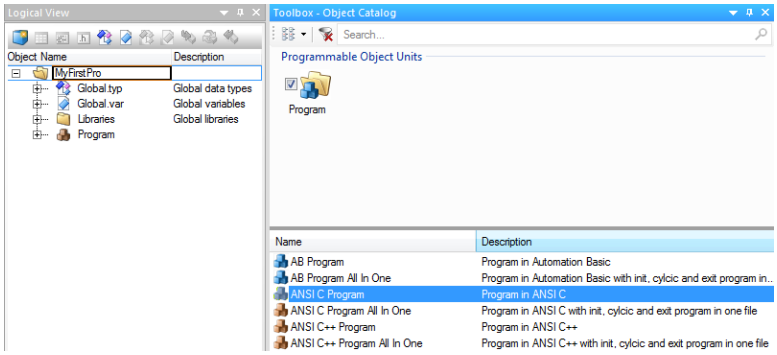


Figure 4 Creating a C task

3.3 C Language extensions

ANSI C 언어 확장자는 task class system 의 cyclic processing 과 init subprogram 과 같은 구현 함수들에 대하여 이루어집니다.

대상 소프트웨어에서 _INIT (Init subprogram)과 _CYCLIC(cyclic 대상이되는 Cyclic# 과 Timer#) 그리고 _EXIT 메크로는 함수 정의시에 사용될 수 있습니다. main() 함수는 없으며, 이것은 C 의 공통 사항입니다.

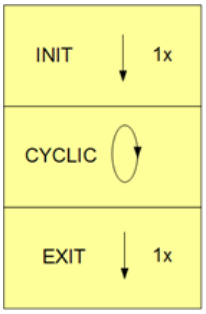


Figure 5 Functions in a C file

3.3.1 INIT

Init subprogram 은 각 task 에 존재하는 것이 가능합니다. 이 프로그램 섹션은 warm restart 또는 task transferring (Overoad mode 에서만)후에만 실행됩니다.

이것은 C 에서 _INIT 함수를 사용하여 구현되었습니다.

Example: Init function

```
void _INIT InitPart(void)
{
  /* Init subroutine*/
}
```

Figure 6 Definition of the _INIT function

Note:
InitPart 라는 함수의 이름은 아무 의미가 없으며 자유롭게 선택 할 수 있습니다.

3.3.2 CYCLIC

_CYCLIC 함수 안의 코드는 주기적으로 실행됩니다. (task class cycle 에 따라서)

Example: _CYCLIC function

```
void _CYCLIC CyclicPart(void)
{
  /* Cyclic Part of the Task */
}
```

Figure 7 Definition of the _CYCLIC function

3.3.3 EXIT

_EXIT 함수는 task 가 uninstall 되거나, 대체되거나 제거될 때만 실행됩니다. 예를 들어 이 함수는 통신을 위해서 파라미터를 정의하는데 사용할 수 있습니다.

Example: _EXIT function

```
void _EXIT ExitPart(void)
{
  /* Exit section */
}
```

Figure 8 Definition of an EXIT function

3.4 Header file

정의, 선언 그리고 메크로들은 헤더파일에 저장될 수 있습니다. 어떤 C 파일이든 #include 명령어를 이용하여 헤더파일에 접근할 수 있기 때문에 이러한 방법은 유용합니다.

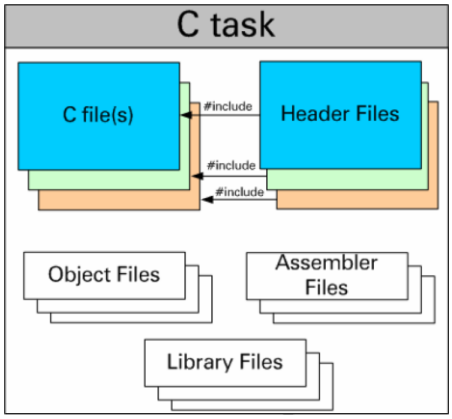


Figure 9 Header files

헤더파일 중의 하나인 "plc.h" 는 _INIT, _CYCLIC , _EXIT 메크로 정의를 포함하고 있습니다. 자동화를 위한 언어 확장은 #include 명령어를 이용하여 C 파일안에 통합 되어야합니다.

Example: include statement

```
#include <bur/plc.h>
```

Figure 10 Include for plc.h

헤더파일은 Automation Studio 디렉토리에 위치해 있습니다.
…₩AS₩GnuInst₩i386-elf₩include₩bur₩plc.h

3.4.1 Rules for the “include” statement

"filename.h" 파일을 찾기 위한 시퀀스는 다음과 같습니다.

ANSI C 에서 **#include "filename.h"** 사용할 때 다음을 적용합니다.

- #include 문을 포함한 파일이 디렉토리 안에 위치해 있어야만 합니다.
- INCLUDE 디렉토리는 대상 Property 에서 정의합니다. (C 컴파일러 탭 리스트)
- 라이브러리 안의 Target system 디렉토리는 프로젝트 안에 import 됩니다. 예를들어, Runtime 와 Standard 라이브러리가 프로젝트에 추가되었다면 다음 디렉토리에서 찾게 됩니다.
  <Project name>.pgp₩Library₩runtime₩i386₩
  <Project name>.pgp₩Library₩standard₩i386₩
- INCLUDE 디렉토리는 project setting 에서 정의합니다. (C 컴파일러 탭 리스트)
- GNU compiler 의 표준 INCLUDE 디렉토리는 다음의 위치에 있습니다:
  …₩AS₩GnuInst₩i386-elf₩include

ANSI C 에서 **#include <filename.h>**을 사용할 때 다음을 적용합니다:

시퀀스는 위의 첫번째 항목을 제외한 나머지 항목들과 동일합니다.

3.5 Compiler settings

include" 디렉토리와 컴파일러 옵션은 Project / Change Runtime verions / Build 에서 정의 할 수 있습니다.

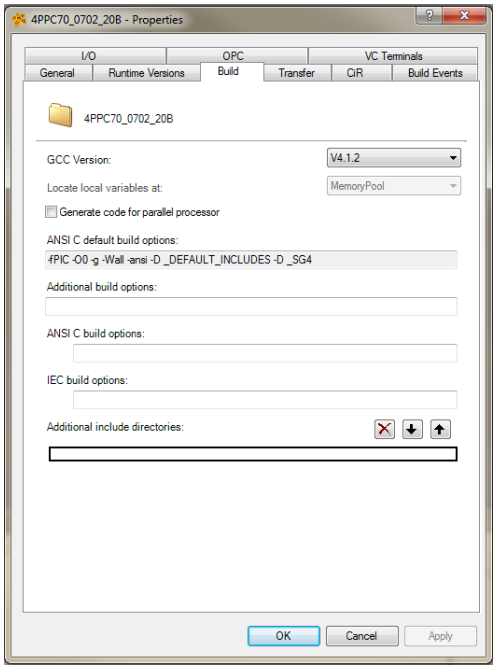


Figure 11 Compiler settings fort he task

Example: C task

C Task 를 생성하세요. (ANSI C Program 또는 ANSI C Program All in One)
아래와 같은 C 파일을 보실 수 있습니다.

```
#include <bur/plctypes.h>

#ifdef _DEFAULT_INCLUDES
    #include <AsDefault.h>
#endif

void _INIT ProgramInit(void)
{

}

void _CYCLIC ProgramCyclic(void)
{

}

void _EXIT ProgramExit(void)
{

}
```

Figure 12 C task with init, cyclic and exit program

4 Variables

4.1 Data types

Automation Studio 의 변수선언, 데이터 정의 등은 IEC61131-3 표준을 준수합니다. 이것은 <bur/plctypes.h> 에 정의되어 있으며, ANSI C 에서 사용할 수 있습니다.

IEC 데이터 타입을 사용한 source code 는 플랫폼에 독립적입니다.

Number of bits	IEC 61131-3	ANSI C
8	BOOL	unsigned char
8	SINT	signed char
8	USINT	unsigned char
16	INT	signed short int
16	UINT	unsigned short int
32	DINT	signed long int
32	UDINT	unsigned long int
32	TIME	signed long int
32	DT DATE_AND_TIME	unsigned long int
String	STRING(x)	char[x+1]
32	REAL	float

NOTE:

헤더파일<bur/plc.h>은 헤더파일<bur/plctypes.h> 에 이미 포함되어있습니다.
그렇기 때문에 C 파일에서 헤더파일<bur/plctypes.h> 만 추가하면 됩니다.

4.2 Scope

Automation Studio 는 지역변수(Local variable)와 전역 변수(Global variable)를 사용합니다. ANSI C 에서 _LOCAL 와 _GLOBAL 메크로를 사용하면 이러한 타입을 생성하는 것이 가능합니다.

물론, “nomal C ” 변수를 사용하는 것도 가능합니다. 그러나 이런 변수들은 C task 에서 지역적으로만 효력을 발휘합니다. 이러한 변수들은 Automation Studio diagnostics tools 인 Watch 와 Monitor 를 사용할 때 입력, 출력, 디스플레이에 링크될 수 없습니다.

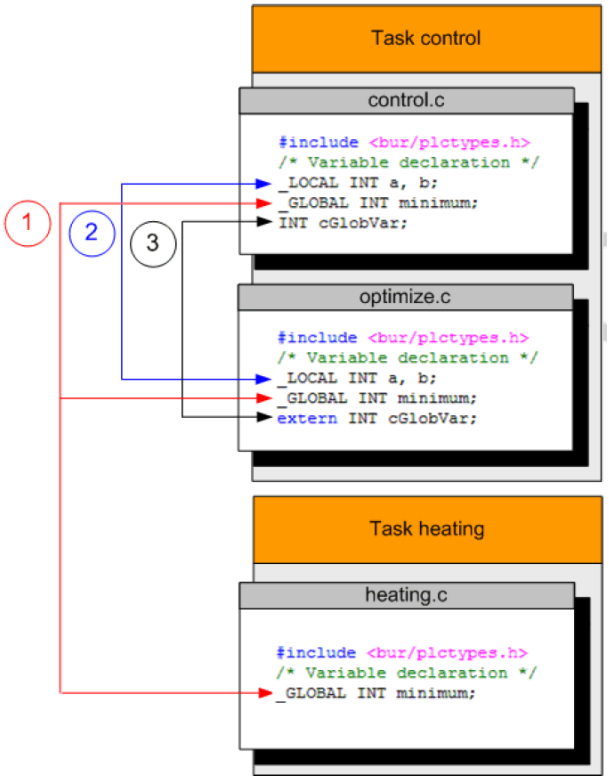


Figure 13 Variable scope.

1

_GLOBAL 로 정의된 변수들은 모든 task 에서 효력을 발휘합니다.

2

_LOCAL 로 정의된 변수들은 각 task 에서만 효력을 발휘합니다.

3

C-global 로 정의된 변수들은 C 파일 안에 속해 있는 모든 task 에 효력을 발휘합니다. 이러한 변수들의 타입은 C 파일에서 “nomal”로 정의되고 다른 모든 C 파일에서 “external” 속성을 갖습니다.
C 와 지역(local)/전역(global) 변수들의 차이점은 아래의 테이블을 참고하십시요.

_Global and _Local variables	C variables
Remanet 나 초기화한 변수로 형성될 수 있음.	C 파일에서 곧바로 초기화 함.
PV 의 최대 크기는 30KB.	PV 의 크기는 단지 메모리 양에의해 제한됨.
PVs 가 I/O 에 접속 할 수 있음.	PVs 가 I/O 에 접속할 수 없음.

_GLOBAL, _LOCAL 로 정의 된 변수는 PVs 가 통신을 이용하여 디스플레이 할 수 있음. (PVI- 예를 들면 시각화(visualization))	변화하는 변수들을 디스플레이 할 수 없음.
다차원 배열이 불가능.	다차원 배열이 가능.
구조체가 16 레이어의 최대량이 될 수 있음.	무제한
열거 데이터 타입(enumeration data type)이 불가능.	열거 데이터 타입(enumeration data type)이 가능.

4.3 Remanent variables

_LOCAL 또는 _GLOBA 로 정의된 변수들은 Remanent 변수로 Variable declaration (shortcut menu:Declaration)에서 정의 할 수 있습니다.

Example: Variable declaration

Name	Type	Scope	Attribute	Value	Owner
MAXTEMP	UINT	local	constant	400	
maxTemp	UINT	local	memory	remanent	
minTemp	UINT	local	memory	remanent	

Figure 14 Declaration

4.4 Constants

필요한 데이터의 타입과 영역(_LOCAL, _GLOBAL)은 “Automation Studio” 에서 상수를 얻기위해 C 파일 내에서 정의되어야 합니다. 그 타입은 variable declaration 에서 값을 변경하는 것이 가능하며, 원하는 값을 입력하시면 됩니다.

4.5 Structures

보통의 C 에서, structure 는 typedef struct 키워드와 데이터의 이름을 사용하여 정의됩니다. IEC61131-3 데이터 타입은 개인적으로 생성한 structure 에서도 사용할 수 있습니다.

Example: Definition of a structure

```
#include <bur/plctypes.h>

typedef struct HeatingZone_typ
{
     INT SetValue;
     INT ActValue;
     STRING  name[10+1];
     UINT status;
     UINT dummy;
}
HeatingZone_typ;
```

Figure 15 Definition of a data type

Note:
헤더 파일에서 Structure 를 정의하면, 모든 C task 에서 Structure 를 쉽게 사용할 수 있습니다.

Example: Weather station

온도 센서는 외부의 온도를 측정합니다. 온도(aiTemp) 는 아날로그로 입력됩니다. 최소 온도(minTemp)와 최고 온도(maxTemp)는 enable 하기 위해 min./max. display 에 저장되어야만 합니다.

aiTemp 변수는 지역변수입니다. minTemp 와 maxTemp 변수는 전역변수이고 remanent 타입입니다.

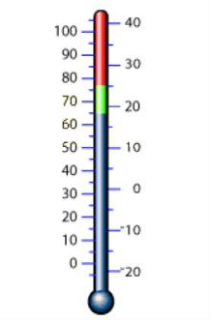


Figure 16 Weather station

코드는 아래와 같습니다:

```
#include <bur/plctypes.h>

#ifdef _DEFAULT_INCLUDES
    #include <AsDefault.h>
#endif

_LOCAL INT aiTemp;

_GLOBAL INT minTemp;
           maxTemp;


void _CYCLIC ProgramCyclic(void)
{
    if (aiTemp < minTemp)
    {
        minTemp = aiTemp;
    }
    if (aiTemp > minTemp)
    {
        maxTemp = aiTemp;
    }

}
```

Figure 17 Code

Variable declaration 은 아래와 같습니다 :

Name	Type	Scope	Attribute	Value	Owner
aiTemp	INT	local	memory		
maxTemp	INT	global	memory	remanent	
minTemp	INT	global	memory	remanent	

Figure 18 Variable declaration

Note:

_GLOBAL, _LOCAL 을 이용하여 변수를 선언하기 위해서는 설정 변경이 필요하다.
Projects/ Settings / ANSI C/C++ compliance 에서 “Enable declaration of PLC variables(_GLOBAL, _LOCAL)를 체크하여 활성화 시킨다.

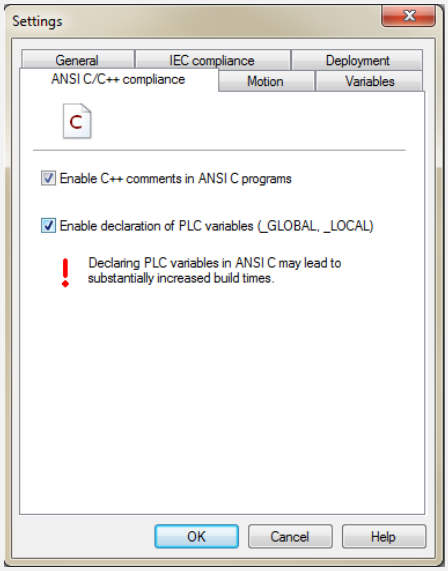


Figure 19 enable the “Enable declaration of PLC variables(_GLOBAL, _LOCAL)“

5 Function Calls

함수는 많은 양의 일을 몇개의 작은 파트로 사용할 수 있는 중요한 요소입니다. 함수는 한번 개발되고 테스트 된 이후에 여러번 재 사용됩니다. 소스코드는 몇몇의 작은 함수로 나눠질 수 있습니다. 다른 사람이 개발한 이전 프로그램 코드를 베이스로 사용될 수 있습니다. 이러한 결과는 깨끗한 프로그램 구조를 만들고 기능을 변경하는데 용이합니다.

5.1 Definition of a function

함수는 N 개의 입력 파라미터와 1 개의 출력파라미터로 구성되어집니다.
각 함수 정의는 아래와 같은 형태를 갖습니다:

```
Function type Function name (parameter declaration)
{
  agreements and statements
}
```

Example: Definition of a function

```
/*Function definition*/
DINT add(INT var1, INT var2) {
    DINT res = 0;
    res = var1 + var2;
    return (res);
}
```

Figure 20 Definition of a funtion

Note:
하나의 함수안에서 정의된 변수는 함수가 call 되었을 때 0 으로 초기화 되지 않습니다. 이런 변수들은 임의의 변수로 가정됩니다.

함수는 소스 파일내에서 어떤 순서로든 선언할 수 있지만 함수 스스로 정의될 수는 없습니다.

Note:
메크로인 _INIT, _CYCLIC 와 _EXIT 또한 함수입니다.

선언된 함수의 파라미터들은 다른 곳에서 선언된 변수와는 다른 유일한 변수가 됩니다.

Example: Definition of a function

```
/* Declaration of function prototyp*/
DINT add(INT var1, INT var2);
```

Figure 21 Declaration of a function

함수는 파일 안에서 정의되고 선언됩니다. 그러나 함수 선언은 *.h 파일에서 만들어지며 함수 정의는 대응하는 *.c 파일에서 만들어집니다.

Note:
함수 사용에 대한 그 이상의 정보는 Automation Studio help 파일의 Automation Software:Automation Studio:Programming Languages:ANSI C:Preliminary notes:C-Limitations. 에서 얻을 수 있습니다.

Exampe: Aquarium

수족관의 온도는 서로 다른 두 곳의 장소에서 측정됩니다. 평균 온도를 계산하기 위한 프로그램을 생성하시오.
평균 온도는 함수를 이용하여 계산하십시요. 함수 선언과 정의는 분리된 파일에서 만들어야 합니다.

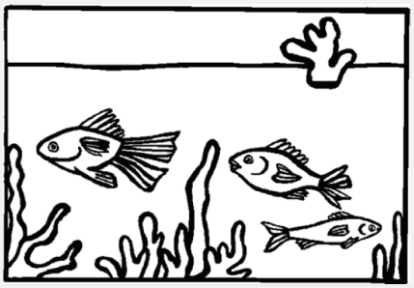


Figure 22 Aquarium

"aquarium" 라는 task 를 생성하십시오.

"average.c" 에 다음 함수를 생성하십시오.

```
#include <bur/plctypes.h>

/* definition pf function */

INT average(INT value1, INT value2)
{
    DINT avg;
    avg = ((DINT)value1 + value2)/2;
    return ((INT) avg);
}
```

Figure 23 Code for the average function

대응하는 헤더파일("average.h")을 생성하십시오:

```
/* declaration of function prototype */
INT average(INT value1, INT value2);
```

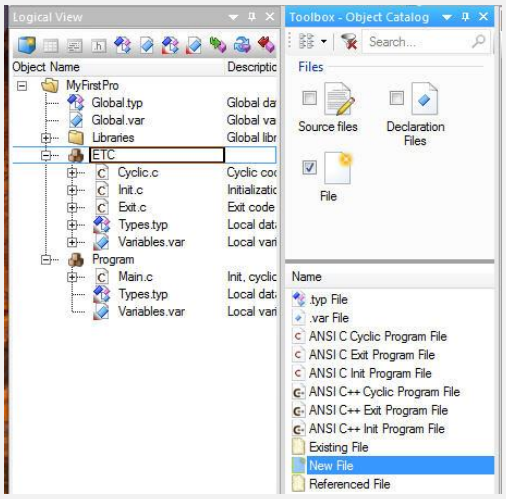
Figure 24 Code for the average header file

메인 프로그램의 모습은 아래를 참조하십시오:

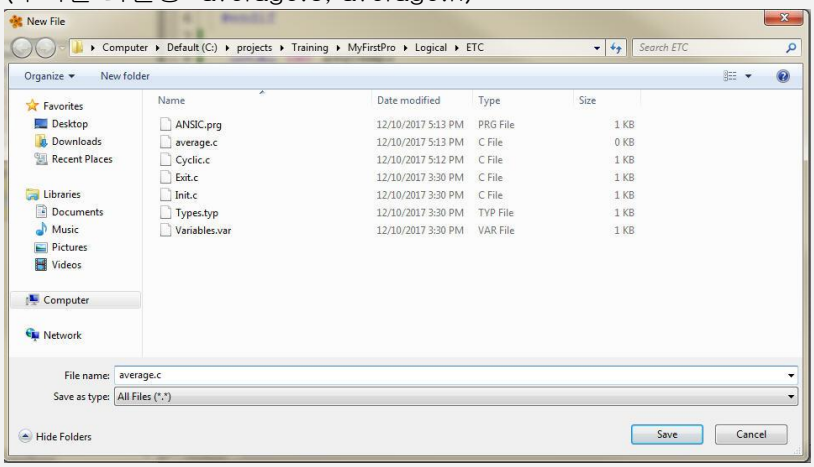
```
#include <bur/plctypes.h>
#include "average.h"

#ifdef _DEFAULT_INCLUDES
    #include <AsDefault.h>
#endif

_LOCAL INT avgTemp;
           Tmp1;
           Tmp2;

void _CYCLIC Cyclic(void)
{
    avgTemp = average(Tmp1, Tmp2);
}
```

Figure 25 Code for the main program

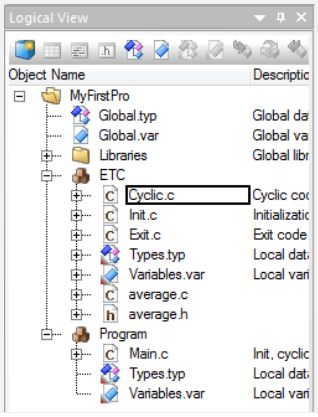
Note:
헤더와 소스를 추가하는 방법은 Logical View 에서 프로그램 선택 후, toolbox 에서 File 선택, New file 을 더블클릭한다.



새로운 팝업창에서 원하는 파일명과 확장명을 기입하고 Save 를 누른다.
(추가할 파일명: average.c, average.h)



로직컬 뷰를 보면 파일이 생성된 것을 확인 할 수 있다.



Note:

ANSI C 는 많은 함수를 제공하고 있습니다. Automation Studio 에서는 이 함수들을 모두 사용할 수는 없습니다. 사용 가능한 함수들의 목록은 Automation Studio help 파일의 Automation Soft-ware:Automation Studio:Programming Languages:ANSI C:B&R Automationstudio GNU-documentation:usable commands 에서 확인 할 수 있습니다.

6 B&R Function Blocks

6.1 General information

C 에서 B&R function block 은 다음의 조건을 만족해야만 합니다.

- function block 을 포함하는 라이브러리는 반드시 Automation Studio 로 import 되어야만 합니다.
- 라이브러리 파일인 *.a 파일이 C task 에 추가되어야만 합니다.
- Function block 이 사용되어지는 동안 C 파일에서 #include 명령어를 사용하여 *.h 파일이 참조되어야만 합니다.

Note:
*.a 파일은 프로젝트 다음의 디렉토리 아래에 위치하게 됩니다.
"*.pgp ₩Library₩Bibliothek₩i386".

Library ... 라이브러리의 이름

6.2 Calling function blocks

C 에서 function block 은 function block 의 이름과 괄호 안의 function block instance 의 주소를 이용하여 call 합니다.

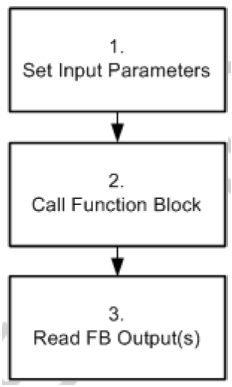


Figure 26 calling a function block

```
/* set input parameters */
TON_01.IN = input;
TON_01.PT = presetTime;

/* call TON with address of instance variable */
TON(&TON_01);

/* read output parameters */
output = TON_01.Q;
elapsedTime = TON_01.ET;
```

Figure 27 Calling a function block in the program

Function block 이 call 되기 전에 입력 파라미터로 사용되는 변수들은 어떤 값이 씌어져야만 합니다. Function block 을 call 하는 것은 1 줄만 차지합니다. Function block 의 출력은 그 후 읽을 수 있습니다.

Example: Bottle counter

컨베이어 밸트 위의 병들을 count 하는 프로그램을 생성하십시요. STANDARD 라이브러리의 CTU(up counter) function block 을 찾아 사용하십시요.

Standard 라이브러리를 import 하기 위해 library manager 를 사용하십시오.
"bottle_c" 로 C 파일을 추가하십시오.

코드는 아래와 같습니다.

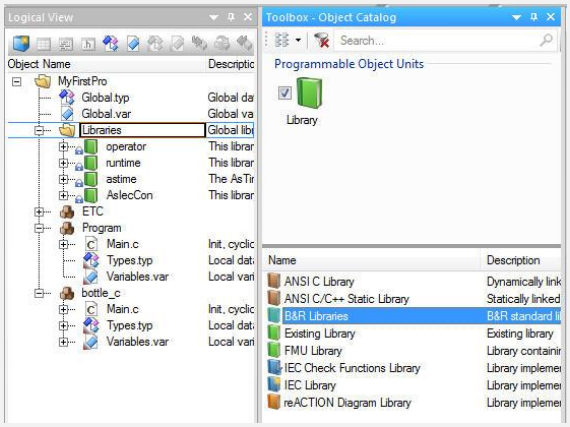
```
#include <bur/plctypes.h>
#include <standard.h>

#ifdef _DEFAULT_INCLUDES
    #include <AsDefault.h>
#endif

_LOCAL BOOL diBottle,
            diReset;
_LOCAL DINT cntCompare,
            cntBottle;
_LOCAL CTU_typ CTU_Bottle; /*instance variable of CTU*/

void _CYCLIC Cyclic(void)
{
    /*set input parameters*/
    CTU_Bottle.CU = diBottle;
    CTU_Bottle.RESET = diReset;

    /*call funtion block with address of instance variable*/
    CTU(&CTU_Bottle);

    /*read output parameters*/
    cntCompare = CTU_Bottle.CV;

}
```

Figure 28 Code

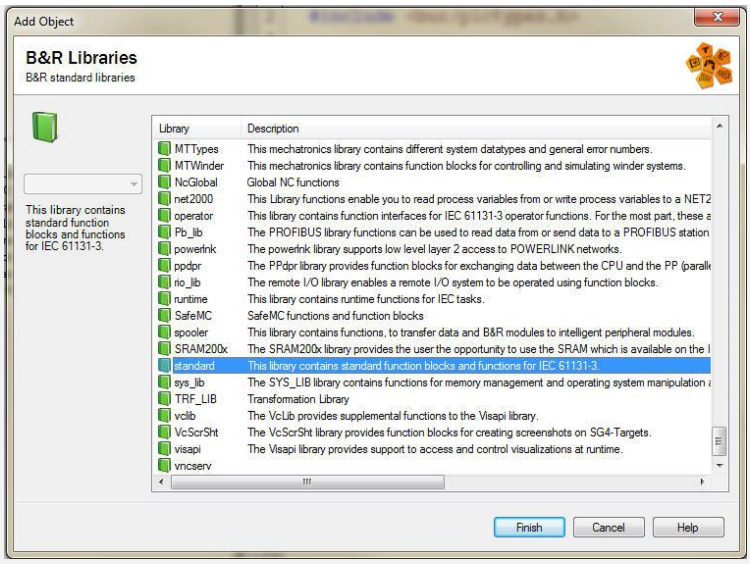
Note:

라이브러리 추가하기

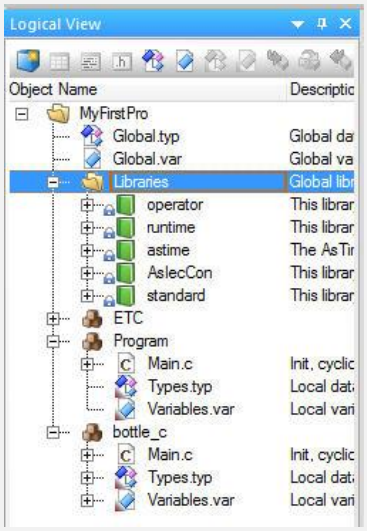
헤더와 소스를 추가하는 방법은 Logical View 에서 Libraries 폴더 선택 후, toolbox 에서 Library 선택, B&R Libraries 을 더블클릭한다.



새로운 팝업창에서 standard 라이브러리를 선택하고 Finish 를 클릭한다.



로직컬 뷰를 보면 Libraries 폴더 아래에 standard 가 추가된 것을 확인할 수 있다.



7 COMPILER

컴파일러는 하나의 프로그램으로서 소스 코드를 assembler 또는 기계어(object code) 로 변환하며, 처리기에 의해 직접적으로 진행됩니다.

Automation Studio 에서는 GNU 컴파일러를 사용합니다. GNU 컴파일러는 ANSI C 컴파일러를 포함하고 있고, K&R C (C 의 처음 버전)를 지원합니다. GNU 컴파일러는 소스 코드의 에러를 테스트하기 위해 다양한 방법을 제공하고 있습니다 (출력에 대한 디버깅 정보와 코드 최적화를 위한 여러 가지 타입을 지원합니다.).

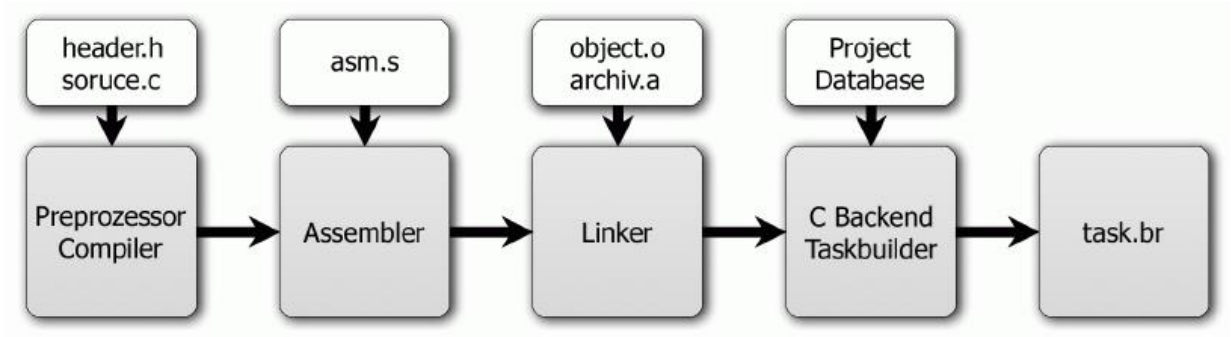


Figure 29 Compiling procedure

Preprocessor	Preprocessor 는 소스코드의 헤더 파일을 통합하고 메크로를 바꿈.
Compiler	Compiler 는 소스코드를 어셈블러 파일(*.s)로 바꿈. 소스코드는 compile 과정에서 처음에 에러 체크를 함.
Assembler	Assembler 는 링커(linker)정보와 함께 object 파일(*.o)을 생성.
Linker	Linker 는 여러 가지 object 파일과 라이브러리로부터 파일을 생성.
Backend, Taskbuilder	(AS 데이터베이스 정보와 하드웨어 세부사항과 함께) Linker 파일은 *.br 파일로 변환됨. 이 파일은 그 다음 CPU 에 의해 처리됨.

Note:

GNU 컴파일러에 대한 문서는 Automation Studio help 의 The B&R-Software:Automation Studio:Programming Languages:ANSI C:B&R Automationstudio GNU-Documentation 에서 확인할 수 있습니다.

8 Summary

많은 개발자들이 선호하는 ANSI C 는 광범위하게 사용되는 프로그래밍 언어입니다. 그렇기 때문에 Automation Studio 에 ANSI C 를 통합하였습니다.

INTRODUCTION 에서 언급한 바와 같이, 이 문서는 C 언어에 대한 교육을 위해 만들어진 자료는 아닙니다. 조금 더 프로그래머들에게 Automation Studio 에서 B&R 의 확장성과 ANSI C 를 정확하게 사용할 수 있도록 도와주기 위해 작성되었습니다.

이 교육 자료는 Automation Studio 에서 ANSI C 프로젝트를 시작할 수 있는 전반적인 정보를 제공하였습니다.

Figure 30 ANSI C

9 Appendix

9.1 Operators

9.1.1 산술 연산자(Arithmetic operators)

Operator	Description	Example
=	Assignment	a = b;
+	Addition	a = b + c;
-	Subtraction	a = b - c;
*	Multiplication	a = b* c;
/	Division	a = b / c;
%	Modulo	a = b % c;

9.1.2 비교 연산자(Comparison operators)

Operator	Description	Example
<	Less than	If (a < b)
>	Greater than	If (a > b)
<=	Less than or equal to	If (a <= b)
>=	Greater than or equal to	If (a >= b)
==	Equal to	If (a == b)
!=	Not equal to	If (a != b)

9.1.3 비트 연산자(Bitwise operators)

Operator	Description	Example
&	Bitwise AND	a = b & c;
|	Bitwise OR	a = b | c;
^	Bitwise XOR	a = b ^ c;
<<	Shift left	a = b << 2
>>	Shift right	a = b >> 3

~	Bitwise negation	a=~b

9.1.4 논리 연산자(Logic operators)

Operator	Description	Example
&&	AND	If (a > 0) && (b > 0)
||	OR	If (a > 0) || (b > 0)

9.2 Commands

9.2.1 IF-Else

IF 문은 프로그램 내에서 결정을 하기 위해 사용됨.

Syntax	Description
If (comparison)	Condition
1 statements	Comparison 이 “True”이면 1 statements 가 실행됨.
else	Optional
2 statements	Comparison 이 “False”이면 2 statements 가 실행됨.

9.2.2 Else-IF

하나 이상의 ELSE_IF 문은 많은 단순한 IF 문으로 인한 혼란스러운 소프트웨어 구조를 만들지 않고, 많은 조건을 테스트할 수 있도록 해 줌.

Syntax	Description
If (comparison 1)	Condition
1 statements	Comparison1 이 “True”이면 1 statements 가 실행됨.
else if (comparison 2)	Optional
2 statements	Comparison2 가 “True”이면 2 statements 가 실행됨. (어떤 대응하는 comparison 도 “True”가 아닌한)
else if (comparison 3)	Optional
3 statements	Comparison3 가 “True”이면 3 statements 가 실행됨. (어떤 대응하는 comparison 도 “True”가 아닌한)
else	
4 statements	어떤 대응하는 comparison 도 “True”가 없으면 4 state-ments 가 실행됨.

Example: IF

```
if (selection ==1)
    program = 0;
else if (selection <=5)
    program = 1;
else
    program = 2;
```

Figure 31 IF example

9.2.3 Switch

Switch 문은 step variable 을 case constant (상수)와 비교함. 만일 이 비교 중 1 개가 맞다면, 대응하는 곳이 실행됨.

Syntax	Description
switch (step variable) {	
Case constant 1: 1 statements	만일 step variable 이 case constant 1 과 같으면 1 statements 가 실행됨.
Case constant 2: 2 statements	만일 step variable 이 case constant 2 와 같으면 2 statements 가 실행됨.
default 3 statements	만일 어떤 경우도 step variable 과 대응하지 않으면 default 로 3 statement 가 실행됨.
}	

Example: Switch

```
switch (selection)
{
    case 1:
        program = 0;
        break;
    case 2:
    case 3:
    case 4:
    case 5:
        program = 1;
        break;
    default:
        program = 2;
        break;
}
```

Figure 32 Switch example

9.2.4 While loops

WHILE 루프(loop)는 특별한 조건이 “True”인 한 문(statement)을 반복 실행하기 위해 사용됨.

Syntax	Description
while (comparison)	실행 조건
Statements	만약 실행 조건이 “True”이면 Statement 가 실행됨.

Example: While

```
while(value < 100)
{
    value+= data;
}
```

Figure 33 While example

9.2.5 For loops

FOR 루프(loop)는 반복을 제한적으로 프로그램 섹션을 실행하기 위해 사용됨.

Syntax	Description
For (expr1; expr2; expr3)	
Statements	

Example: For

```
for(n=0; n<=10; n++)
{
    value[n] =0;
}
```

Figure 34 For example

9.2.6 Do-While

Syntax	Description
do	
Statements	실행 조건이 “True”이면 statement 가 실행됨.
while (comparison);	

Example: Do-While

```
do
{
     value+=data;
}
while (value <100);
```

Figure 35 Do-While example

9.2.7 Break

Break 문은 모든 루프 타입과 switch 문을 중단하는 것을 가능하게 함. 코드는 루프나 switch 문의 끝에서 사용됨.

9.2.8 Continue

Continue 문은 break 문과 정반대의 것임. 이것은 루프에서 문(statement)의 나머지를 처리하지 않고 루프를 계속하기 위해 사용됨.

9.2.9 Goto

C 는 Goto 와 Label 명령을 지원함.