To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

SuperH RISC engine C/C++ Compiler Package

APPLICATION NOTE : [Compiler use guide] Usage of C++ language

This document explains usage and notes of the C++ language using SuperH RISC engine C/C++ compiler V.9.

Contents

1. Uage of C++ language

1.1 Initialization Processing and Post-Processing of Global Class Object

■Important Points:

To use a global class object in C++, you need to call the initialization processing function (_CALL_INIT) and the post-processing function (_CALL_END) before and after the *main* function, respectively.

■What is a global class object?

A global class object is a class object that is declared outside of a function.

（ Class object declaration inside a

```
void main(void)
{
    X XSample(10);
    X* P = &XSample;
    P->Sample2();
    }
```

（Global class object declaration）

```
X XSample(10);
void main(void)
  {
      X* P = &XSample;
      P->Sample2();
  }
```

Declared outside of a function

■Why is initialization processing/post-processing necessary?

If a class object is declared inside a function as shown above, the constructor of class *X* is called when function *main* is executed. In contrast, a global class object declaration is not executed even when a function is executed.
Thus, you need to call _CALL_INIT before calling the *main* function in order to explicitly call the constructor of class *X*. Likewise, call _CALL_END after calling the *main* function in order to call the destructor of class *X*.

■Operations when using and not using _CALL_INIT/_CALL_END:

The following shows the values obtained when the value of member variable *x* of class *X* is referenced.
When not using _CALL_INIT/_CALL_END, no correct value can be obtained and no expression in the *while* statement will be executed as follows:

（Value of member variable *x*）

When using _CALL_INIT →10

When not using _CALL_INIT→0

```
class X{
    int x;
public:
     X(int n){x = n}; // constructor
     ~X(){}              // destructor
     void Sample2(void);
};
X XSample(10); // global class object
void X::Sample2(void)
{
    while(x == 10)
    {
    }
}
void main(void)
{
    X* P = &XSample;

    P->Sample2();
}
```

← Reference

■ How to call _CALL_INIT/_CALL_END:

Provide the following code before and after calling the *main* function.

```
void INIT(void)
{
      _INITSCT();
      _CALL_INIT();
      main();
      _CALL_END();
}
```

If HEW is used, remove the comment characters in the section for calling _CALL_NIT/_CALL_END of *resetprg.c*.

(*PowerON_Reset* function of *resetprg.c*)

```
__entry(vect=0) void PowerON_Reset(void)
{
    set_imask_ccr(1);
    _INITSCT();

   _CALL_INIT();      // Remove the comment when you use global class object

// _INIT_IOLIB();          // Remove the comment when you use SIM I/O

// errno=0;             // Remove the comment when you use errno
// srand(1);            // Remove the comment when you use rand()
// _s1ptr=NULL;        // Remove the comment when you use strtok()

    HardwareSetup();  // Use Hardware Setup
    set_imask_ccr(0);

    main();

// _CLOSEALL();        // Remove the comment when you use SIM I/O

   _CALL_END();        // Remove the comment when you use global class object

    sleep();
}
```

1.2 How to Reference a C Object

■Important Points:

Use an '*extern* "*C*"' declaration to directly use in a C++ program the resources in an existing C object program. Likewise, the resources in a C++ object program can be used in a C program.

■Example of Use:

1. Use an '*extern* "*C*"' declaration to reference a function in a C object program

```
(C++ program)

extern "C" void CFUNC();
void main(void)
{
    X XCLASS;
    XCLASS.SetValue(10);

    CFUNC();
}
```

```
(C++ program)

extern void CFUNC();
void CFUNC()
{
    while(1)
    {
        a++;
    }
}
```

2. Use an 'extern "C"' declaration to reference a function in a C++ object program.

```
(C++ program)

void CFUNC()
{
    CPPFUNC();
}
```

```
(C++ program)

extern "C" void CPPFUNC();
void CPPFUNC(void)
{
    while(1)
    {
        a++;
    }
}
```

■Important Information:

1. A C++ object generated by a previous-version(Ver.5) compiler cannot be linked because the encoding and executing methods have been changed.
   Be sure to recompile it before using it.
2. A function called in the above method cannot be overloaded.

1.3 How to Implement new and delete

■Important Points

To use *new*, implement a low-level function.

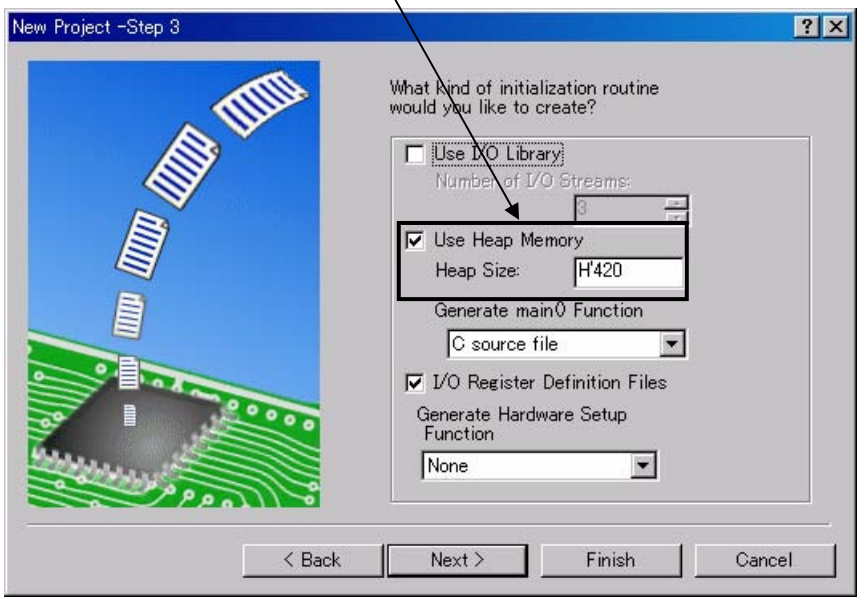
■Description

If *new* is used in an embedded system, the dynamic allocation of actual heap memory is realized using *malloc*. Thus, implement a low-level interface routine (*sbrk*) to specify the size of heap memory to be allocated just as when using *malloc*.

■ Implementation Method:

To use HEW, make sure that [Use Heap Memory] is checked when a workspace is created.
If this option is checked, *sbrk.c* and *sbrk.h* shown on the next page will be automatically created.
Specify the size of heap memory to be allocated in Heap Size.
To change the size after creating a workspace, change the value defined in HEAPSIZE in *sbrk.h*.

If HEW is not used, create a file shown on the next page and implement it in a project.



(sbrk.c)

```
#include <stdio.h>
#include "sbrk.h"

//const size_t _sbrk_size=    /* Specifies the minimum unit of      */
                             /* the defined heap area                        */

static  union  {
   long  dummy ;          /* Dummy for 4-byte boundary          */
   char heap[HEAPSIZE]; /* Declaration of the area managed   */
                          /*                              by sbrk */
 }heap_area ;

static  char  *brk=(char *)&heap_area;/* End address of area assigned     */

/************************************************************************/
/*             sbrk:Data write
                                                                        */
/*             Return value:Start address of the assigned area (Pass)
                    */
/*                                          -1
        (Failure)                                                       */
/************************************************************************/
char  *sbrk(size_t size)             /* Assigned area size    */
{
   char  *p;

   if(brk+size>heap_area.heap+HEAPSIZE) /* Empty area size           */
     return (char *)-1 ;

   p=brk ;                                          /* Area assignment
         */
   brk += size ;                             /* End address update    */
   return p ;
}
```

(sbrk.h)

```
/* size of area managed by sbrk */
#define HEAPSIZE 0x420
```

1.4 Static Member Variable

■Description

In C++, a class member variable with the *static* attribute can be shared among multiple objects of a class type.
Thus, a static member variable comes in handy because it can be used, for example, as a common flag among multiple objects of the same class type.

■Example of Use:

Create five class-A objects within the main function.
Static member variable *num* has an initial value of 0. This value will be incremented by the constructor every time an object is created.
Static member variable num, shared among objects, will have a value of 5 at the maximum.

■FAQ

The following lists some frequently asked questions on using a static member variable.

[L2310 Error Occurred]
When a static member variable is used, message "** L2310 (E) Undefined external symbol "class-name::static-member-variable-name" referenced in "file-name"" is output at linkage.

[Solution]

This error occurs because the static member variable is not defined.
Add either of the following definition as shown on the next page:

If there is an initial value: `int A::num = 0;`
If there is no initial value: `int A::num = 0;`

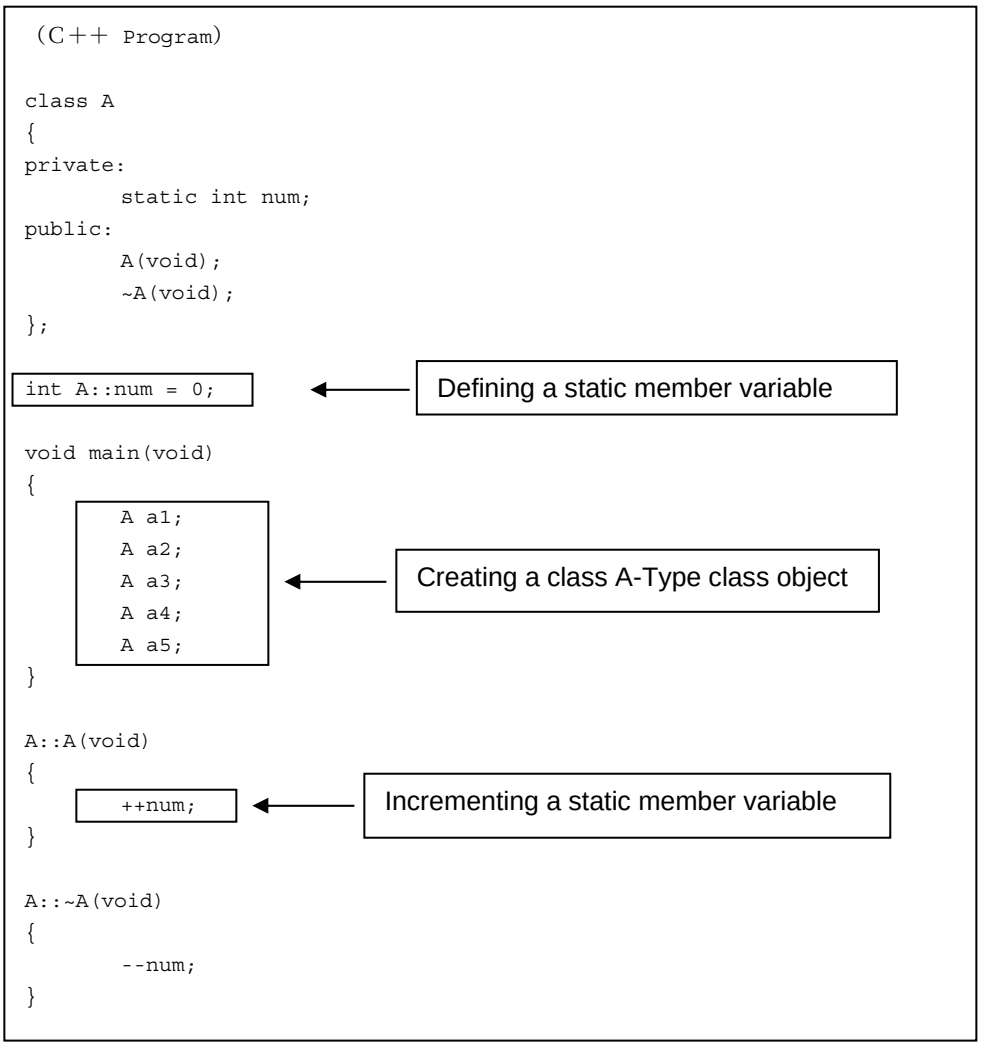
[Unable to assign an initial value]

No initial value is assigned to a *static* member variable to be initialized.

[Solution]

A *static* member variable to be initialized, handled as a variable with an initial value, is created in the D-section by default. Thus, specify the ROM implementation support option of the optimization linkage editor and, in the initial routine, copy the D-section from the ROM to the RAM using the *_INITSCT* function*.

Note: * This solution is not required if HEW automatically creates an initial routine.


(C++ Program)
class A
{
private:
static int num;
public:
A(void);
~A(void);
};
int A::num = 0;
Defining a static member variable
void main(void)
{
A a1;
A a2;
A a3;
A a4;
A a5;
Creating a class A-Type class object
}
A::A(void)
{
++num;
Incrementing a static member variable
}
A::~A(void)
{
--num;
}

2. How to Use Options

2.1 C++ Language for Embedded Applications

■Description

The ROM/RAM sizes and the execution speed are important for an embedded system.
The C++ language for embedded applications (EC++) is a subset of the C++ language. For EC++, some of the C++ functions not appropriate for an embedded system have been removed.
Using EC++, you can create an object appropriate for an embedded system.

■Specification method:

Dialog menu: C/C++ tab Category: Other tab, Check against EC++ language specification
Command line: *eccp*

■Unsupported keywords:

An error message will be output if either of the following keywords is included.

catch、const_cast、dynamic_cast、explicit、mutable、namespace、reinterpret_cast、static_cast、template、throw、try、typeid、typename、using

■Unsupported language specifications:

A warning message will be output if either of the following language specifications is included.

Multiple inheritance, virtual base class

2.2 Run-Time Type Information

■Description

In C++, a class object with a virtual function may have a type identifiable only at run-time.
A run-time identification function is available to provide support in such a situation.
To use this function in C++, use the *type_info* class, *typeid* operator, and *dynamic_cast* operator.
For the Compiler, specify the following option to use run-time type information.
Additionally, specify the following option at linkage to start up the prelinker.

■Specification method:

Dialog menu: CPU tab, Enable/disable runtime type information
Command line: rtti=on | off
Dialog menu: Link/Library tab, Category: Input tab, Prelinker control
Then, select Auto or Run prelinker.
Command line: Do not specify noprelink (default).

■Example of Use of *type_info* Class and *typeid* Operator:

The *type_info* class is intended to identify the run-time type of an object.
Use the *type_info* class to compare types at program execution or acquire a class type.
To use the *type_info* class, specify a class object with a virtual function using the typeid operator.

```
#include <typeinfo.h>
#include <string>
class Base{
protected:
        string *pname1;
public:
        Base() {
                pname1 = new string;
                if (pname1)
                        *pname1 = "Base";
        }
        virtual string Show() {return *pname1;}
        virtual ~Base() {
                if (pname1)
                        delete pname1;
        }
};
class Derived : public Base{
        string *pname2;
public:
        Derived() {
                pname2 = new string;
                if (pname2)
                        *pname2 = "Derived";
        }
        string Show() {return *pname2;}
        ~Derived() {
                if (pname2)
                        delete pname2;
        }
};
void main(void)
{
        Base* pb = new Base;
    Derived* pd = new Derived;

        const type_info& t = typeid(pb);
        const type_info& t1 = typeid(pd);
        t.name();
        t1.name();
}
```

Must be included

Base class

Virtual function

Virtual destructor

Derived class

Virtual function

Virtual destructor

Specifying a class object

Acquiring type name [Base*]

Acquiring type name [Derived*]

■Example of Use of *dynamic_cast* Operator:

Use the *dynamic_cast* operator, for example, to cast at run-time a pointer or reference of the derived-class type to a pointer or reference of the base-class type between a class including a virtual function and its derived class.

```
#include <string>
class Base{
protected:
        string *pname1;
public:
        Base() {
                pname1 = new string;
                if (pname1)
                        *pname1 = "Base";
        }
        virtual string Show() {return *pname1;}
        virtual ~Base() {
                if (pname1)
                        delete pname1;
        }
};
class Derived : public Base{
        string *pname2;
public:
        Derived() {
                pname2 = new string;
                if (pname2)
                        *pname2 = "Derived";
        }
        string Show() {return *pname2;}
        ~Derived() {
                if (pname2)
                        delete pname2;
        }
};
void main(void)
{
        Derived *pderived = new Derived;
        Base *pbase = dynamic_cast<Base *> (pderived);

        string ddd;
        ddd = pbase-> Show();

        delete pbase;
}
```

Base class

Virtual function

Virtual destructor

Derived class

Virtual function

Virtual destructor

Cast to Base * at run-time

Acquiring class name

2.3 Exception Handling Function

■Description:

Unlike C, C++ has a mechanism for handling an error called an exception.
An exception is a means for connecting an error location in a program with an error handling code.
Use the exception mechanism to put together error handling codes in one location.
For the Compiler, specify the following option to use the exception mechanism.

■Specification method:

Dialog menu: CPU tab, Use try, throw and catch of C++
Command line: exception

■Example of Use:

If opening of file "INPUT.DAT" fails, initiate the exception handling and display an error in the standard error output.

```
(C++ program example for exception handling)

void main(void)
{
        try
        {
                if ((fopen("INPUT.DAT","r"))==NULL){
                        char * cp = "cannot open input file\n";
                        throw cp;
                }
        }
        catch(char *pstrError)
        {
                fprintf(stderr,pstrError);
                abort();
        }
        return;
}
```

■Important Information:

The coding performance may deteriorate.

2.4 Disabling Startup of Prelinker

■Description

Starting up the Prelinker will reduce the link speed. The Prelinker need not be running unless the template function or run-time type conversion of C++ is used.
To use the Linker from a command line, specify the following *noprelink* option.
If Hew is used and the *Prelinker control* list box is set to Auto, the output of the *noprelink* option will be automatically controlled.

■Specification method:

Dialog menu: Link/Library tab, Category: Input tab, Prelinker control
Command line: noprelink

3. Advantages and Disadvantages of C++ Coding

There is a possibility that the performance of the object decreases when C++ is used with the embedded system.

This chapter introduces the example that performance deteriorates by using the C++ language and the example that do not deteriorate.

No.	Function	Development and maintenance	Size Reduction	Speed	Section
1	Constructor (1)	◎	△	△	3.1
2	Constructor (2)	◎	△	△	3.2
3	Default parameter	◎	○	○	3.3
4	Inline expansion	○	△	○	3.4
5	Class member function	◎	△	△	3.5
6	*operator* Operator	◎	△	△	3.6
7	Function overloading	◎	○	○	3.7
8	Reference type	◎	○	○	3.8
9	Static function	◎	○	○	3.9
10	Static member variable	◎	○	○	3.10
11	Anonymous *union*	◎	○	○	3.11
12	Virtual function	◎	△	△	3.12

◎ : Same as C ○: Requiring caution in use △: Performance decrease

3.1 Constructor (1)

Development and Maintenance	◎	Size Reduction	△	Speed	△

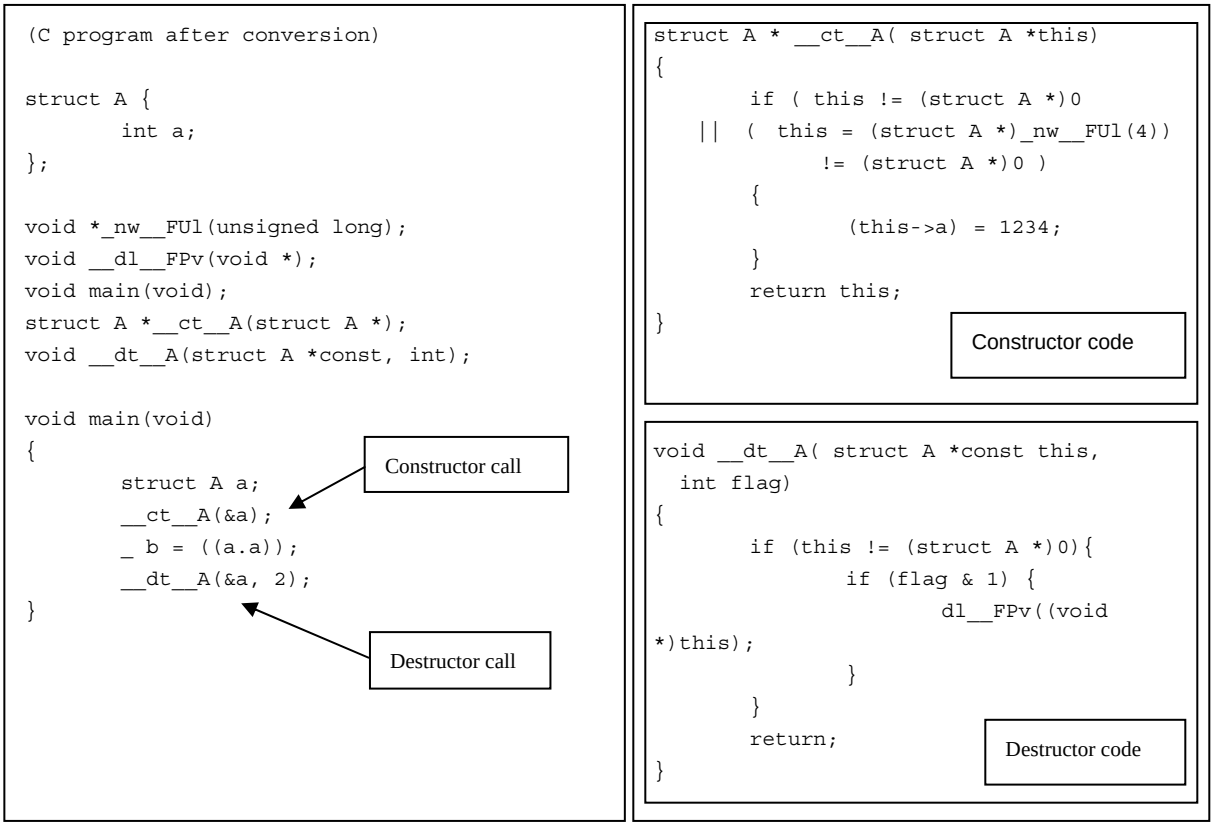
■Important Points:

Use a constructor to automatically initialize a class object. However, use it with caution because it will influence the object size and processing speed as follows:

■Example of Use:

Create a class-A constructor and destructor and compile them.The size and processing speed will be influenced because the constructor and destructor will be called in the class declaration and decisions will be made in the constructor and destructor codes.

```
(C++ program)

class A
{
private:
        int a;
public:
        A(void);
        ~A(void);
        int getValue(void){ return
a; }
};

void main(void)
{
        A a;
        b = a.getValue();
}

A::A(void)
{
        a = 1234;
}

A::~A(void)
{
}
```


(C program after conversion)
struct A {
int a;
};
void *_nw__FUl(unsigned long);
void __dl__FPv(void *);
void main(void);
struct A *__ct__A(struct A *);
void __dt__A(struct A *const, int);
void main(void)
{
struct A a;
__ct__A(&a);
_ b = ((a.a));
__dt__A(&a, 2);
}
Constructor call
Destructor call
struct A * __ct__A(struct A *this)
{
if (this != (struct A *)0
|| (this = (struct A *)_nw__FUl(4))
!= (struct A *)0)
{
(this->a) = 1234;
}
return this;
}
Constructor code
void __dt__A(struct A *const this,
int flag)
{
if (this != (struct A *)0){
if (flag & 1) {
dl__FPv((void
*)this);
}
}
return;
}
Destructor code

3.2 Constructor (2)

Development and Maintenance	◎	Size Reduction	△	Speed	△

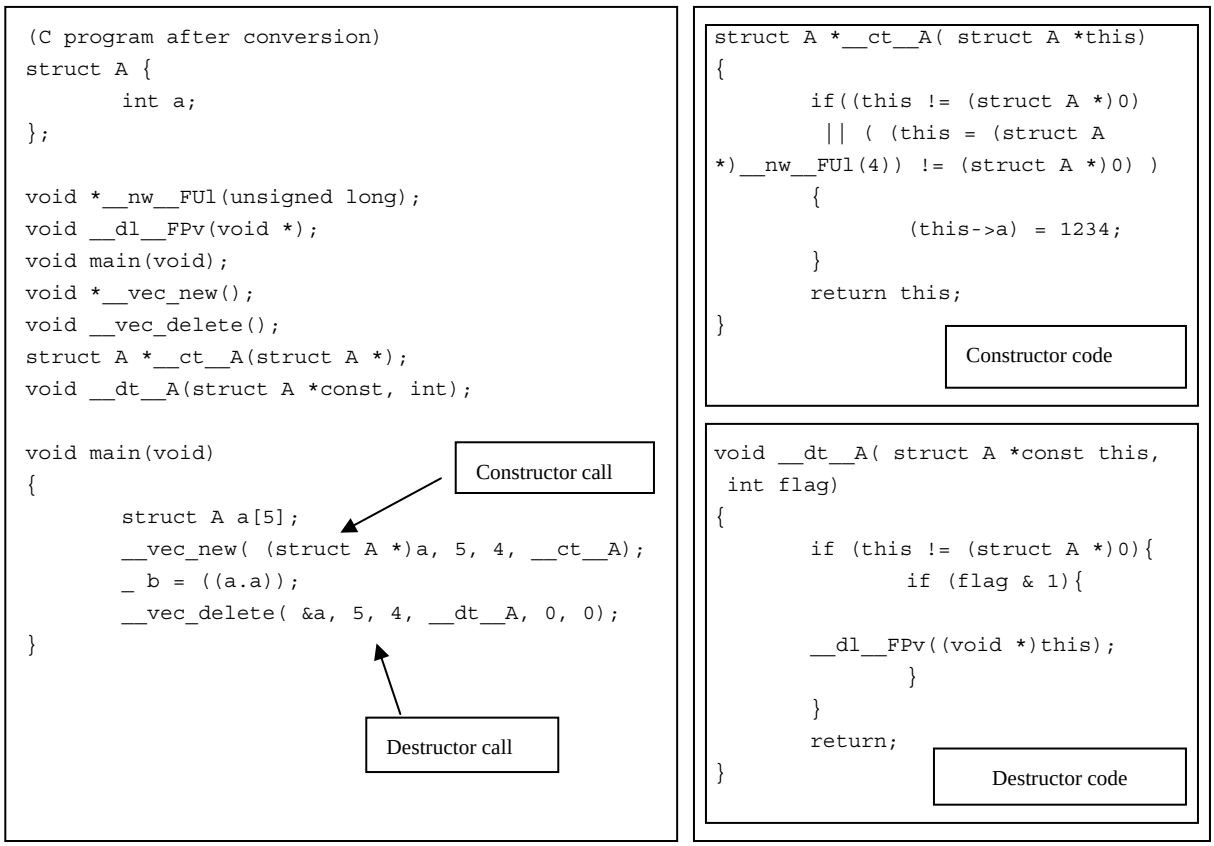
■Important Points

To declare a class in an **array**, use a constructor to automatically initialize a class object. However, use it with caution because it will influence the object size and processing speed as follows:

■Example of Use:

Create a class-A constructor and destructor and compile them.The memory needs to be dynamically allocated and deallocated because the constructor and destructor are called in the class declaration but are declared in the array.
Use *new* and *delete* to dynamically allocate and deallocate the memory.
This requires implementation of a low-level function. (For details on the implementation method, refer to section 8.1.2, Execution Environment Settings, in the SuperH RISC engine C/C++ Compiler, Assembler, Optimizing Linkage Editor User's Manual.)
The size and processing speed will be influenced because decisions and the low-level function processing are added in the constructor and destructor codes.

```
(C++ program)
class A
{
private:
        int a;
public:
        A(void);
        ~A(void);
        int getValue(void){ return
a; }
};

void main(void)
{
        A a[5];
        b = a[0].getValue();
}

A::A(void)
{
        a = 1234;
}

A::~A(void)
{
}
```


(C program after conversion)
struct A {
int a;
};
void *__nw__FUl(unsigned long);
void __dl__FPv(void *);
void main(void);
void *__vec_new();
void __vec_delete();
struct A *__ct__A(struct A *);
void __dt__A(struct A *const, int);
void main(void)
{
struct A a[5];
__vec_new((struct A *)a, 5, 4, __ct__A);
_ b = ((a.a));
__vec_delete(&a, 5, 4, __dt__A, 0, 0);
}
Constructor call
Destructor call
struct A *__ct__A(struct A *this)
{
if((this != (struct A *)0)
|| ((this = (struct A
*)__nw__FUl(4)) != (struct A *)0))
{
(this->a) = 1234;
}
return this;
}
Constructor code
void __dt__A(struct A *const this,
int flag)
{
if (this != (struct A *)0){
if (flag & 1){
__dl__FPv((void *)this);
}
}
return;
}
Destructor code

3.3 Default Parameter

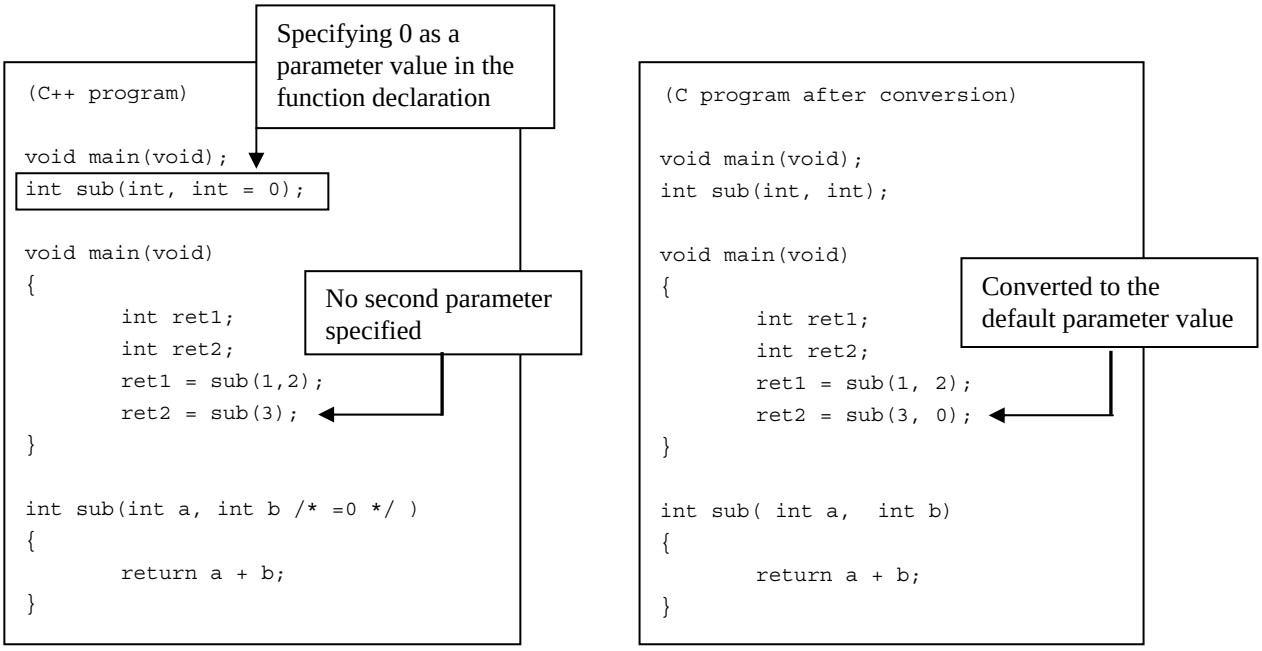
Development and Maintenance	◎	Size Reduction	△	Speed	△

■Important Points:

In C++, a default parameter can be used to set a default used when calling a function.
To use a default parameter, specify a default value for parameters of a function when declaring the function.
This will eliminate the need of specifying a parameter in many of the function calls and enable the use of a default parameter instead, thus improving the development efficiency.
A parameter value can be changed if a parameter is specified.

■Example of Use:

The following shows an example of calling function *sub* when 0 is specified as a default parameter value in the declaration of function *sub*.
As shown below, no parameter needs to be specified if the default parameter value is acceptable when calling function *sub*. Moreover, the efficiency of a program is not deteriorated even when it is converted into C.
In sum, a default parameter ensures superior development and maintenance efficiency and has no disadvantage compared with C.



3.4 Inline Expansion

Development and Maintenance	○	Size Reduction	△	Speed	○

■Important Points:

When coding the definition of a function, specify *inline* in the beginning to cause inline expansion of the function. This will eliminate the overhead of a function call and improve the processing speed.

■Example of Use:

Specify function *sub* as an inline function and inline-expand it in the main function.Then, remove the function *sub* code. However, function *sub* cannot be reference from other files.
Use inline expansion with caution because, although the processing speed is certain to improve, the program size will become too large unless only small functions are used.

```
(C++ program)

int a;

inline int sub(int x, int y)
{
        return (x+y);
}

void main(void)
{
        a = sub(1,2);
}
```

```
(C program after conversion)

int a;
void main(void)
{
    a = 3;
    return;
}
```

Expanding the content of function *sub*

3.5 Class Member Function

Development and Maintenance	◎	Size Reduction	△	Speed	△

■Important Points:

Defining a class will enable information hidingand improve the development and maintenance efficiency.
However, use this technique with caution because it will influence the size and processing speed.

■Example of Use:

In the following example, class member functions *set* and *add* are used to access *private* class member variables *a*, *b*, and *c*.
When calling a class member function, the parameter specification in a C++ program either has only a value or no parameter.
As shown in the C program after conversion, however, the address of class A (struct A) is also passed as a parameter.
Additionally, *private* class member variables *a*, *b*, and *c* are accessed in the class member function code.
However, the *this* pointer is used to access them.
In sum, use a class member function with caution because it will influence the size and processing speed.

```
(C++ program)

class A
{
private:
        int a;
        int b;
        int c;
public:
        void set(int, int, int);
        int add();
};

int main(void)
{
        A a;
        int ret;

        a.set(1,2,3);
        ret = a.add();

        return ret;
}
void A::set(int x, int y, int z)
{
        a = x;
        b = y;
        c = z;
}
int A::add()
{
        return (a += b + c);
}
```

```
(C program after conversion)

struct A {
        int a;
        int b;
        int c;
};
void set__A_int_int(struct A *const, int, int, int);
int add__A(struct A *const);

int main(void)
{
        struct A a;
        int ret;

        set__A_int_int(&a, 1, 2, 3);
        ret = add__A(&a);

        return ret;
}
void set__A_int_int(struct A *const this, int x, int y, int z)
{
        this->a = x;
        this->b = y;
        this->c = z;
        return;
}
int add__A(struct A *const this)
{
        return (this->a += this->b + this->c);
}
```

3.6 operator Operator

Development and Maintenance	○	Size Reduction	△	Speed	△

■Important Points:

In C++, use the keyword, *operator* to overload an operator.
This will enable simple coding of the user's operations such as matrix operations and vector calculations.
However, use *operator* with caution because it will influence the size and processing speed.

■Example of Use:

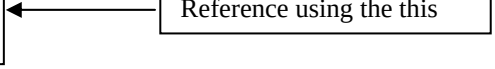
In the following example, unary operator "+" is overloaded using the *operator* keyword.
If the *Vector* class is declared as shown below, unary operator "+" can be changed to the user's operation.
However, the size and processing speed will be influenced because, as shown in the C program after conversion, reference using the *this* pointer is made.

```
(C++ program)

class Vector
{
private:
        int x;
        int y;
        int z;
public:
        Vector & operator+ (Vector &);
};

void main(void)
{
        Vector a,b,c;

        a = b + c;
}

Vector & Vector::operator+ (Vector & vec)
{
        static Vector ret;

        ret.x = x + vec.x;
        ret.y = y + vec.y;
        ret.z = z + vec.z;

        return ret;
}
```

User's operation (addition)

(C program after conversion)

```
struct Vector {
        int x;
        int y;
        int z;
};

void main(void);
struct Vector *__plus__Vector_Vector(struct Vector *const, struct Vector *);

void main(void)
{
        struct Vector a;
        struct Vector b;
        struct Vector c;

        a = *__plus__Vector_Vector(&b, &c);
        return;
}

struct Vector *__plus__Vector_Vector( struct Vector *const this,  struct Vector *vec)
{
        static struct Vector ret;

        ret.x = this->x + vec->x;
        ret.y = this->y + vec->y;
        ret.z = this->z + vec->z;

        return &ret;
}
```



3.7 Overloading of Functions

Development and Maintenance	◎	Size Reduction	○	Speed	○

■Important Points:

In C++, you can "overload" functions, i.e., give the same name to different functions.
Specifically, this feature is effective when you use functions with the same processing but with different types of arguments.
Be careful not to give the same name to functions with no commonality because it is sure to cause malfunctions.
The use of this function will not influence the size or processing speed.

■Example of Use:

In the following example, the first and second parameters are added and the resultant value is used as a return value.
All the functions have the same name, *add* but different parameter and return value types.
As shown in the C program after conversion, the call of the add functions or the code of the add functions do not increase the code size.
Thus, the use of this feature will not influence the size and processing speed.

```
(C++ program)

void main(void);
int add(int,int);
float add(float,float);
double add(double,double);

void main(void)
{
        int    ret_i = add(1, 2);
        float  ret_f = add(1.0f, 2.0f);
        double ret_d = add(1.0, 2.0);
}

int add(int x,int y)
{
        return x+y;
}

float add(float x,float y)
{
        return x+y;
}

double add(double x,double y)
{
        return x+y;
}
```

(C program after conversion)

```
void main(void);
int add__int_int(int, int);
float add__float_float(float, float);
double add__double_double(double, double);

void main(void)
{
        auto int ret_i;
        auto float ret_f;
        auto double ret_d;

        ret_i = add__int_int(1, 2);
        ret_f = add__float_float(1.0f, 2.0f);
        ret_d = add__double_double(1.0, 2.0);
}

int add__int_int( int x,  int y)
{
        return x + y;
}

float add__float_float( float x,  float y)
{
        return x + y;
}

double add__double_double( double x,  double y)
{
        return x + y;
}
```

3.8 Reference Type

Development and Maintenance	◎	Size Reduction	○	Speed	○

■Important Points:

The use of a reference-type parameter will enable simple coding of a program and improve the development and maintenance efficiency.
Additionally, the use of the reference type will not influence the size or processing speed.

■Example of Use:

As shown below, reference-type passing instead of pointer passing will enable simple coding.
In a reference type, not the values but the addresses of *a* and *b* are passed.
The use of a reference type, as shown in the C program after conversion, will not influence the size and processing speed.

(C++ program)

```
void main(void);
void swap(int&, int&);

void main(void)
{
        int a=100;
        int b=256;

        swap(a,b);
}

void swap(int &x, int &y)
{
        int tmp;
        tmp = x;
        x = y;
        y = tmp;
}
```

(C program after conversion)

```
void main(void);
void swap(int *, int *);

void main(void)
{
        int a=100;
        int b=256;

        swap(&a, &b);
}

void swap(int *x,  int *y)
{
        int tmp;
        tmp = *x;
        *x = *y;
        *y = tmp;
}
```

3.9 Static Function

Development and Maintenance	◎	Size Reduction	○	Speed	○

■Important Points:

If the class configuration becomes complex due to derived classes, etc., it will be increasingly more difficult to access *static* class member variables with the *private* attribute until they need to be changed to the *public* attribute.
To access a *static* class member variable without changing the *private* attribute in such a case, create a member function to be used as an interface and specify the *static* variable in the function.
A *static* function is thus used to access only static class member variables.

■Example of Use:

As shown on the next page, use a static function to access a static member variable.
Although the use of a class will influence the code efficiency, the use of a static function itself will not influence the size and processing speed.

■Note:

For details on a static member variable, refer to section 1.4, Static Member Variable.

```
(C++ program)

class A
{
private:
        static int num;
public:
        static int getNum(void);
        A(void);
        ~A(void);
};

int A::num = 0;

void main(void)
{
        int num;

        num = A::getNum();

        A a1;
        num = a1.getNum();

        A a2;
        num = a2.getNum();
}

A::A(void)
{
        ++num;
}

A::~A(void)
{
        --num;
}

int A::getNum(void)
{
        return num;
}
```

Static member variable

Static function

Accessing the static member variable

```
 (C program after conversion)
struct A
{
        char __dummy;
};
void *__nw__FUl(unsigned long);
void __dl__FPv(void *);
int getNum__A(void);
struct A *__ct__A(struct A *);
void __dt__A(struct A *const, int);
int num__1A = 0;
void main(void)
{
        int num;
        struct A a1;
        struct A a2;

        num = getNum__A();

        __ct__A(&a1);
        num = getNum__A();

        __ct__A(&a2);
        num = getNum__A();

        __dt__A(&a2, 2);
        __dt__A(&a1, 2);
}
int getNum__A(void)
{
        return num__1A;
}
struct A *__ct__A( struct A *this)
{
        if ( (this != (struct A *)0)
        || ( (this = (struct A *)__nw__FUl(1)) != (struct A *)0) ){
                ++num__1A;
        }
        return this;
}
void __dt__A( struct A *const this,  int flag)
{
        if (this != (struct A *)0){
                --num__1A;
                if(flag & 1){
                        __dl__FPv((void *)this);
                }
        }
        return;
}
```

Static function

Static member variable

Accessing the static member variable

3.10 Static Member Variable

Development and Maintenance	◎	Size Reduction	○	Speed	○

■Important Points:

In C++, a class member variable with the static attribute can be shared among multiple objects of a class type.
Thus, a static member variable comes in handy because it can be used, for example, as a common flag among multiple objects of the same class type.

■Example of Use:

Create five class-A objects within the *main* function.
Static member variable *num* has an initial value of 0. This value will be incremented by the constructor every time an object is created.
Static member variable num, shared among objects, will have a value of 5 at the maximum.
Additionally, the use of a class will influence the code efficiency.
However, the use of a static member variable itself will not influence the size and processing speed because the Compiler internally handles member variable *num* as if it is an ordinary global variable.

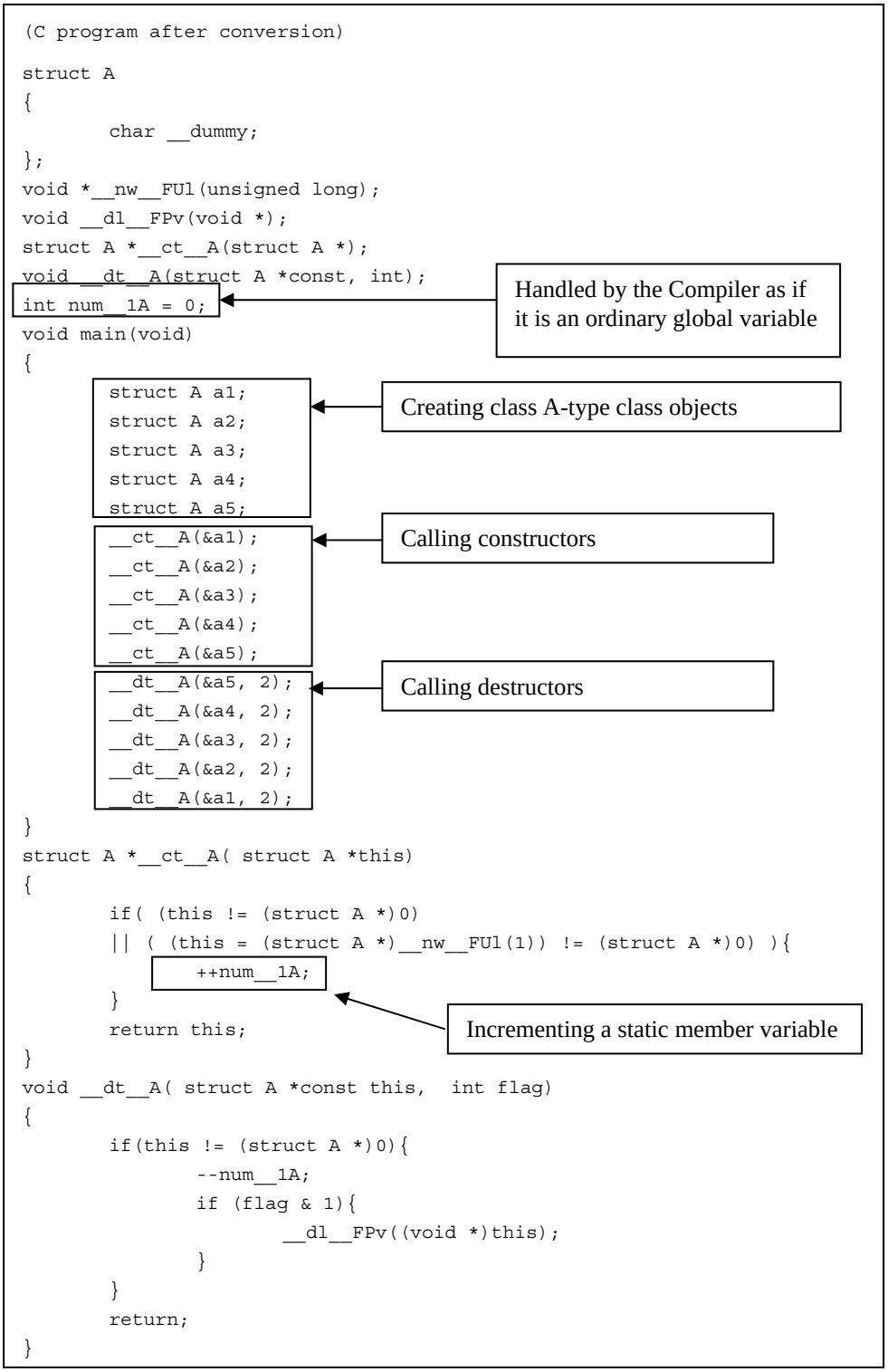
■Note:

For details on a static member variable, refer to section 1.4, Static Member Variable.

```
(C++ program)

class A
{
private:
        static int num;
public:
        A(void);
        ~A(void);
};

int A::num = 0;

void main(void)
{
        A a1;
        A a2;
        A a3;
        A a4;
        A a5;
}

A::A(void)
{
        ++num;
}

A::~A(void)
{
        --num;
}
```

Creating a class A-type class object

Incrementing a *static* member variable


(C program after conversion)
struct A
{
char __dummy;
};
void *__nw__FUl(unsigned long);
void __dl__FPv(void *);
struct A *__ct__A(struct A *);
void __dt__A(struct A *const, int);
int num__1A = 0;
Handled by the Compiler as if it is an ordinary global variable
void main(void)
{
struct A a1;
struct A a2;
struct A a3;
struct A a4;
struct A a5;
Creating class A-type class objects
__ct__A(&a1);
__ct__A(&a2);
__ct__A(&a3);
__ct__A(&a4);
__ct__A(&a5);
Calling constructors
__dt__A(&a5, 2);
__dt__A(&a4, 2);
__dt__A(&a3, 2);
__dt__A(&a2, 2);
__dt__A(&a1, 2);
Calling destructors
}
struct A *__ct__A(struct A *this)
{
if((this != (struct A *)0)
|| ((this = (struct A *)__nw__FUl(1)) != (struct A *)0)){
++num__1A;
}
return this;
Incrementing a static member variable
}
void __dt__A(struct A *const this, int flag)
{
if(this != (struct A *)0){
--num__1A;
if (flag & 1){
__dl__FPv((void *)this);
}
}
return;
}

3.11 Anonymous Union

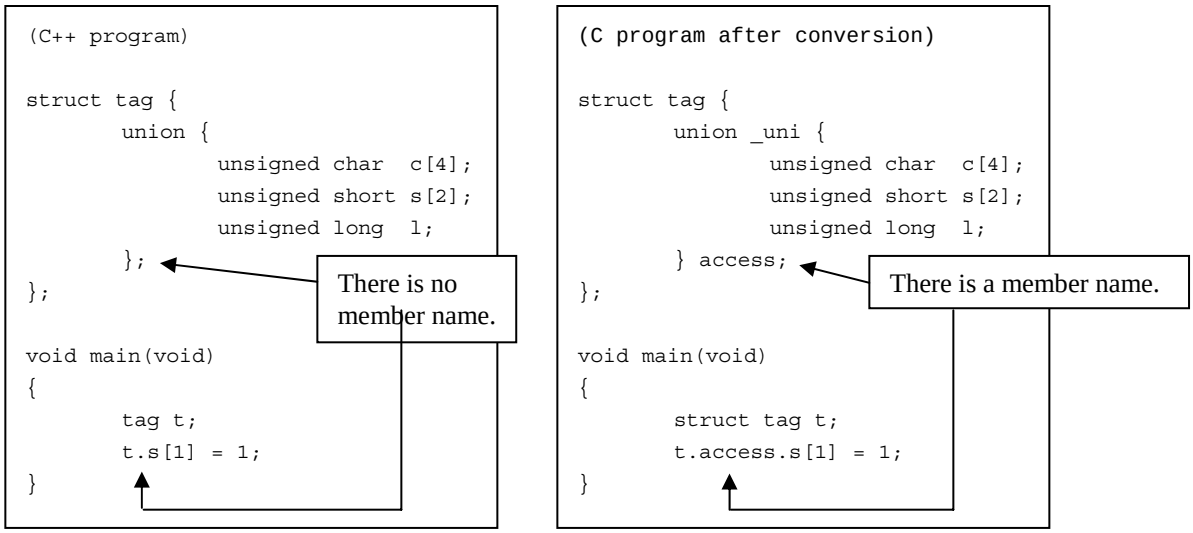
Development and Maintenance	◎	Size Reduction	○	Speed	○

■Important Points:

In C++, use an anonymous *union* to directly access a member without, like in C, having to specify the member name. This will improve the development efficiency. Additionally, it will not influence the size and processing speed.

■Example of Use:

In the following example, function main is used to access *union* member variable *s*.
In the C++ program, member variable *s* is directly accessed. In the C program after conversion, it is accessed using a member name that the Compiler has automatically created.
The use of this simple code enables access to a member variable without influencing the object efficiency.



3.12 Virtual Function

Development and Maintenance	◎	Size Reduction	△	Speed	△

■Important Points:

A virtual function must be used if, as shown in the following program, there is a function with the same name in each of a base class and a derived class. Otherwise, the function call cannot be properly made as intended.
If a virtual function is declared, these calls can be properly made as intended.
Use a virtual function to improve the development efficiency. However, use it with caution because it will influence the size and processing speed.

■Example of Use:

In the *main3* function call, two pointers store class-B addresses.
If *virtual* is declared, the class-B *foo* function is properly called.
If *virtual* is not declared, one of the pointers calls the class-A *foo* function.
The use of a virtual function, resulting in creation of a table, etc. as shown on the next page, will influence the size and speed.

```
(C++ program)

class A
{
private:
    int a;
public:
    virtual void foo(void);
–       };

class B : public A
{
private:
    int b;
public:
    virtual void foo(void);
};

void A::foo(void)
{
}

void B::foo(void)
{
}
```

Virtual function declaration

Virtual function declaration (Can be omitted)

```
void main1(void)
{
    A a;
    a.foo();
}

void main2(void)
{
    B b;
    b.foo();
}

void main3(void)
{
    B  b ;
    A * pa = & b ;
    B * pb = & b ;

    pa->foo();
    pb->foo();
}
```

Virtual function call

■C program after conversion (tables, etc. for virtual functions):

```
struct __T5585724;
struct __type_info;
struct __T5584740;
struct __T5579436;
—       struct A;
struct B;
extern void main1__Fv(void);
extern void main2__Fv(void);
extern void main3__Fv(void);
extern void foo__1AFv(struct A *const);
extern void foo__1BFv(struct B *const);
struct __T5585724
{
    struct __T5584740 *tinfo;
    long offset;
    unsigned char flags;
};
struct __type_info
{
    struct __T5579436 *__vptr;
};
struct __T5584740
{
    struct __type_info tinfo;
    const char *name;
    char *id;
    struct __T5585724 *bc;
};
struct __T5579436
{
    long d;                             // this-pointer offset
    long i;                             // Unassigned
    void (*f)();                        // For virtual function call
};
struct A {                              // Class-A declaration
    int a;
    struct __T5579436 *__vptr;          // Pointer to a virtual function table
};
struct B {                              // Class-B declaration
    struct A __b_A;
    int b;
};
static struct __T5585724 __T5591360[1];
#pragma section $VTBL
extern const struct __T5579436 __vtbl__1A[2];
extern const struct __T5579436 __vtbl__1B[2];
extern const struct __T5579436 __vtbl__Q2_3std9type_info[];
#pragma section
extern struct __T5584740 __T_1A;
extern struct __T5584740 __T_1B;
```

```
static char __TID_1A;                       // Unassigned
static char __TID_1B;                       // Unassigned
static struct __T5585724 __T5591360[1] =    // Unassigned
{
    {
        &__T_1A,
        0L,
        ((unsigned char)22U)
    }
};
#pragma section $VTBL
const struct __T5579436 __vtbl__1A[2] =     // Virtual function table for class-A
{
    {
        0L,                                 // Unassigned area
        0L,                                 // Unassigned area
        ((void (*)())&__T_1A)               // Unassigned area
    },
    {
        0L,                                 // this-pointer offset
        0L,                                 // Unassigned area
        ((void (*)())foo__1AFv)             // ((void (*)())foo__1AFv) // Pointer to
A::foo()
    }
};
const struct __T5579436 __vtbl__1B[2] =     // Virtual function table for class-B
{
    {
        0L,                                 // Unassigned area
        0L,                                 // Unassigned area
        ((void (*)())&__T_1B)               // Unassigned area
    },
    {
        0L,                                 // this-pointer offset
        0L,                                 // Unassigned area
        ((void (*)())foo__1BFv)             // ((void (*)())foo__1BFv) // Pointer to
B::foo()
    }
};
#pragma section
struct __T5584740 __T_1A =                  // Type information for class-A (unassigned)
{
    {
        (struct __T5579436 *)__vtbl__Q2_3std9type_info
    },
    (const char *)"A",
    &__TID_1A,
    (struct __T5585724 *)0
};
struct __T5584740 __T_1B =                  // Type information for class-B (unassigned)
{
    {
        (struct __T5579436 *)__vtbl__Q2_3std9type_info
    },
    (const char *)"B",
    &__TID_1B,
    __T5591360
};
```

■C program after conversion (virtual function calls):

```
void main1__Fv(void)
{
    struct A _a;
    _a.__vptr = __vtbl__1A;
    foo__1AFv( &_a );                              // Call A::foo()
    return;
}
void main2__Fv(void)
{
    struct B _b;
    _b.__b_A.__vptr = __vtbl__1A;
    _b.__b_A.__vptr = __vtbl__1B;
    foo__1BFv( &_b );                              // Call B::foo()
    return;
}
void main3__Fv(void)
{
    struct __T5579436 *_tmp;
    struct B _b;
    struct A *__pa;
    struct B *__pb;

    (*((struct A*)(&_b))).__vptr = __vtbl__1A;
    (*((struct A*)(&_b))).__vptr = __vtbl__1B;
    _pa = (struct A *)&_b;
    _pb = &_b;

    _tmp = _pa->__vptr + 1;
    (  (void (*)(struct A *const))  _tmp->f  )  (  (struct A *)_pa + _tmp->d );
    // Call to B::foo() (The object what is pointed by _pa  is B)

    _tmp = _pb->__b_A.__vptr + 1;
    (  (void (*)(struct B *const))  _tmp->f  )  (  (struct B *)_pb + _tmp->d );
    // Call to B::foo()

    return;
}
```

4. FAQ

4.1 C++ Language Specifications

■Question

Are there any function supporting the development of programs in the C++ language?

■Answer

The SuperH RISC engine C/C++ compiler supports the following functions to support program development in C++:

(1) Support of EC++ class libraries

As EC++ class libraries are supported, the intrinsic C++ class libraries can be used from a C++ program without any specification.

The following four-type libraries are supported:

- Stream I/O class library
- Memory manipulation library
- Complex number calculation class library
- Character string manipulation class library

For details, refer to section 10.4.2, EC++ Class Libraries, in the SuperH RISC engine C/C++ Compiler, Assembler, Optimizing Linkage Editor User's Manual.

(2) EC++ language specification syntax check function

Syntaxes are checked on C++ programs, based upon the EC++ language specifications, using a compiler option.

[Specification method]
Dialog menu: **C/C++** Tab **Category: [Other] Miscellaneous options: Check against EC++ language specification**
Command line: ***ecpp***

(3) Other functions

The following functions are supported for efficient coding of C++ programs:

<Better C functions>

- Inline expansion of functions
- Customization of operators such as +, -,<<
- Simplification of names through the use of multiple definition functions
- Simple coding of comments

<Object-oriented functions>

- Classes
- Constructors
- Virtual functions

For a description of how to set the execution environment at using library functions in a C++ program, refer to section 9.2.2(4), C/C++ library function initial settings(_INILIB), in the SuperH RISC engine C/C++ Compiler, Assembler, Optimizing Linkage Editor User's Manual.

4.2 When the exception handling of the C++ language is used, L2310 link error is occurred.

■Question

Building an application software developed by C++ Language generates two external references of symbols [__curr_eh_stack_entry] and [__eh_curr_region], and results in link error. Why?

■Answer

Compiler automatically generates these symbols when -exception (C++ exception handling) and -rtti=on (Runtime type information) options are used. If these option is not required to use, Please do not enable them.

4.3 When the C++ language is used, #pragma section cannot be correctly specified.

■Question

In using C++ Language, #pragma section must be specified to the first declaration of compiler unit.

■Answer

1. In using C++ Language, #pragma section must be specified to the first declaration of compiler unit.

```
[Example]
#pragma section _A
extern int N;
#pragma section

int N

/* Variable N is allocated in B_A section. */
---------------------------------
extern int N;

#pragma section _A
int N
#pragma section

/* Variable N is allocated in B section. (by default) */
---------------------------------
```

2.In setting the section of static class member variable, #pragma section specification is necessary both to the class member declaration and to the substantial definition.

```
[Example]
class A
{
        private:
        //There is no initial value.
        #pragma section DATA
        static int data_;
        #pragma section

        // There is initial value.
        #pragma section TABLE
        static int table_[2];
        #pragma section
};

// There is no initial value.
#pragma section DATA
int A::data_;
#pragma section

// There is no initial value.
#pragma section TABLE
int A::table_[2]={ 0, 1 };
#pragma section
```

4.4 Which EC++ class library is reentrant?

■Question

Which EC++ class library is reentrant?

■ Answer

Reentrant of the EC++ class library is as follows.

The new is reentrant when the -reent option is specified with the standard library construction tool and there is no exception specification.

■

○: Reentrant △:errno is set ＊: When exception is specified, it is not reentrant.

	FunctionName	Reentrant	
ios	ios_base & boolalpha(ios_base&);	○	
	ios_base & noboolalpha(ios_base&);	○	
	ios_base & showbase(ios_base&);	○	
	ios_base & noshowbase(ios_base&);	○	
	ios_base & showpoint(ios_base&);	○	
	ios_base & noshowpoint(ios_base&);	○	
	ios_base & showpos(ios_base&);	○	
	ios_base & noshowpos(ios_base&);	○	
	ios_base & skipws(ios_base&);	○	
	ios_base & noskipws(ios_base&);	○	
	ios_base & uppercase(ios_base&);	○	
	ios_base & nouppercase(ios_base&);	○	
	ios_base & internal(ios_base&);	○	
	ios_base & left(ios_base&);	○	
	ios_base & right(ios_base&);	○	
	ios_base & dec(ios_base&);	○	
	ios_base & hex(ios_base&);	○	
	ios_base & oct(ios_base&);	○	
	ios_base & fixed(ios_base&);	○	
	ios_base & scientific(ios_base&);	○	
istream	istream& operator>>(istream&, char&);	○	＊
	istream& operator>>(istream&, unsigned char&);	○	＊
	istream& operator>>(istream&, signed char&);	○	＊
	istream& operator>>(istream&, char*);	○	＊
	istream& operator>>(istream&, unsigned char*);	○	＊
	istream& operator>>(istream&, signed char*);	○	＊
	istream& ws(istream& is);	○	＊
ostream	ostream& operator<<(ostream&, char);	○	＊
	ostream& operator<<(ostream&, const char*);	○	＊
	ostream& operator<<(ostream&, signed char);	○	＊
	ostream& operator<<(ostream&, unsigned char);	○	＊
	ostream& operator<<(ostream&, const signed char*);	○	＊
	ostream& operator<<(ostream&, const unsigned char*);	○	＊
	ostream & endl(ostream &);	○	＊
	ostream & ends(ostream &);	○	＊

	ostream & flush(ostream &);	○	*
iostream	extern istream cin;	○	
	extern ostream cout;	○	
iomanip	smanip resetiosflags(ios_base::fmtflags);	○	*
	smanip setiosflags(ios_base::fmtflags);	○	*
	smanip setbase(int);	○	*
	smanip setfill(char);	○	*
	smanip setprecision(int);	○	*
	smanip setw(int);	○	*
complex	float_complex operator+(const float_complex&);	○	*
	float_complex operator−(const float_complex&);	○	*
	float_complex operator+(const float_complex&, const float_complex&);	○	*
	float_complex operator+(const float_complex&, const float&);	○	*
	float_complex operator+(const float&, const float_complex&);	○	*
	float_complex operator−(const float_complex&, const float_complex&);	○	*
	float_complex operator−(const float_complex&, const float&);	○	*
	float_complex operator−(const float&, const float_complex&);	○	*
	float_complex operator*(const float_complex&, const float_complex&);	○	*
	float_complex operator*(const float_complex&, const float&);	○	*
	float_complex operator*(const float&, const float_complex&);	○	*
	float_complex operator/(const float_complex&, const float_complex&);	○	*
	float_complex operator/(const float_complex&, const float&);	○	*
	float_complex operator/(const float&, const float_complex&);	○	*
	bool operator==(const float_complex&, const float_complex&);	○	
	bool operator==(const float_complex&, const float&);	○	
	bool operator==(const float&, const float_complex&);	○	
	bool operator!=(const float_complex&, const float_complex&);	○	
	bool operator!=(const float_complex&, const float&);	○	
	bool operator!=(const float&, const float_complex&);	○	
	istream& operator>>(istream&, float_complex&);	○	*
	ostream& operator<<(ostream&, const float_complex&);	○	
	double_complex operator+(const double_complex&);	○	*
	double_complex operator−(const double_complex&);	○	*
	double_complex operator+(const double_complex&, const double_complex&);	○	*
	double_complex operator+(const double_complex&, const double&);	○	*
	double_complex operator+(const double&, const double_complex&);	○	*
	double_complex operator−(const double_complex&, const double_complex&);	○	*
	double_complex operator−(const double_complex&, const double&);	○	*
	double_complex operator−(const double&, const double_complex&);	○	*
	double_complex operator*(const double_complex&, const double_complex&);	○	*
	double_complex operator*(const double_complex&, const double&);	○	*
	double_complex operator*(const double&, const double_complex&);	○	*
	double_complex operator/(const double_complex&, const double_complex&);	○	*
	double_complex operator/(const double_complex&, const double&);	○	*
	double_complex operator/(const double&, const double_complex&);	○	*
	bool operator==(const double_complex&, const double_complex&);	○	
	bool operator==(const double_complex&, const double&);	○	

	bool operator==(const double&, const double_complex&);	○	
	bool operator!=(const double_complex&, const double_complex&);	○	
	bool operator!=(const double_complex&, const double&);	○	
	bool operator!=(const double&, const double_complex&);	○	
	istream& operator>>(istream&, double_complex&);	○	*
	ostream& operator<<(ostream&, const double_complex&);	○	
	float real(const float_complex&);	○	*
	float imag(const float_complex&);	○	*
	float abs(const float_complex&);	△	
	float arg(const float_complex&);	△	
	float norm(const float_complex&);	○	*
	float_complex conj(const float_complex&);	○	*
	float_complex polar(const float&, const float&);	△	*
	double real(const double_complex&);	○	*
	double imag(const double_complex&);	○	*
	double abs(const double_complex&);	△	
	double arg(const double_complex&);	△	
	double norm(const double_complex&);	○	*
	double_complex conj(const double_complex&);	○	*
	double_complex polar(const double&, const double&);	○	*
	float_complex cos (const float_complex&);	△	*
	float_complex cosh (const float_complex&);	△	*
	float_complex exp (const float_complex&);	△	*
	float_complex log (const float_complex&);	△	*
	float_complex log10(const float_complex&);	△	*
	float_complex pow(const float_complex&, int);	△	
	float_complex pow(const float_complex&, const float&);	△	
	float_complex pow(const float_complex&, const float_complex&);	△	
	float_complex pow(const float&, const float_complex&);	△	
	float_complex sin (const float_complex&);	△	*
	float_complex sinh (const float_complex&);	△	*
	float_complex sqrt (const float_complex&);	△	*
	float_complex tan (const float_complex&);	○	
	float_complex tanh (const float_complex&);	○	
	double_complex cos (const double_complex&);	△	*
	double_complex cosh (const double_complex&);	△	*
	double_complex exp (const double_complex&);	△	*
	double_complex log (const double_complex&);	△	*
	double_complex log10(const double_complex&);	△	*
	double_complex pow(const double_complex&, int);	△	
	double_complex pow(const double_complex&, const double&);	△	
	double_complex pow(const double_complex&, const double_complex&);	△	
	double_complex pow(const double&, const double_complex&);	△	
	double_complex sin (const double_complex&);	△	*
	double_complex sinh (const double_complex&);	△	*
	double_complex sqrt (const double_complex&);	△	*
	double_complex tan (const double_complex&);	○	
	double_complex tanh (const double_complex&);	○	

string	string operator + (const string &lhs,const string &rhs);	○	*
	string operator + (const char *lhs,const string &rhs);	○	*
	string operator + (char lhs,const string &rhs);	○	*
	string operator + (const string &lhs,const char *rhs);	○	*
	string operator + (const string &lhs,char rhs);	○	*
	bool operator == (const string &lhs,const string &rhs);	○	*
	bool operator == (const char *lhs,const string &rhs);	○	*
	bool operator == (const string &lhs,const char *rhs);	○	*
	bool operator != (const string &lhs,const string &rhs);	○	*
	bool operator != (const char *lhs,const string &rhs);	○	*
	bool operator != (const string &lhs,const char *rhs);	○	*
	bool operator < (const string &lhs,const string &rhs);	○	*
	bool operator < (const char *lhs,const string &rhs);	○	*
	bool operator < (const string &lhs,const char *rhs);	○	*
	bool operator > (const string &lhs,const string &rhs);	○	*
	bool operator > (const char *lhs,const string &rhs);	○	*
	bool operator > (const string &lhs,const char *rhs);	○	*
	bool operator <= (const string &lhs,const string &rhs);	○	*
	bool operator <= (const char *lhs,const string &rhs);	○	*
	bool operator <= (const string &lhs,const char *rhs);	○	*
	bool operator >= (const string &lhs,const string &rhs);	○	*
	bool operator >= (const char *lhs,const string &rhs);	○	*
	bool operator >= (const string &lhs,const char *rhs);	○	*
	void swap(string &lhs, string &rhs);	○	*
	istream & operator >> (istream &is,string &str);	○	*
	ostream & operator << (ostream &os,const string &str);	○	*
	istream & getline (istream &is,string &str,char delim);	○	*
	istream & getline (istream &is,string &str);	○	*
	double_complex::operator /=(const double_complex &)	△	*
	float_complex::operator /=(const float_complex &)	△	
	double_complex::operator *=(const double_complex &)	○	*
	float_complex::operator /=(const float_complex &)	○	*
	float_complex::operator *=(const float_complex &)	○	*
	float_complex::float_complex(const double_complex &)	○	*
	istream::operator >>(float &)	○	*
	istream::operator >>(double &)	○	*
	istream::operator >>(long double &)	○	*
	istream::_ec2p_getfstr(char *, unsigned int)	○	*
	istream::operator >>(bool &)	○	*
	istream::operator >>(short &)	○	*
	istream::operator >>(unsigned short &)	○	*
	istream::operator >>(int &)	○	*
	istream::operator >>(unsigned int &)	○	*
	istream::operator >>(long &)	○	*
	istream::operator >>(unsigned long &)	○	*
	istream::operator >>(void *&)	○	*
	istream::_ec2p_getistr(char *, unsigned int, int)	○	*
	istream::_ec2p_strtoul(char *, char **, int)	○	*

	istream::operator >>(streambuf *)	○	*
	istream::get()	○	*
	istream::get(char &)	○	*
	istream::get(signed char &)	○	*
	istream::get(unsigned char &)	○	*
	istream::get(char *, long)	○	*
	istream::get(char *, long, char)	○	*
	istream::get(signed char *, long)	○	*
	istream::get(signed char *, long, char)	○	*
	istream::get(unsigned char *, long)	○	*
	istream::get(unsigned char *, long, char)	○	*
	istream::get(streambuf &)	○	*
	istream::get(streambuf &, char)	○	*
	istream::getline(char *, long)	○	*
	istream::getline(char *, long, char)	○	*
	istream::getline(signed char *, long)	○	*
	istream::getline(signed char *, long, char)	○	*
	istream::getline(unsigned char *, long)	○	*
	istream::getline(unsigned char *, long, char)	○	*
	istream::_ec2p_gets(char *, long, char, int)	○	*
	istream::ignore(long, long)	○	*
	istream::peek()	○	*
	istream::putback(char)	○	*
	istream::unget()	○	*
	istream::sync()	○	*
	istream::tellg()	○	*
	istream::seekg(long)	○	*
	istream::seekg(long, int)	○	*
	streambuf::_$gptr() const	○	*
	streambuf::_$egptr() const	○	*
	istream::read(char *, long)	○	*
	istream::read(signed char *, long)	○	*
	istream::read(unsigned char *, long)	○	*
	istream::readsome(char *, long)	○	*
	istream::readsome(signed char *, long)	○	*
	istream::readsome(unsigned char *, long)	○	*
	istream::sentry::sentry(istream &, bool)	○	*
	ostream::operator <<(void *)	○	*
	ostream::operator <<(streambuf *)	○	*
	ostream::operator <<(float)	○	*
	ostream::operator <<(double)	○	*
	ostream::operator <<(long double)	○	*
	ostream::operator <<(short)	○	*
	ostream::operator <<(unsigned short)	○	*
	ostream::operator <<(int)	○	*
	ostream::operator <<(unsigned int)	○	*
	ostream::operator <<(long)	○	*
	ostream::operator <<(unsigned long)	○	*

	ostream::sentry::sentry(ostream &)	○	*
	ostream::sentry::~sentry()	○	*
	ostream::put(char)	○	*
	ostream::write(const char *, long)	○	*
	ostream::write(const signed char *, long)	○	*
	ostream::write(const unsigned char *, long)	○	*
	ostream::flush()	○	*
	ostream::tellp()	○	*
	ostream::seekp(long)	○	*
	ostream::seekp(long, int)	○	*
	string::_$size() const	○	*
	string::operator +=(const string &)	○	*
	string::operator +=(const char *)	○	*
	string::operator +=(char)	○	*
	string::append(const string &)	○	*
	string::append(const string &, unsigned long, unsigned long)	○	*
	string::append(const char *, unsigned long)	○	*
	string::append(const char *)	○	*
	string::append(unsigned long, char)	○	*
	string::_$c_str() const	○	*
	string::operator =(const string &)	○	*
	string::operator =(const char *)	○	*
	string::operator =(char)	○	*
	string::assign(const string &)	○	*
	string::assign(const string &, unsigned long, unsigned long)	○	*
	string::assign(const char *, unsigned long)	○	*
	string::assign(const char *)	○	*
	string::assign(unsigned long, char)	○	*
	string::swap(string &)	○	*
	string::_$size() const	○	*
	string::_$c_str() const	○	*
	string::compare(const string &) const	○	*
	string::compare(unsigned long, unsigned long, const string &) const	○	*
	string::compare(unsigned long, unsigned long, const string &, unsigned long, unsigned long) const	○	*
	string::compare(const char *) const	○	*
	string::compare(unsigned long, unsigned long, const char *, unsigned long) const	○	*
	string::find(const string &, unsigned long) const	○	*
	string::find(const char *, unsigned long, unsigned long) const	○	*
	string::find(const char *, unsigned long) const	○	*
	string::find(char, unsigned long) const	○	*
	string::find_first_of(const string &, unsigned long) const	○	*
	string::find_first_of(const char *, unsigned long, unsigned long) const	○	*
	string::find_first_of(const char *, unsigned long) const	○	*
	string::find_first_of(char, unsigned long) const	○	*
	string::find_first_not_of(const string &, unsigned long) const	○	*
	string::find_first_not_of(const char *, unsigned long, unsigned long) const	○	*
	string::find_first_not_of(const char *, unsigned long) const	○	*
	string::find_first_not_of(char, unsigned long) const	○	*

	string::find_last_of(const string &, unsigned long) const	○	*
	string::find_last_of(const char *, unsigned long, unsigned long) const	○	*
	string::find_last_of(const char *, unsigned long) const	○	*
	string::find_last_of(char, unsigned long) const	○	*
	string::find_last_not_of(const string &, unsigned long) const	○	*
	string::find_last_not_of(const char *, unsigned long, unsigned long) const	○	*
	string::find_last_not_of(const char *, unsigned long) const	○	*
	string::find_last_not_of(char, unsigned long) const	○	*
	string::rfind(const string &, unsigned long) const	○	*
	string::rfind(const char *, unsigned long, unsigned long) const	○	*
	string::rfind(const char *, unsigned long) const	○	*
	string::rfind(char, unsigned long) const	○	*
	string::insert(unsigned long, const string &)	○	*
	string::insert(unsigned long, const string &, unsigned long, unsigned long)	○	*
	string::insert(unsigned long, const char *, unsigned long)	○	*
	string::insert(unsigned long, const char *)	○	*
	string::insert(unsigned long, unsigned long, char)	○	*
	string::insert(char *, char)	○	*
	string::insert(char *, unsigned long, char)	○	*
	string::replace(unsigned long, unsigned long, const string &)	○	*
	string::replace(unsigned long, unsigned long, const string &, unsigned long, unsigned long)	○	*
	string::replace(unsigned long, unsigned long, const char *, unsigned long)	○	*
	string::replace(unsigned long, unsigned long, const char *)	○	*
	string::replace(unsigned long, unsigned long, unsigned long, char)	○	*
	string::replace(char *, char *, const string &)	○	*
	string::replace(char *, char *, const char *, unsigned long)	○	*
	string::replace(char *, char *, const char *)	○	*
	string::replace(char *, char *, unsigned long, char)	○	*
	string::substr(unsigned long, unsigned long) const	○	*
	ios::init(streambuf *)	○	*
	ios_base::Init::init_cnt	○	*
	string::npos	○	*
	string::_ec2p_at	○	*
	mystrbuf::open(const char *, int)	○	*
	mystrbuf::close()	○	*
	mystrbuf::setbuf(char *, long)	○	*
	mystrbuf::seekoff(long, int, int)	○	*
	mystrbuf::seekpos(long, int)	○	*
	mystrbuf::sync()	○	*
	mystrbuf::_$showmanyc()	○	*
	mystrbuf::underflow()	○	*
	mystrbuf::pbackfail(long)	○	*
	mystrbuf::overflow(long)	○	*
	mystrbuf::_Init(_f_type *)	○	*
	mystrbuf::open(const char *, int)::_$wkmode	○	*
	mystrbuf::open(const char *, int)::_$mddat	○	*
	streambuf::sbumpc()	○	*
	streambuf::sgetc()	○	*

	streambuf::sputbackc(char)	○	*
	streambuf::sungetc()	○	*
	streambuf::sputc(char)	○	*
	streambuf::streambuf()	○	*
	streambuf::gbump(int)	○	*
	streambuf::pbump(int)	○	*
	streambuf::_$setbuf(char *, long)	○	*
	streambuf::_$seekoff(long, int, int)	○	*
	streambuf::_$seekpos(long, int)	○	*
	streambuf::_$sync()	○	*
	streambuf::_$showmanyc()	○	*
	streambuf::xsgetn(char *, long)	○	*
	streambuf::_$underflow()	○	*
	streambuf::uflow()	○	*
	streambuf::_$pbackfail(long)	○	*
	streambuf::xsputn(const char *, long)	○	*
	streambuf::_$overflow(long)	○	*
	streambuf::_ec2p_setbPtr(char **, char **, long *, long *, char *)	○	*
	ios_base::boolalpha	○	*
	ios_base::dec	○	*
	ios_base::fixed	○	*
	ios_base::hex	○	*
	ios_base::internal	○	*
	ios_base::left	○	*
	ios_base::oct	○	*
	ios_base::right	○	*
	ios_base::scientific	○	*
	ios_base::showbase	○	*
	ios_base::showpoint	○	*
	ios_base::showpos	○	*
	ios_base::skipws	○	*
	ios_base::unitbuf	○	*
	ios_base::uppercase	○	*
	ios_base::adjustfield	○	*
	ios_base::basefield	○	*
	ios_base::floatfield	○	*
	ios_base::badbit	○	*
	ios_base::eofbit	○	*
	ios_base::failbit	○	*
	ios_base::goodbit	○	*
	ios_base::app	○	*
	ios_base::ate	○	*
	ios_base::binary	○	*
	ios_base::in	○	*
	ios_base::out	○	*
	ios_base::trunc	○	*
	ios_base::beg	○	*
	ios_base::cur	○	*

	ios_base::end	○	*
	ios_base::_fmtmask	○	*
	ios_base::_statemask	○	*
new	void* operator new(size_t size)		*
	void* operator new[ ](size_t size)		*
	void* operator new(size_t size, void* ptr)	○	*
	void* operator new[ ](size_t size, void* ptr)	○	*
	void operator delete(void* ptr)		*
	void operator delete[ ](void* ptr)		*
	new_handler set_new_handler(new_handler new_P)		*

Website and Support <website and support,ws>

Renesas Technology Website
http://www.renesas.com/

Inquiries
http://www.renesas.com/inquiry
csc@renesas.com

Revision Record <revision history,rh>

		Description	
Rev.	**Date**	**Page**	**Summary**
1.00	Sep.01.07	—	First edition issued

Notes regarding these materials

1. This document is provided for reference purposes only so that Renesas customers may select the appropriate Renesas products for their use. Renesas neither makes warranties or representations with respect to the accuracy or completeness of the information contained in this document nor grants any license to any intellectual property rights or any other rights of Renesas or any third party with respect to the information in this document.
2. Renesas shall have no liability for damages or infringement of any intellectual property or other rights arising out of the use of any information in this document, including, but not limited to, product data, diagrams, charts, programs, algorithms, and application circuit examples.
3. You should not use the products or the technology described in this document for the purpose of military applications such as the development of weapons of mass destruction or for the purpose of any other military use. When exporting the products or technology described herein, you should follow the applicable export control laws and regulations, and procedures required by such laws and regulations.
4. All information included in this document such as product data, diagrams, charts, programs, algorithms, and application circuit examples, is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas products listed in this document, please confirm the latest product information with a Renesas sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas such as that disclosed through our website. (http://www.renesas.com)
5. Renesas has used reasonable care in compiling the information included in this document, but Renesas assumes no liability whatsoever for any damages incurred as a result of errors or omissions in the information included in this document.
6. When using or otherwise relying on the information in this document, you should evaluate the information in light of the total system before deciding about the applicability of such information to the intended application. Renesas makes no representations, warranties or guaranties regarding the suitability of its products for any particular application and specifically disclaims any liability arising out of the application and use of the information in this document or Renesas products.
7. With the exception of products specified by Renesas as suitable for automobile applications, Renesas products are not designed, manufactured or tested for applications or otherwise in systems the failure or malfunction of which may cause a direct threat to human life or create a risk of human injury or which require especially high quality and reliability such as safety systems, or equipment or systems for transportation and traffic, healthcare, combustion control, aerospace and aeronautics, nuclear power, or undersea communication transmission. If you are considering the use of our products for such purposes, please contact a Renesas sales office beforehand. Renesas shall have no liability for damages arising out of the uses set forth above.
8. Notwithstanding the preceding paragraph, you should not use Renesas products for the purposes listed below:
   (1) artificial life support devices or systems
   (2) surgical implantations
   (3) healthcare intervention (e.g., excision, administration of medication, etc.)
   (4) any other purposes that pose a direct threat to human life

   Renesas shall have no liability for damages arising out of the uses set forth in the above and purchasers who elect to use Renesas products in any of the foregoing applications shall indemnify and hold harmless Renesas Technology Corp., its affiliated companies and their officers, directors, and employees against any and all damages arising out of such applications.
9. You should use the products described herein within the range specified by Renesas, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas shall have no liability for malfunctions or damages arising out of the use of Renesas products beyond such specified ranges.
10. Although Renesas endeavors to improve the quality and reliability of its products, IC products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Please be sure to implement safety measures to guard against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other applicable measures. Among others, since the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
11. In case Renesas products listed in this document are detached from the products to which the Renesas products are attached or affixed, the risk of accident such as swallowing by infants and small children is very high. You should implement safety measures so that Renesas products may not be easily detached from your products. Renesas shall have no liability for damages arising out of such detachment.
12. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written approval from Renesas.
13. Please contact a Renesas sales office if you have any questions regarding the information contained in this document, Renesas semiconductor products, or if you have any other inquiries.