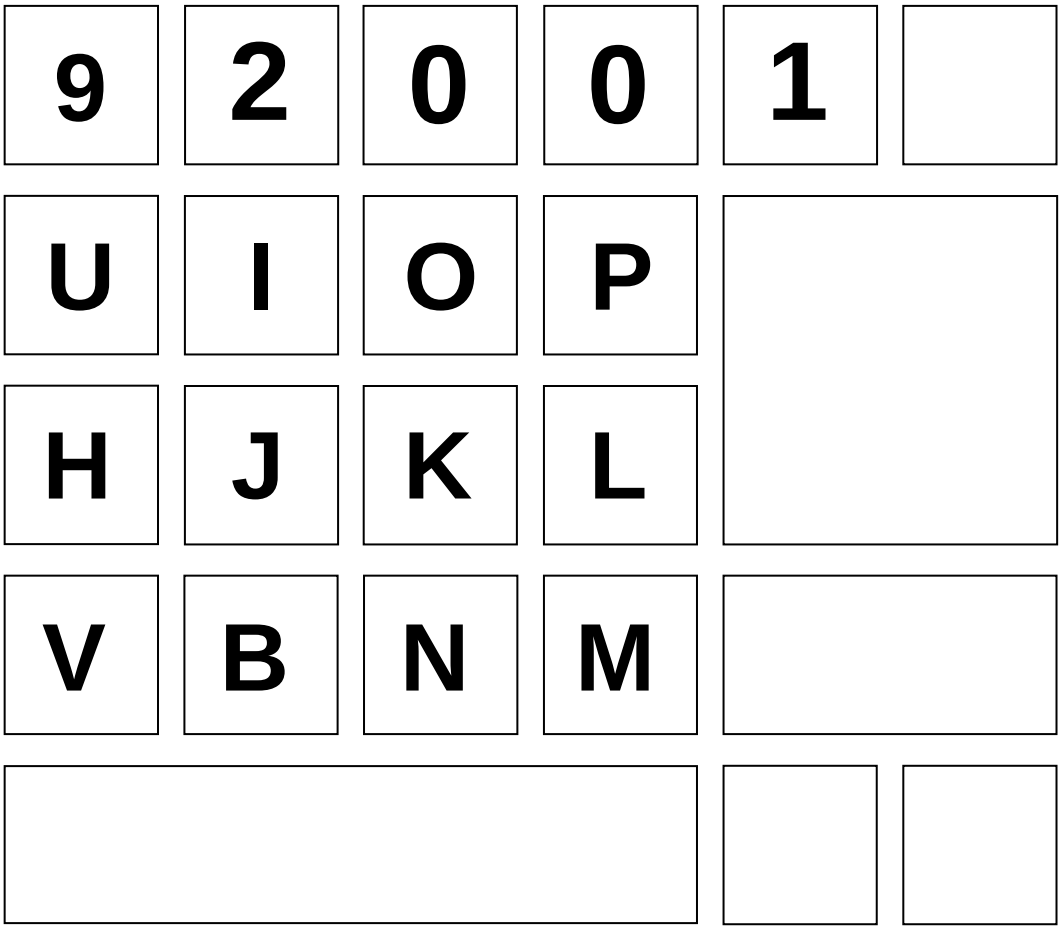
METI
Ministry of Economy,
Trade and Industry

Textbook for Fundamental Information Technology Engineers

NO. 3

INTERNAL DESIGN AND PROGRAMMING



Second Edition

REVISED AND UPDATED BY

IPA Japan Information-Technology Engineers Examination Center
INFORMATION-TECHNOLOGY PROMOTION AGENCY,JAPAN

Contents

Textbook for Fundamental Information Technology Engineers

No. 3 INTERNAL DESIGN AND PROGRAMMING

First edition first printed September 1, 2001
Second edition first printed August 1, 2002
Third edition first printed August 1, 2003

Japan Information-Technology Engineers Examination Center
INFORMATION-TECHNOLOGY PROMOTION AGENCY, JAPAN
Center Office 15F, Bunkyo Green Court 2-28-8, Hon-Komagome, Bunkyo-Ku,
Tokyo, 113-8663, JAPAN

1 Data Structures

Chapter Objectives

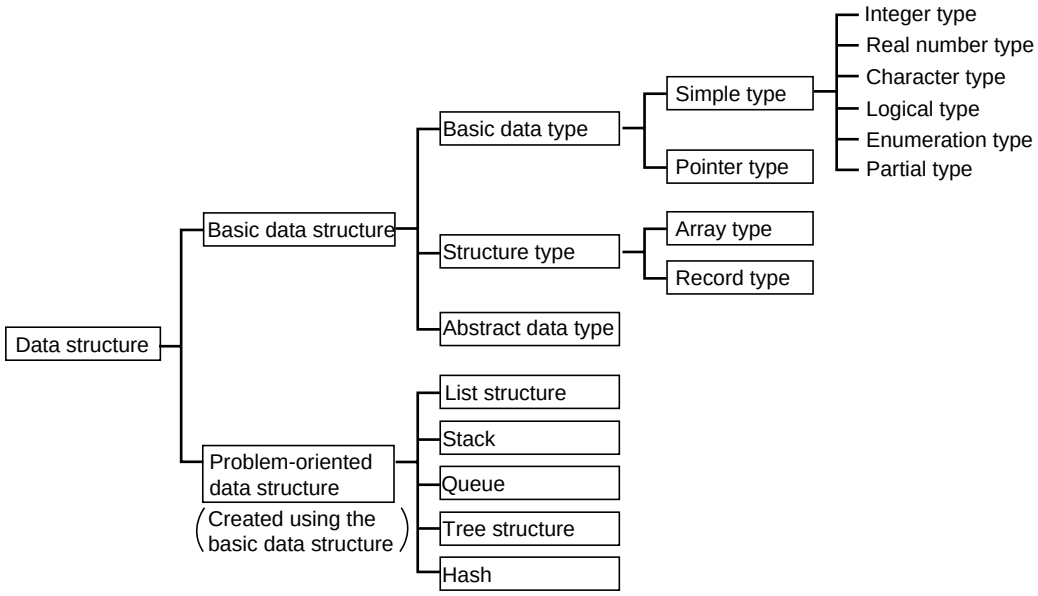
Choosing the most appropriate data structure and data description procedure is the key to creating an efficient, easy-to-understand program.
This chapter describes various data structures you must understand as the first step to learning programming.

① Understanding how various data structures can be classified
② Understanding the most commonly used, basic data types and data arrays
③ Understanding the characteristics and mechanisms of problem-oriented data structures used to solve specific problems, as well as how to use a basic data structure for program implementation

1.1 What is the data structure?

A set of the same kind of data processed by a computer is called a "data type." In the program design stage, the way data should be represented and programmed into a computer must be examined carefully, so that the most appropriate data type can be chosen. A data type that is represented and programmed is called a "data structure."
Figure 1-1-1 shows the classification of data structures.

Figure 1-1-1 Classification of data structures



The basic data structure can be represented in almost all programming languages. The problem-oriented data structure is a data structure that can be effectively used to solve specific problems. There are some problem-oriented data structures that cannot be represented in programming languages. In that case, the basic data structure is used.

1.2 Basic data structure

1.2.1 Basic data type

The basic data type is a set of individual data and is frequently used to create a program. It is classified into simple and pointer types.

(1) Simple type

The simple type is the most basic data type. When using the simple type for programming, the data type is usually declared according to the syntax rule of a language.

① Integer type

The integer type represents integers, and is represented inside a computer as binary numbers of fixed-point numbers that have no significant digits below the decimal point. A maximum or minimum value of the integer type is the unit of data that a computer can process at one time, and it is determined by the length of one word.

② Real number type

The real number type represents real numbers. It is used to represent fixed-point and floating-point numbers.

③ Character type

The character type represents alphabets, numerals and symbols as characters. A character code is expressed as a binary number inside a computer.

④ Logical type

The logical type is used to perform logical operations, such as AND, OR and NOT operations.

⑤ Enumeration type

The enumeration type is defined as a data type that enumerates all possible values of variables. In the case of the enumeration type, integers can be named.

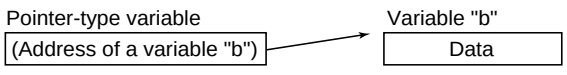
⑥ Partial type

The partial type is used to specify an original-value subset by constraining existing data types. The data type that has upper and lower limits specified as constraints is called the partial range type.

(2) Pointer type

The pointer type has an address allocated in a main memory unit. It is used to refer to variables, file records or functions. It is used for Pascal and C but not for FORTRAN and COBOL.

Figure 1-2-1 Image of the pointer type



1.2.2 Structured type

A data structure that contains a basic data structure or any of defined data types as its elements (data), is called the structured type. The structured type is classified into the array type and record type.

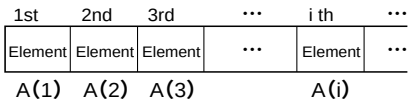
(1) Array type

An array is called a table. The array type is a data structure that contains data of the same type and size. Each individual data is called an array element, a table element or an element. How arrays are described or how data is arranged varies, depending on the programming language used.

① One-dimensional array

A one-dimensional array has a data structure in which data is arrayed in a line. To specify an element in this array, first enter a left parenthesis (or left bracket [after the name of an array, then enter a subscript and a right parenthesis) or right bracket]. A subscript, also called an index, indicates the ordinal number from the top of an array where a specified element is located. An array "A" having the number of elements denoted by "i" is expressed as A (i).

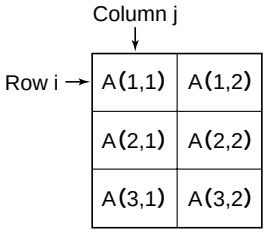
Figure 1-2-2 One-dimensional array



② Two-dimensional array

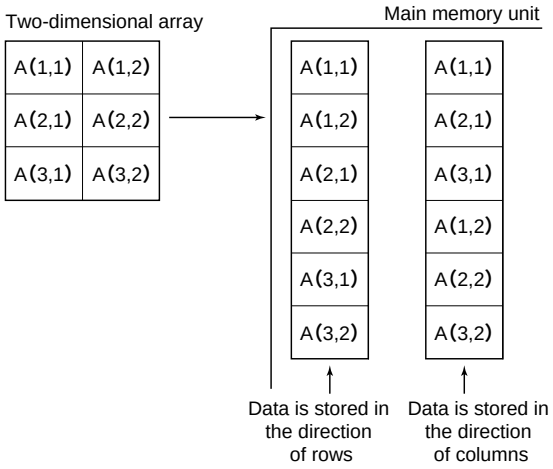
A data structure in which data is lined up in both vertical and horizontal directions is called a two-dimensional array. Data in a vertical direction is called a column and data in a horizontal direction is called a row. To specify a certain element in this array, two subscripts become necessary: one subscript showing in an ordinal number in a vertical direction (in what row) where a specified element is located and the other subscript showing in what ordinal number in a horizontal direction (in what column) it is located. An array "A" located in the "i" row and "j" column, for example, can be expressed as A (i, j).

Figure 1-2-3 Two-dimensional array (with three rows and two columns)



When a two-dimensional array is stored in a main memory unit, it takes the form of a one-dimensional array. The two-dimensional array shown in Figure 1-2-3 takes the form of a one-dimensional array having six elements. As shown in Figure 1-2-4, data is stored sequentially in either the direction of rows or the direction of columns. The direction in which data is stored varies, depending on the compiler of the programming language used. In general, data is stored vertically when Fortran is used and horizontally when COBOL is used.

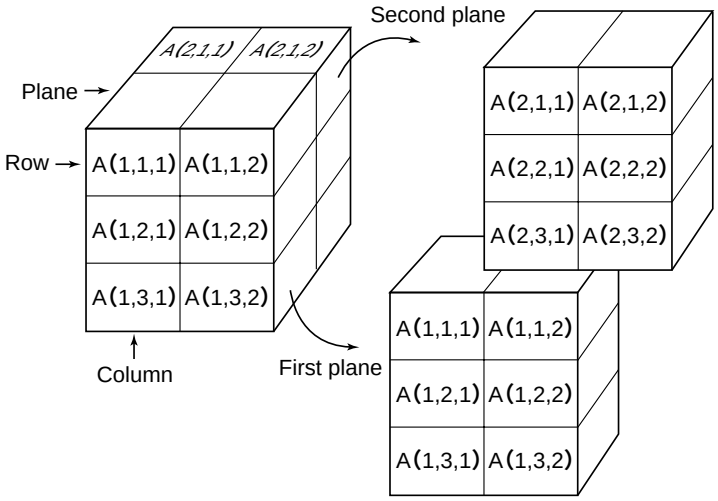
Figure 1-2-4 How data of a two-dimensional array is stored in a main memory unit



③ Three-dimensional array

A three-dimensional array has a data structure in which more than one two-dimensional array is contained. It has a three-dimensional structure comprising planes, rows and columns as elements. By developing a three-dimensional array into a two-dimensional one, a three dimensional array can be handled in the same way as a two-dimensional array.

Figure 1-2-5 Developing a three-dimensional array into a two-dimensional array



Multidimensional arrays such as four-, five-, or n-dimensional arrays can be defined. However, there may be certain limitations to the number of definable dimensions, depending on the type of programming language or compiler.

Arrays can be classified into static and dynamic arrays according to the method used to secure an area.

- Static array: An array for which a required area is determined by a program
- Dynamic array: An array for which a required area is determined after a subscript used for arraying is provided with an expression and the expression is evaluated during execution of a program

(2) Record type

Although structured type data is superior in the ease of element reference and operation, it has a drawback in that it can handle only data of the same type. Therefore, data that contains data of different types must take the form of record-type data. This record type is also called a structure.

Figure 1-2-6 Record type

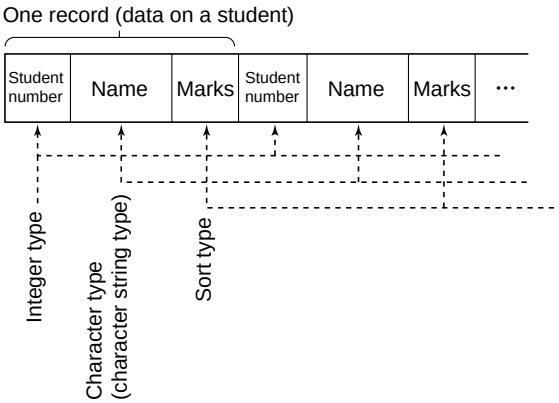
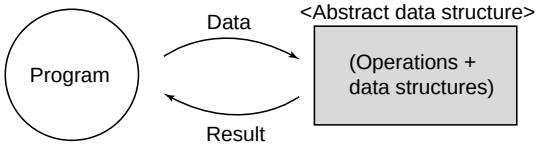


Figure 1-2-6 shows the data structure of the record type. One record contains a student number (integer type), a name (character type) and marks (integer type). One record type data contains a set of records of this format. Although one-dimensional record type data can be handled in the same way as a one-dimensional array, each data must be named for identification since each element contains more than one data.

1.2.3 Abstract data type

Data that contains a certain data structure and type of operations is called an abstract data type. To access this type of data, you do not need to be aware of its internal structure. All data are hidden except the data that you access for reference, insertion or deletion. This is called information hiding. Hiding information or hiding data on the level of data types is called data encapsulation.

Figure 1-2-7 Abstract data type



1.3 Problem-oriented data structure

Various problem-oriented data structures can be designed using the array type, pointer type and other basic data structures.

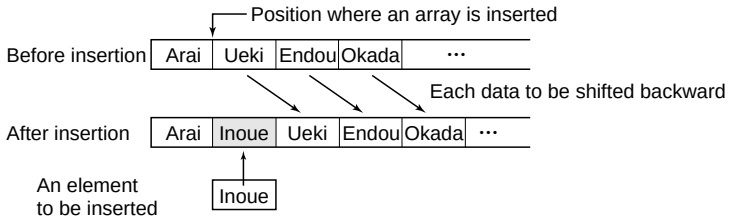
1.3.1 List structure

Unlike the basic data type that handles individual data, a list structure allows data to be linked to each other and handled in one lump. Data arranged according to this list structure is called a list.

(1) List structure and cells

Using a subscript for each element in an array, quick access to any element is possible. Also, a change to data can be made easily. If you insert one piece of data somewhere in arrays, you must shift each of all pieces of data subsequent to the inserted element backward. If you delete one piece of data in arrays, you must likewise shift each of all pieces of data subsequent to the deleted piece forward.

Figure 1-3-1 Inserting an array element

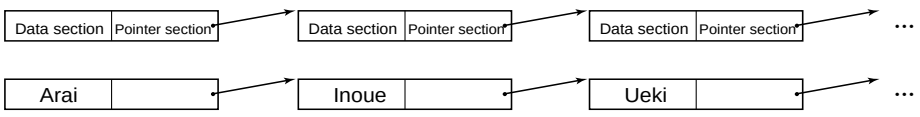


Unlike the array type structure, the list structure allows data element to be inserted or deleted easily. The list structure is similar to the array structure in that data elements of the same type is sequentially lined up. The array type requires that the logical arrangement of elements is the same with the physical arrangement of elements stored in a main memory unit. In the case of the list structure, the logical arrangement does not need to match the physical arrangement.
The list contains cells and each cell consists of the following:

- Data section that contains data element
- Pointer section that contains an address

Therefore, the data section of a cell has the same data structure as that of stored data and the pointer section of a cell has a pointer-type data structure. This means that cells represent record-type (structure) data that contain elements of different data structures. The list contains cell addresses in the pointer section and one cell is linked to another cell via a pointer.

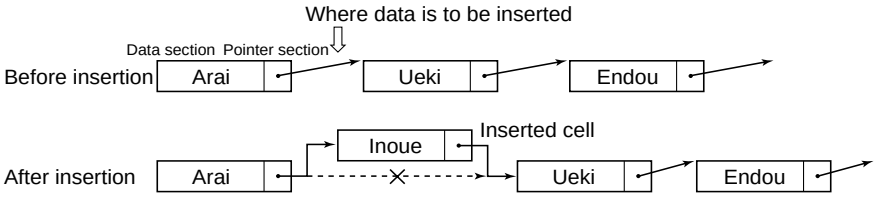
Figure 1-3-2 List structure



(2) Inserting data into a list

To insert data into a list, all you have to do is to replace pointers preceding and subsequent to data to be inserted. Because you do not have to shift elements as in the case of array-type data, you can insert data easily and quickly. (See Figure 1-3-3.)

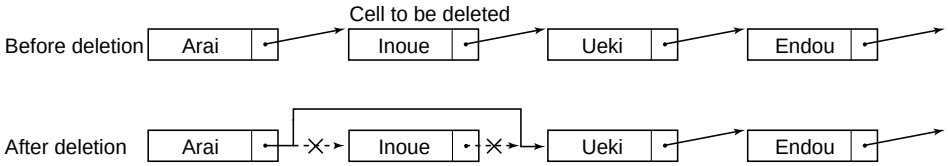
Figure 1-3-3 Inserting data into a list



(3) Deleting data from a list

To delete data from a list, all you have to do is to replace pointers as you do to insert data into a list. Cells that contain data (Inoue) remain in a memory area as garbage after data is deleted, as shown in Figure 1-3-4.

Figure 1-3-4 Deleting data from a list



Although the list structure allows data to be inserted or deleted by simply replacing pointers, it has the drawback that you must track each pointer one by one from the top one if you want to access specific data.

(4) Types of list structures

Representative list structures include:

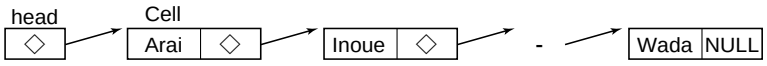
- Uni-directional list
- Bi-directional list
- Ring list.

Because these lists are represented in the form of a straight line, they are generically called linear lists.

① Uni-directional list

The uni-directional list is also called a one-way list. The pointer section of a cell contains the address of a cell in which the next data is stored. By tracking these addresses one by one, you can perform a search on data.

Figure 1-3-5 Uni-directional list

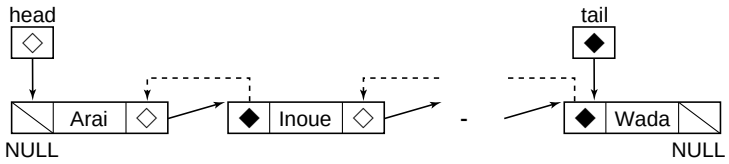


The first pointer is called a root or head. Because the pointer section of the last cell does not have any address in which data can be stored, NULL (numerical value of zero) or ＼is entered in this section.

② Bi-directional list

The bi-directional list has two pointer sections (◇and◆) which contain cell addresses as shown in Figure 1-3-6.

Figure 1-3-6 Bi-directional list

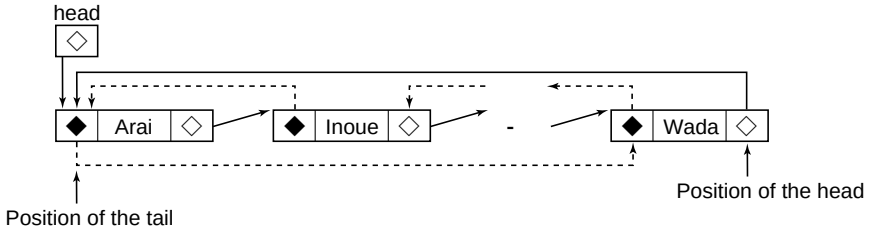


The pointer sections ◇ and ◆ shown in Figure 1-3-6 contain the address of the subsequent cell and the address of the preceding cell respectively. The address of the last cell is contained in the pointer tail. In the case of the bi-directional list, data can be tracked from either the head or the tail of cells.

③ Ring list

A bi-directional list containing NULL in the first cell is called a ring list. The pointer section of this first cell and the pointer section containing NULL of the last cell contain addresses of one another, thus data is in the form of a ring. Data can be searched in the same way as in the bi-directional list.

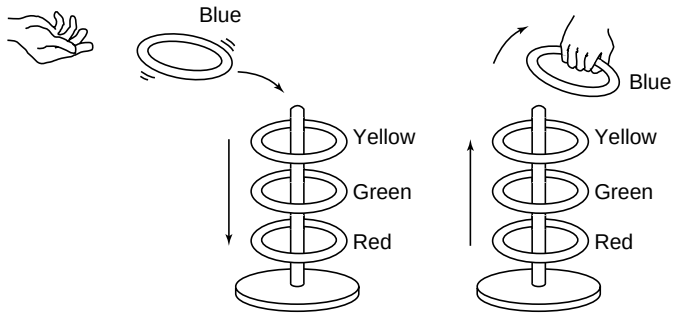
Figure 1-3-7 Ring list



1.3.2 Stack

A stack is a data structure designed based on one-dimensional arrays. Last stored data is read first. It is likened to the game of quoits.

Figure 1-3-8 Quoits

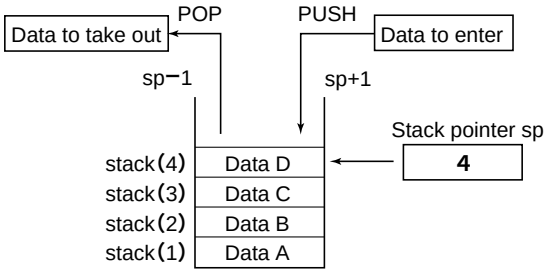


Quoits are thrown in the order of red, green, yellow and blue (data input). They are retrieved one by one (data output) in reverse order of throwing, i.e., blue, yellow, green and red. That is, a blue quoit last thrown is first retrieved.

This type of data structure which can be likened to the game of quoits is called a stack. This system is termed the last-in first-out (LIFO) system. Storing data in a stack is called "push down (PUSH)" and taking data from a stack is called "pop up (POP)." A variable that controls pushing down and popping up of data is called a stack pointer.

To push down data into a stack, set the stack pointer "sp" to +1 and store data in array elements indicated by "sp." To pop up data from a stack, take out data stored in array elements indicated by "sp" and set the stack pointer to sp-1.

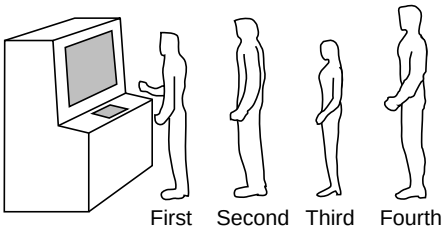
Figure 1-3-9 Stack structure



1.3.3 Queue

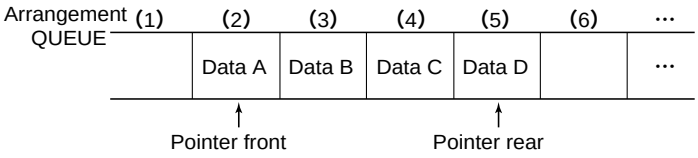
A queue is a data structure designed based on one-dimensional arrays. Data first stored is first read. It is likened to a queue of people who are waiting in front of a cash dispenser machine of a bank.

Figure 1-3-10 Queue



A data structure that allows customers to be served on the first-come first-served basis is called a queue. This system is called the first-in first-out (FIFO) system. Two pointers that indicate the head and tail of a queue are required for queue control. Pointers that indicate the head and tail of a queue are represented as a front variable and a rear variable respectively. (See Figure 1-3-11.)

Figure 1-3-11 Queue structure



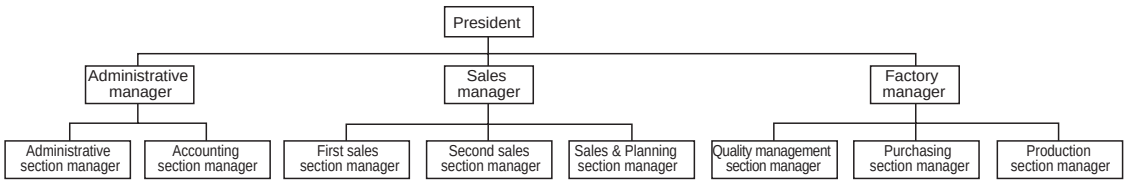
<Queue operation procedure>

1. To store data in a queue (enqueue), set a rear variable of a pointer indicating the tail to +1 and store data.
2. To take out data from a queue (dequeue), take out data and set a front variable of a pointer indicating the head to +1.

1.3.4 Tree structure

The tree structure is one of very useful data structures since it can control complex data better than array- or list-type data structures.
Because it has a hierarchical structure like a company organization or a family tree, it is suited for the type of work involving step-by-step classification.

Figure 1-3-12 Organization chart

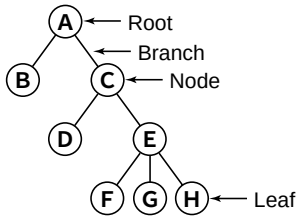


Although each cell is linearly correlated in the list structure, data is correlated while it branches off in the tree structure.
The tree structure consists of elements shown below:

- Node: Correspond to data
- Branch: Connecting one node to the other
- Root: Highest-order node that has no parent
- Child: Node that branches off under a node
- Parent: Original data before it begins to branch
- Leaf: Lowest-order node that has no child

Figure 1-3-13 shows the tree structure.

Figure 1-3-13 Tree structure



- A parent of the node C is the node A.
- Children of the node C are the nodes D and E.

(1) Binary tree

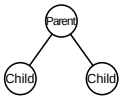
If the number of branches branching off from a node is "n" or less, that is, if the number of children is 0 to "n," such a tree structure is called an N-ary tree. An N-ary tree that has two branches (n=2), that is, an N-ary tree that has no child, one child or two children is called a binary tree, which is the most commonly used data structure. On the other hand, a tree structure that has three or more branches (n>2) is called a multiway tree.
Binary-tree data consists of one data section and two pointer sections. The left pointer section shows the position of a node that extends to the left and branches off while the right pointer section shows the position of a node that extends to the right and branches off.

Figure 1-3-14 Binary-tree data structure

Left pointer section	Data section	Right pointer section

The binary tree has a parent-child hierarchical structure, as shown in Figure 1-3-15.

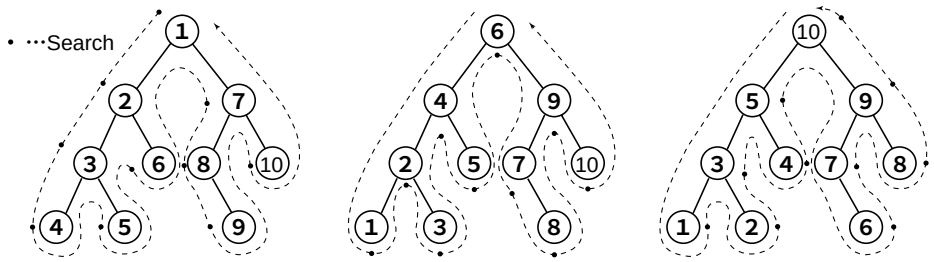
Figure 1-3-15 Basic binary-tree structure



<How to perform a search on binary-tree data>

To find specific data in binary-tree data, each node must be tracked one by one. There are three methods to perform a search on binary-tree data. As it is difficult to explain them in words, check Figure 1-3-16 and see how data is searched using each method.

Figure 1-3-16 How to perform a search on binary-tree data

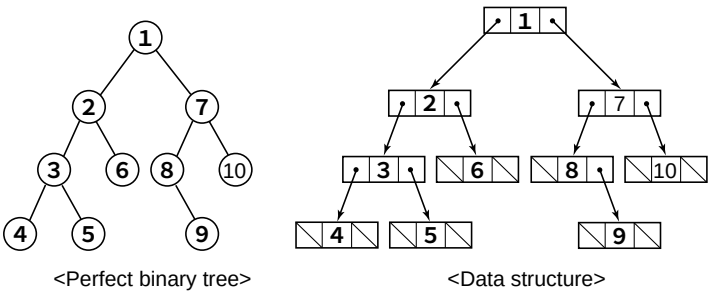


- Pre-order: With a root as a starting point, the left side of each node is tracked in a sequential way.
- Mid-order: With a leaf at the bottom left as a starting point, the underside of each node is tracked in a sequential way.
- Post order: With a leaf at the bottom left as a starting point, the right side of each node is tracked in a sequential way.

(2) Perfect binary tree

If a binary tree is constructed in such a way that the number of branches from a root to each leaf along one branch is equal to or different by one from that of branches from a root to each leaf along another branch (or if the height from a root to each leaf along one branch is equal to or different by one from that from a root to each leaf along another branch), it is called a perfect binary tree. Figure 1-3-17 shows a perfect binary tree that has ten nodes.

Figure 1-3-17 Perfect binary tree



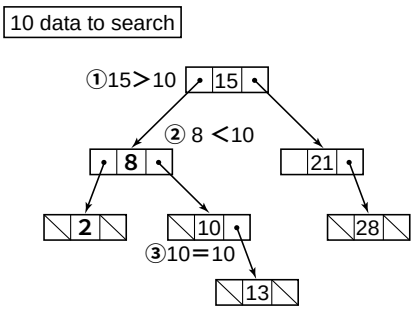
(3) Binary search tree

A binary search tree is used as a variant of a binary tree. In the case of a binary search tree, a descendant to the left is smaller than a parent and a descendant to the right is larger than a parent.
The algorithm of a binary search tree is as follows:

1. A root is a point where a search starts.
2. Binary-tree data is compared with data to search.
3. If binary-tree data = data to search, a search is successful (completed).

4. If binary-tree data > data to search, nodes to the left of a binary tree are searched and compared.
5. If binary-tree data < data to search, nodes to the right of a binary tree are searched and compared.
6. If no child is found, a search is unsuccessful (no data is found).

Figure 1-3-18 Performing a search on data in a binary search tree



(4) Balanced tree

If data is added or deleted in a tree structure, leaves randomly develop and the operation efficiency will decrease. A tree structure capable of reorganizing itself is called a balanced tree. Representative balanced trees are a B-tree and a heap.

① B-tree

A B-tree is a further developed version of a binary tree:

- End leaves are removed and each node has a number of nodes specified as "m."
- Each node has a maximum number of data specified as "m-1."
- All leaves are on the same level.
- Data contained in each node is arranged in a queue

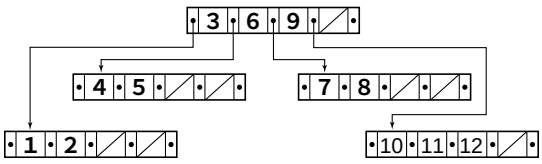
Because the above characteristics can increase the memory and calculation efficiency, the name "B-tree" means a well-balanced tree.

a. Characteristics of a B-tree

- To increase the memory area usage, the number of pointers that each node has is set at m/2 or more and m or less. The number of pointers along a route, however, is set at 2 or more.
- Each time data is deleted or added, splitting and concatenation are automatically executed so that the number of elements of a node can become m/2 or more or m or less. This allows leaves to be always maintained on the same level.
- A search starts from a root.
- Addition or deletion of data starts from a leaf.

Figure 1-3-19 shows a quint-B-tree with elements ｜7, 6, 10, 2, 5, 12, 4, 9, 8, 11, 1, 3｜.

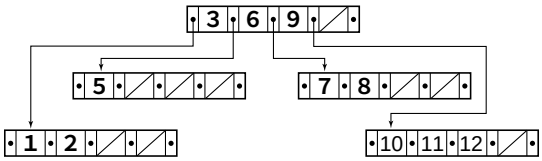
Figure 1-3-19 Quint-B-tree



b. Deleting data

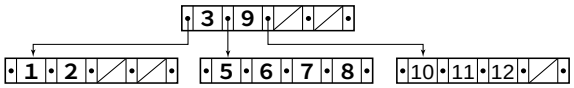
Figure 1-3-20 shows a case where the data "4" is deleted from a B-tree shown in Figure 1-3-19.
If "4" alone is deleted, the number of deleted elements of a node becomes one and the requirements for a B-tree cannot be met.

Figure 1-3-20 Wrong B-tree



The node from which "4" is deleted is merged with an adjacent node either to the right or to the left. Because the pointers of a parent must also be reorganized, "6" is moved to a lower-order node and each node can be reorganized, as shown in Figure 1-3-21.

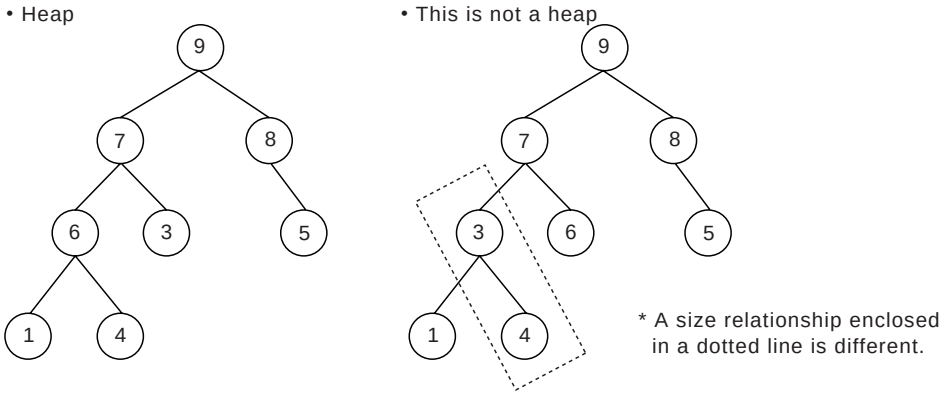
Figure 1-3-21 Correct B-tree



② Heap

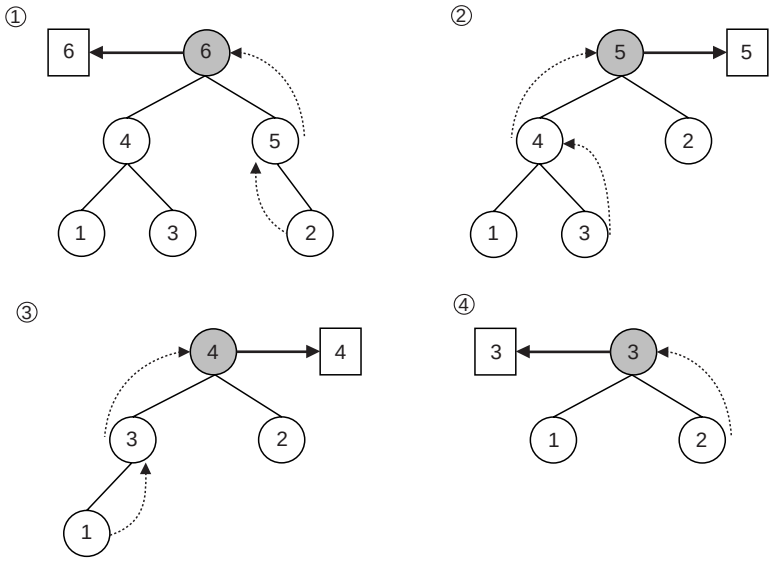
A perfect binary tree that has a certain size relationship between a parent node and a child node is called a heap. A heap is different from a binary search tree in that a heap does not have a certain size relationship between brothers.

Figure 1-3-22 Example of a heap



Because a heap has the structure of a perfect binary tree, it is a kind of organizable, balanced tree. In the case of a heap, maximum values (or minimum values) of all data are recorded in a root Using this characteristic, data can be sorted by taking out data from a root in a sequential way.

Figure 1-3-23 Sorting data using a heap



1.3.5 Hash

A hash is a way of using an array-type data structure. Using a hash, you can directly access specific data using a key without accessing recorded data one by one.

(1) Converting to a hash address

To convert a key to a hash address (subscript), a hash function is used. Representative hash functions are:

- Division method
- Registration method
- Base conversion method.

① Division method

A key is divided by a certain value (a prime number closest to the number of array elements is usually used) and the remainder is used as the address (subscript) of a key.

Example Key: 1234 and the number of array elements: 100
1234÷97 (a prime number closest to 100) = 12 70 : Address (subscript)

② Registration method

A key is decomposed according to a certain rule and the total is used as the address (subscript) of a key.

Example Key: 1234
1234 is decomposed into 12 and 34 and both numbers are totaled.
12 + 34 = 46 : Address (subscript)

③ Base conversion method

A key is normally represented in decimal numbers. This key is converted into a radix other than decimal numbers (tertiary numbers, for example) and the result is used as the address (subscript) of a key.

Example Record key: 1234

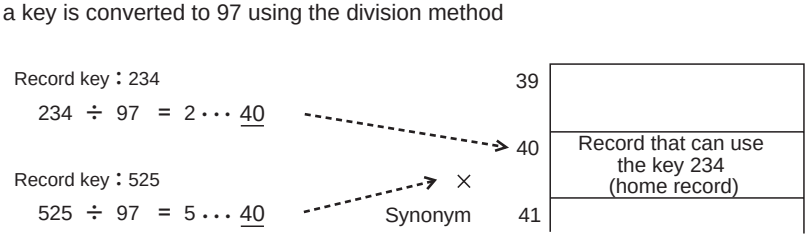
$$1 \times 3^3 + 2 \times 3^2 + 3 \times 3^1 + 4 \times 3^0 = 27 + 18 + 9 + 4$$
$$= 58 \quad \text{: Address (subscript)}$$

∗ The value of each digit of tertiary numbers is 3 or smaller. In this case, however, this fact does not need to be considered.

(2) Synonym

When a key is converted to an address using a hash function, different keys can be converted to the same address. This is called a synonym (collision) (see Figure 1-3-24). A record that can use the converted address is called a home record and a record that cannot use it is called a synonym record.

Figure 1-3-24 Occurrence of a synonym



To deal with a synonym, a sequential method or a chain method is used:

① Sequential method

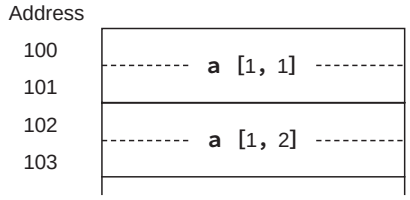
Using a sequential method, a synonym record is stored in free space that is located close to the desired address. There is the possibility that synonyms may occur in a chain reaction.

② Chain method

Using a chain method, a separate memory area is established and synonym records are stored in this area. In this case, it is necessary to provide a pointer that shows the address where synonym records are stored.

Exercises

Q1 **When storing a two-dimensional array "a" with ten rows and ten columns in continuous memory space in the direction of rows, what is the address where a [5, 6] is stored? In this question, the address is represented in decimal numbers.**



a. 145 b. 185 c. 190 d. 208 e. 212

Q2 **The figure below is a uni-directional list. Tokyo is the head of this list and the pointer contains addresses of data shown below. Nagoya is the tail of the list and the pointer contains 0. Choose one correct way to insert Shizuoka placed at address 150 between Atami and Hamamatsu.**

Pointer for the head
10

Address	Data	Pointer
10	Tokyo	50
30	Nagoya	0
50	Shin Yokohama	90
70	Hamamatsu	30
90	Atami	70
150	Shizuoka	

a. The pointer for Shizuoka to be set to 50 and that for Hamamatsu to be set to 150
b. The pointer for Shizuoka to be set to 70 and that for Atami to be set to 150
c. The pointer for Shizuoka to be set to 90 and that for Hamamatsu to be set to 150
d. The pointer for Shizuoka to be set to 150 and that for Atami to be set to 90

Q3 **What is the data structure suitable for the FIFO (first-in first-out) operation?**

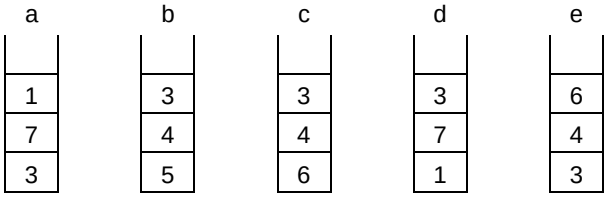
a. Binary tree b. Queue c. Stack d. Heap

Q4 **Two stack operations are defined as follows:**

PUSH n: Data (integer "n") is pushed into a stack.
POP: Data is popped out of a stack.

If a stack operation is performed on an empty stack according to the procedure shown below, what result can be obtained?

PUSH 1 → PUSH 5 → POP → PUSH 7 → PUSH 6 → PUSH 4 →POP → POP → PUSH 3



Q5 **Figure 2 is the array representation of a binary tree shown in Figure 1. What value should be put into the space "a"?**

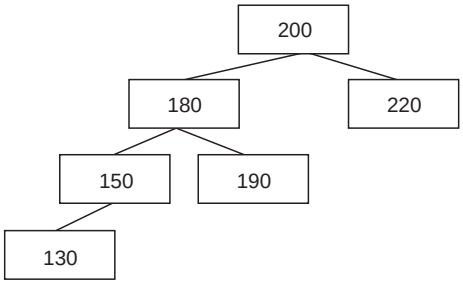


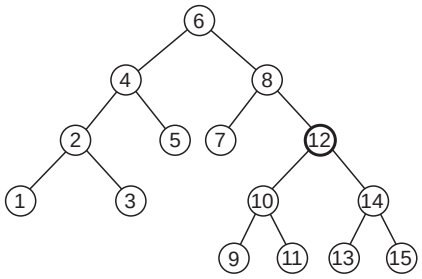
Figure 1 Binary tree

Subscript	Value	Pointer 1	Pointer 2
1	200	3	2
2	220	0	0
3	180	5	a
4	190	0	0
5	150	6	0
6	130	0	0

Figure 2
Array representation of a binary tree

a. 2 b. 3 c. 4 d. 5

Q6 **When the element 12 is deleted from the binary search tree shown below, which element should be moved to the point where the element was deleted to reorganize a binary search tree?**

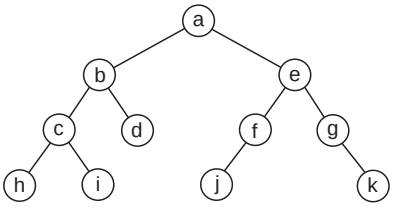


a. 9 b. 10 c. 13 d. 14

Q7 **If two or less than two branches branch off from each node of a tree, such a tree is called a binary tree. A binary tree consists of one node, a left tree and a right tree. There are three methods of performing a search on this tree:**

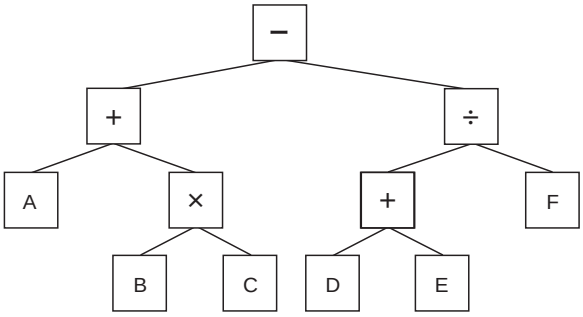
(1) Pre-order: A search is performed in the order of a node, a left tree and a right tree.
(2) Mid-order: A search is performed in the order of a left tree, a node and a right tree.
(3) Post-order: A search is performed in the order of a left tree, a right tree and a node.

If a search is performed using the post-order method, which can be output as the value of a node?



a. abchidefjgk b. abechidfjgk c. hcibdajfegk d. hicdbjfkgea

Q8 **A binary tree shown below can be represented using an arithmetic expression. Which expression is correct?**

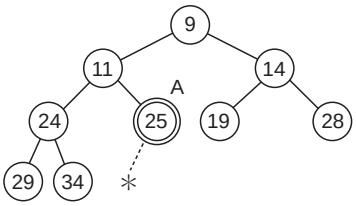


a. A + B × C + (D + E) ÷ F
b. A + B × C - (D + E) ÷ F
c. A + B × C - D + E ÷ F
d. A × B + C + (D - E) ÷ F
e. A × B + C - D + E ÷ F

Q9 **Regarding storing data using a B-tree, choose one correct comment from the following:**

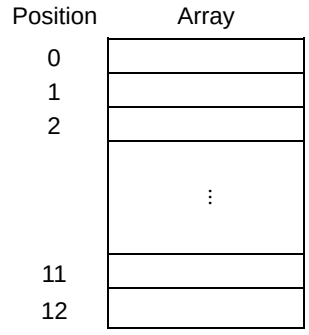
a. Split and merge nodes to allow the hierarchical depth to become the same.
b. Identify the position where data is to be stored by using a certain function and key value.
c. Only sequential access to head data and subsequent data is possible.
d. There are a directory and a member. A member is a sequentially organized file.

Q10 **There is a heap with the value of a parent's node smaller than that of a child's node. To insert data into this heap, you can add an element to the backmost part and repeat exchanging a parent with a child when the element is smaller than a parent. When element 7 is added to the position * of the next heap, which element comes to the position A?**



a. 7 b. 9 c. 11 d. 24 e. 25

Q11 **A five-digit number (a_1 a_2 a_3 a_4 a_5) must be stored in an array using the hash method. If the hash function is defined as mod (a_1+a_2+a_3+a_4+a_5, 13) and if the five-digit number is to be stored in an array element located in the position that matches a calculated hash value, in which position will 54321 be placed in an array shown below? Here, the mod (x, 13) value is the remainder when x is divided by 13.**



a. 1 b. 2 c. 7 d. 11

Q12 **Five data with key values distributed uniformly and randomly in the range of 1 to 1,000,000 must be registered in a hash table of 10 in size. What is the approximate probability that collisions occur? Here, the remainder obtained when a key value is divided by the size of a hash table is used as a hash value.**

a. 0.2 b. 0.5 c. 0.7 d. 0.9

2 Algorithms

Chapter Objectives

The basis of programming is algorithm design. In designing the logical structure of a program, it is important to use the most appropriate algorithm. Also, in choosing a program from a library, what algorithm to use is one of the most important criteria for choosing the most appropriate program.
This chapter describes the basics of algorithm design and representative algorithms.

① Understanding the basics of algorithms, i.e., algorithm definitions, design, relationships with data structures, representation methods, etc.
② Understanding the characteristic and meanings of representative search, sort, character string processing, and file processing algorithms.

Introduction

Using an efficient, easy-to-understand algorithm enables one to increase the execution speed and decrease the number of hidden bugs. An algorithm is one of the critical factors that determine the system performance. Also, the quality of an algorithm is used as criteria for choosing parts from a library.
This chapter describes the basics of algorithms used for module logic design and the algorithms used to solve typical problems. Choosing a well-known or generally used algorithm without fully analyzing given problems should be avoided. An algorithm that is the most appropriate for characteristics of problems must be chosen.
Evaluating algorithms themselves is also an important task. Data obtained by comparing plural algorithms on the basis of objective figures will greatly help you choose the most appropriate algorithm.
In this chapter, you learn the basics of algorithms so that you can design an easy-to-understand module.

2.1 Basics of algorithms

This section explains the following:
- Definition of an algorithm
- Relationships between an algorithm and a data structure

2.1.1 What is an algorithm?

(1) Definition of an algorithm

An algorithm is defined in the Japanese Industrial Standard (JIS) as follows:
A set of a limited number of clearly defined rules that are applied a limited number of times to solve problems.
This means that an algorithm is a set of rules (procedures) established to solve problems. To solve one problem, there are several routes that we can take. As a simple example, when we calculate a value Y times as large as X, we can take two approaches:
- Multiply X by Y
- Adding X Y times

In determining an algorithm, it is important to not only think out a procedure for solving a problem but also design and adopt an algorithm that can solve a problem effectively and efficiently.
Another important point to note is that an algorithm must stop (it must not run in an infinite loop). This matter will be taken up in section 2.3.2 Valuation by correctness.

(2) Algorithm and programming

An algorithm and programming are two sides of the same coin. Programming is describing data and algorithms in a programming language so that a computer can execute its given tasks.
In general, a program consists of algorithms and data and an algorithm consists of logic and control.
Programming can be classified into four different types according to how algorithms are viewed:
- Procedural programming
- Functional programming
- Logic programming
- Object-oriented programming

The most appropriate type of programming must be chosen with consideration of the characteristics of a problem.

① Procedural programming

Procedural programming is the most commonly used type of programming. This category includes the following programming languages, FORTRAN, COBOL, PL/I, Pascal, C, etc.
Characteristic

- A program is divided into modules to make a large, complicated program easy to understand. Coding is performed for each module.
- Structured theorems are introduced to programming (to be explained later).
- Structured figures are used to compile a program.

Because procedural programming is procedure (control)-oriented, there are the following restrictions:

- In addition to a procedure (control), variables (variable names, types, sizes, etc.) must be declared.
- Instructions are executed one by one in a sequential manner (parallel processing cannot be done).
- Comparisons and calculations must be made to solve a problem.

② Functional programming

Functional programming is function-oriented programming. It is used in the field of artificial intelligence (AI), basic calculation theories and other research tasks. LISP, among others, is a functional programming language.
Characteristic

- Unlike sequential-execution-type programming, expressions are built by nesting and they are replaced with results of calculations to execute a program.
- Recursive calls can be described easily.
- The level of affinity with parallel processing is high.
- A calculation process or procedure does not need to be considered.

③ Logic programming

A predicate based on facts and inferences is the base of logic programming. Prolog is an example of a logic programming language.
Characteristic

- Because facts are described based on the predicate logic, the process of programming is made easy.
- Inferences and logical calculations can be made easily.

④ Object-oriented programming

In object-oriented programming, a system is considered a group of objects. Languages include Smalltalk, C++, Java and others.

2.1.2 Algorithm and the data structure

An algorithm and the data structure are closely related to one another. The data structure determines the framework of an algorithm to a certain extent.

(1) Data structure

The data structure is defined as follows:
A procedure that a program follows to store data and perform given tasks.

① Basic data structure

Storing data means storing data in a main memory unit. In storing data in a main memory unit, the data type (data type, size, etc.) must be declared. The most basic data structure unit used to declare the data type is called a basic data structure. In performing this step of data-type declaration, data is manipulated using the name of data whose data type has been declared. (See Figure 2-1-1.)

② Problem-oriented data structure

The problem-oriented data structure built by combining basic data structures contains one or more of the following:

- List

- Stack
- Queue
- Tree structure.

These elements are determined by programming (designed algorithms). If data is processed in the order it is input, a queue is used. If data is processed in the reverse order of input, a stack is used.
Therefore, the contents of the problem-oriented data structure are determined by algorithms to a certain extent.

Figure 2-1-1 Data definition and procedure division in a COBOL program

Data definition

```
DATA                        DIVISION.
FILE                          SECTION.
FD  TOUGETU-FILE.
01   T-REC.
     02  T-HIZUKE       PIC  X(06).
         88  T-ENDF      VALUE  HIGH-VALUE.
     02  FILLER          PIC  X(34).
FD  RUISEKI-FILE.
01   R-REC.
     02  R-HIZUKE       PIC  X(06).
         88  R-ENDF      VALUE  HIGH-VALUE.
     02  FILLER          PIC  X(34).
FD  N-RUISEKI-FILE.
01  N-REC               PIC  X(40).
WORKING-STORAGE  SECTION.
01   T-COUNT           PIC  9(05).
01  R-COUNT           PIC  9(05).
01  N-COUNT           PIC  9(05).
*
```

Procedure division

```
PROCEDURE           DIVISION.
HEIGOU-SYORI.
     OPEN  INPUT      TOUGETU-FILE  RUISEKI-FILE
           OUTPUT  N-RUISEKI-FILE.
     INITIALIZE        T-COUNT  R-COUNT  N-COUNT.
     PERFORM  T-YOMIKOMI.
     PERFORM  R-YOMIKOMI.
     PERFORM  UNTIL  T-ENDF  AND  R-ENDF
        IF  T-HIZUKE  <  R-HIZUKE
          THEN    WRITE  N-REC  FROM  T-REC
                    PERFORM  T-YOMIKOMI
          ELSE    WRITE  N-REC  FROM  R-REC
                    PERFORM  R-YOMIKOMI
        END-IF
        COMPUTE    N-COUNT  =  N-COUNT  +  1
     END-PERFORM.
     DISPLAY  T-COUNT  R-COUNT  N-COUNT.
     CLOSE  TOUGETU-FILE  RUISEKI-FILE  N-RUISEKI-FILE.
     STOP  RUN.
T-YOMIKOMI.
     READ  TOUGETU-FILE
        AT  END        MOVE  HIGH-VALUE  TO  T-HIZUKE
        NOT  AT  END  COMPUTE  T-COUNT  =  T-COUNT  +  1
     END-READ.
R-YOMIKOMI.
     READ  RUISEKI-FILE
        AT  END        MOVE  HIGH-VALUE  TO  R-HIZUKE
        NOT  AT  END  COMPUTE  R-COUNT  =  R-COUNT  +  1
     END-READ.
```

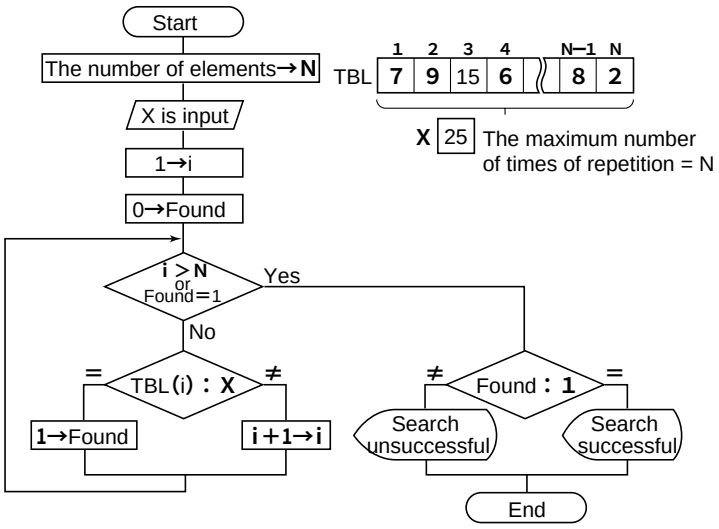
(2) Relationships between an algorithm and the data structure

The relationships between an algorithm and the data structure can be described as follows:

① Array processing

Take a linear search algorithm (details explained in Section 2.2.1) shown in Figure 2-1-2 for example.

Figure 2-1-2 The linear search algorithm and the data structure

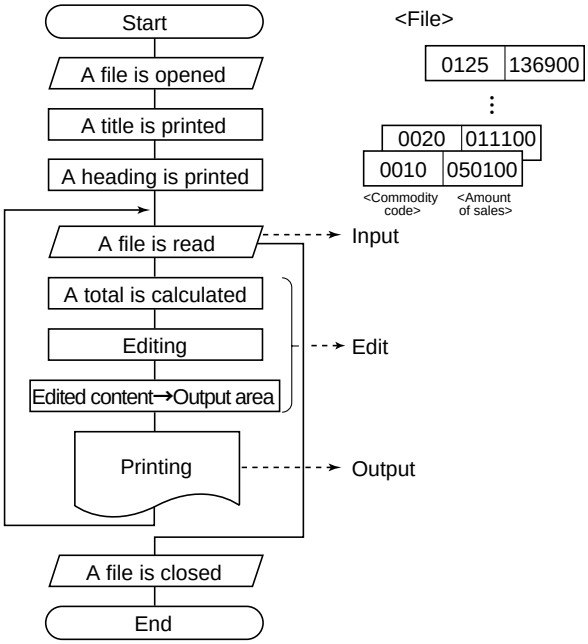


The linear search is most frequently used to search data. In performing the linear search on data, data is searched while subscripts in an array are being incremented. The maximum number of times the search is repeated is the size of an array. That is, if an array data structure is used, the procedure and the number of times of repetition are determined.

② File processing

Take an algorithm for reading and printing a file shown in Figure 2-1-3 for example. (Details are explained in section 2.2.5.)

Figure 2-1-3 Algorithm for processing a file and the data structure



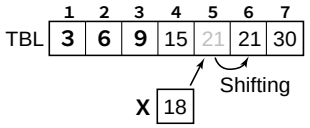
The process of reading, editing and printing a file is repeated until there is no data in a file.

Because a file must be accessed in each process cycle, this task is executed in the form of a loop.

③ List processing

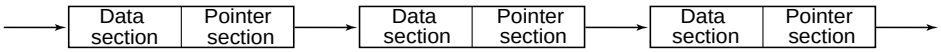
In processing an array structure, data can be searched and updated smoothly but it takes time to insert or delete data. Because data is arranged in queues, inserting or deleting data is inevitably accompanied by the shifting of related data backward or forward.

Figure 2-1-4 Inserting data into an array



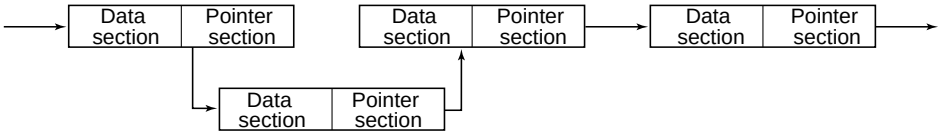
Using the list structure, inserting or deleting data is easy. Although data is arranged in an orderly manner in the list structure, it is arranged so in terms of logic since it does not need to be physically arranged in sequential order.

Figure 2-1-5 List structure



In the case of the list structure, data does not need to be shifted as shown in Figure 2-1-6; data can be inserted or deleted by simply manipulating a pointer.

Figure 2-1-6 Inserting data into the list structure



2.2 Various algorithms

An algorithm should be thought out to solve each specific problem. If an algorithm that can solve similar problems is already designed and made available, using such an algorithm will enable you to create a better algorithm in an efficient manner.
This section describes the representative algorithms that have until now been developed in relation to each type of problem.

2.2.1 Search algorithm

A table search is the method of performing a search on tables and files stored in a memory unit to find the elements that meet specified requirements.
This section describes two table search methods: linear (or sequential) search and binary search methods.

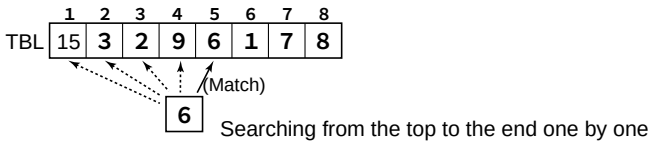
(1) Linear (or sequential) search method

A linear (or sequential) search method is a very simple search method of searching from the top of a table in a sequential manner.

① Exhaustive search method

An exhaustive search method is the simplest method of combing a table from the top to the end.

Figure 2-2-1 Image of the exhaustive search method

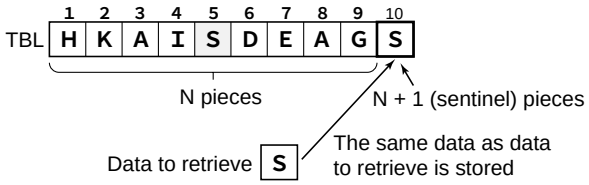


Data retrieved is collated with each data in a table. The algorithm is designed such that the search is successful if there is data that matches data to retrieve and it is unsuccessful if there is no data that matched data to retrieve.
The search procedure ends when the first successful match occurs. Therefore, this search method is inappropriate for counting the number of data that matches data to retrieve.

② Sentinel search method

The sentinel search method uses the algorithm in which the same data (sentinel) as the data to retrieve is placed at the end of a table to simplify the search algorithm and to decrease the number of times data is compared.

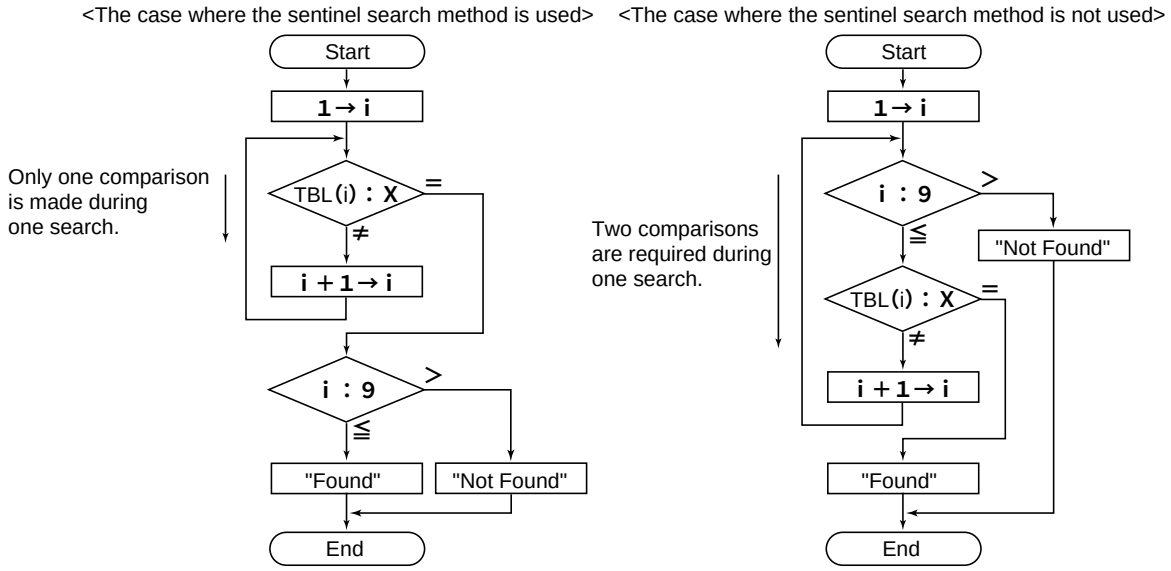
Figure 2-2-2 Sentinel search method



If the number of data contained in a table is N, the same data (sentinel) as the data to retrieve is stored in the (N + 1) position so that the data to retrieve can be instantly collated with the sentinel data.

Figure 2-2-3 shows a comparison of the case where the sentinel search method is used and the case where it is not used.

Figure 2-2-3 A comparison of the case where the sentinel search method is used and the case where it is not used



<The case where the sentinel search method is used>

In the first round of data collation, data to retrieve is authenticated. In the second round, it is determined whether there is data that matches data to retrieve. That is, if the number of data is N, a comparison is made only (N + 1) times.

<The case where the sentinel search method is not used>

In one round of data collation, the authenticity of data to retrieve as well as whether data to retrieve ends must be determined. That is, a comparison is made (N × 2) times.

<Maximum number of times a comparison is made if the number of elements is N:>

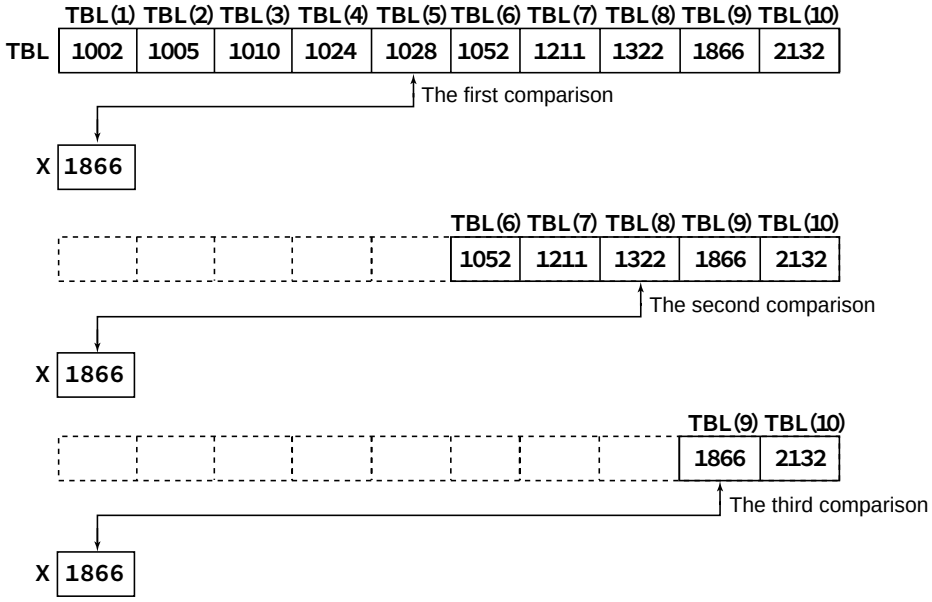
- (N + 1) times if the sentinel search method is used
- (N × 2) times if the exhaustive search method is used

(2) Binary search method

A binary search method is the method of narrowing down target data while successively dividing a search area into two parts. The number of comparisons can be greatly decreased compared with the linear search method and the search efficiency can equally improve. This search method, however, requires that elements are arranged in ascending or descending order.
Figure 2-2-4 shows the algorithm used for the binary search method.

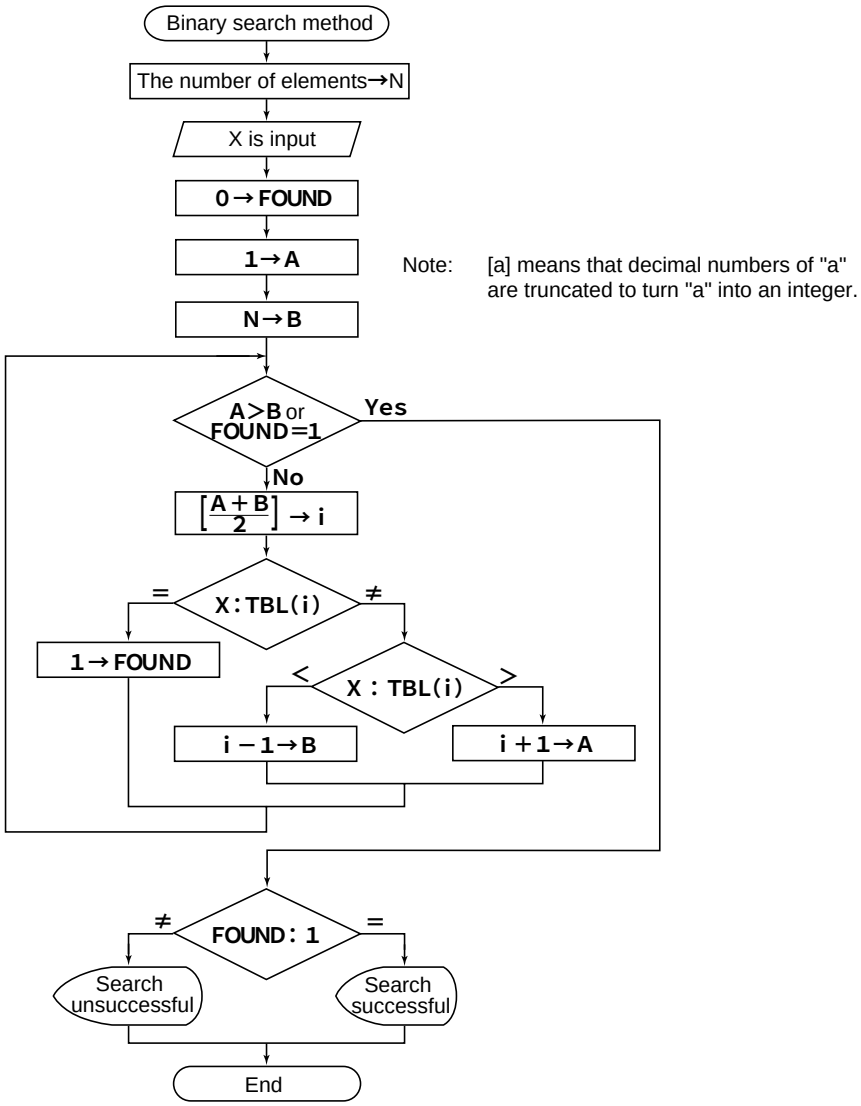
Figure 2-2-4 Algorithm for the binary search method

Step 1:	A total of the value of a subscript representing the top of a table and that of a subscript representing the end of a table is divided by 2.
Step 2:	The elements having the value obtained in step 1 as a subscript are compared with a target element.
Step 3:	If there is an element that matches a target element, the search is successful.
Step 4:	If the value of a target element is smaller than that of an element in a table, 1 is subtracted from the current subscript and the value is used as a subscript for representing the end of a table. If the value of a target element is larger than that of an element in a table, 1 is added to the current subscript and the value is used as a subscript for representing the top of a table.
Step 5:	Step 1 through step 4 is repeated. If an element matching a target element cannot be found at the point where the value of a subscript representing the top of a table becomes larger than that of a subscript representing the end of a table, the search is unsuccessful. This completes the search.



Because elements are arranged in ascending or descending order, data smaller than reference data does not need to be searched if data being searched is larger than reference data. Therefore, the number of data to search can be halved after the first search--a great advantage in the search efficiency.
Figure 2-2-5 shows a flowchart of the binary search method.

Figure 2-2-5 Flowchart of the binary search method



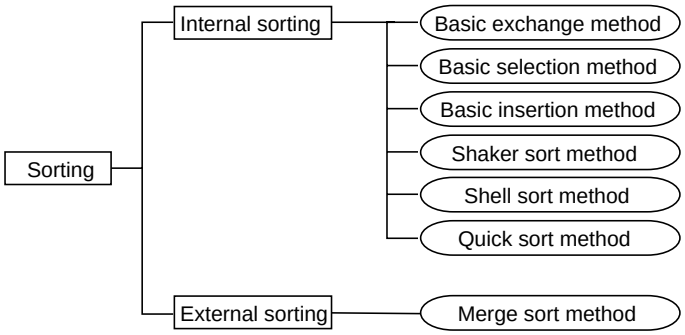
<Maximum and average number of times a comparison is made: If the number of elements is N>

- Average number of times = $[\log_2 N]$
- Maximum number of times-$[\log_2 N]$ +1
  ([ ] is a Gaussian symbol and decimal numbers of the value shown in this symbol are truncated.)

2.2.2 Sort algorithm

Sorting data is reorganizing data in a specified order. Sorting data in the order of small to large values is called ascending order sorting, and sorting data vice versa is called descending order sorting.
If data is sorted or organized in a certain specified order (amounts of sales, commodity codes, etc.), the efficiency of the work of data processing can be increased. Therefore, the sort algorithm is one of the most commonly used algorithms. Using this sort algorithm, one can sort numbers and characters. This type of sorting is possible because characters are represented as character codes (internal codes) in a computer.
Figure 2-2-6 shows various sort methods.

Figure 2-2-6 Sort methods



Internal sorting means sorting data in a main memory unit. External sorting means sorting data stored on a magnetic disk and other auxiliary storage unit.
In choosing one sort method, the speed of sorting, as well as how data is sorted, must be considered to minimize the time needed for data comparison and exchange.
In choosing one sort method, you must also clarify the time of calculation (computational complexity). This point will be described in detail in Section 2.3.

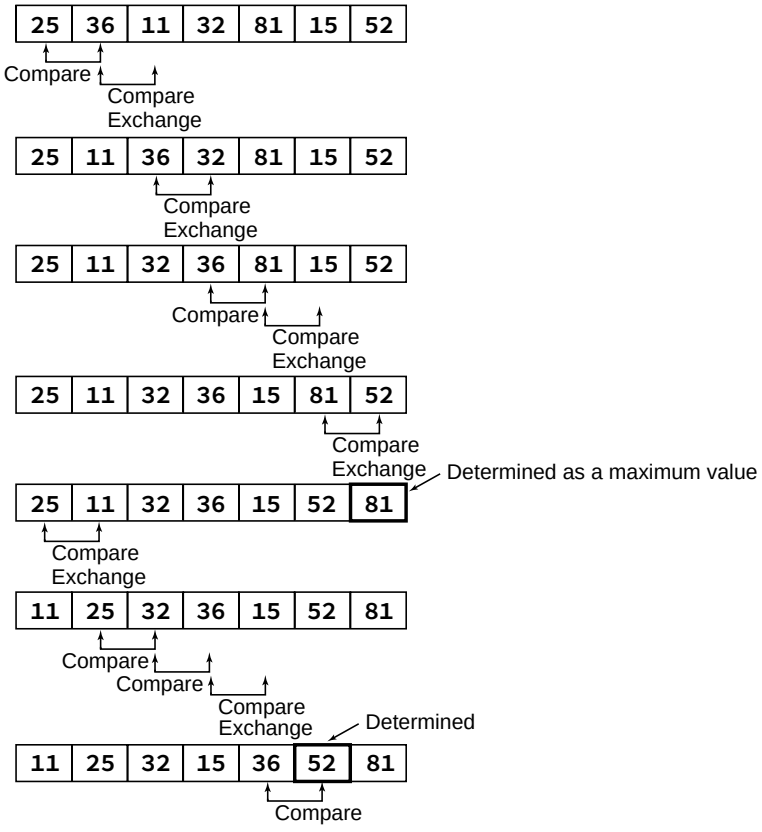
(1) Basic exchange method (bubble sort)

The basic exchange method (bubble sort) is used to compare a pair of data sequentially from the head of an array. If the sequence is wrong, data is exchanged. When all data items in one array are compared, the routine returns to the head of an array and it is repeated until there are no remaining data items to exchange. This method is the simplest, best known sort method. The name "bubble sort" was given since the movement of maximum or minimum data looks like bubbles popping out on the surface of the water.
Figure 2-2-7 shows the algorithm of the basic exchange method.

Figure 2-2-7 Steps of the basic exchange method

Algorithm for sorting data in the ascending order
- Step 1: The first and second elements in a table are compared.
- Step 2: If the first element is larger than the second, the first is exchanged with the second.
- Step 3: If the second element is larger than the first, no exchange occurs.
- Step 4: The second and third elements are compared and steps 2 and 3 are repeated.
- Step 5: This routine operation is repeated to the last element in a table. As it reaches the last element, a maximum value is stored in the last element in that table.
- Step 6: Steps 1 through 4 and step 5 are executed until the operation reaches the last element but one.
- Step 7: Steps 1 through 6 are repeated until only the first and second elements in a table remain.

This completes data sorting.



All these steps are repeated as one cycle.

<Characteristics>

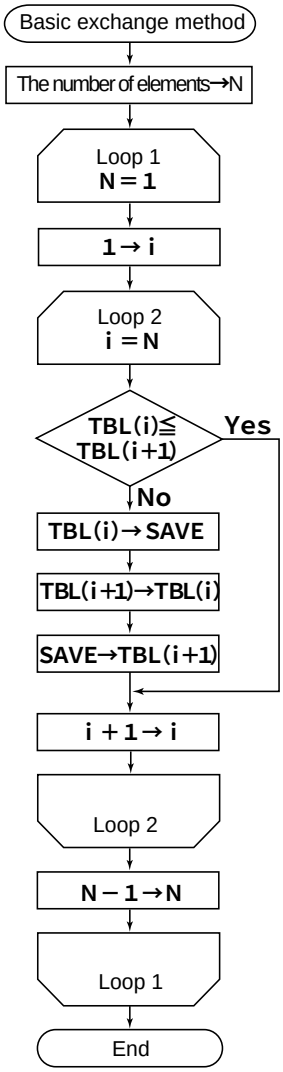
- One of the simplest sort methods
- The efficiency is low since data item are unconditionally compared even if they are sorted correctly.
- If the volume of data is large, the process is time-consuming.

<Computational complexity>

- Maximum computational complexity: $O(n^2)$
- Average computational complexity: $O(n^2)$

Figure 2-2-8 shows the flowchart of the basic exchange method.

Figure 2-2-8 Flowchart of the basic exchange method

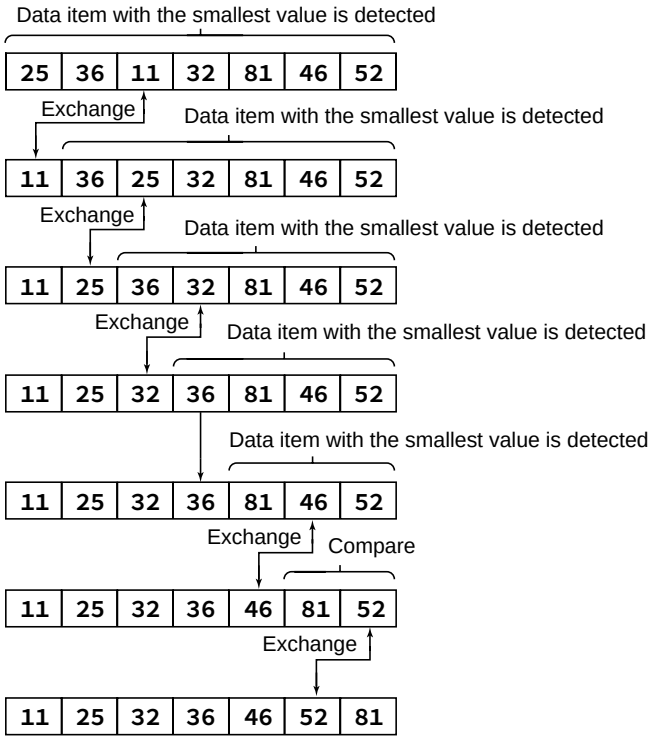


(2) Basic selection method

In the sort algorithm of the basic selection method, a data item with the smallest (or the largest) value is first selected from all the data items and it is exchanged with the data item at the head of an array, then the same routine is repeatedly executed on all the remaining data items. When the correct data item enters the last position but one, data sorting is completed.
Because this method allows a remaining element with the smallest or largest value to be selected, it is called the basic selection method.
Figure 2-2-9 shows the algorithm of the basic selection method.

Figure 2-2-9 Steps of the basic selection method

Steps of the algorithm for sorting data in the ascending order
- Step 1: Data item with the smallest value is detected in data items stored in a table.
- Step 2: Data item with the smallest value detected is exchanged with the first data item in a table (space into which the data item with the smallest value can be temporarily saved is required).
- Step 3: Data item with the smallest value is detected in the second to the last data item in a table.
- Step 4: Data item with the smallest value detected is exchanged with the second data item in a table.
- Step 5: The same operation is repeatedly performed until the last data item but one is reached. When the last data item but one is reached, data sorting is completed.



<Characteristics>

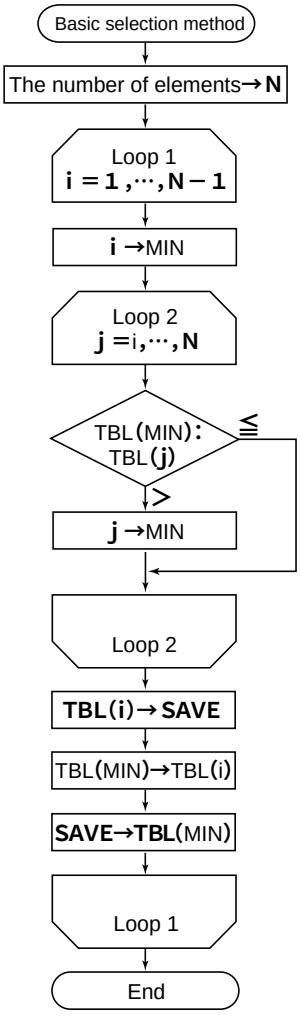
- One of the simplest sort methods, like the basic exchange method
- The efficiency is low since data items are unconditionally compared even if they are sorted correctly.
- If the volume of data is large, the process is time-consuming.

<Computational complexity>

- Maximum computational complexity: $O(n^2)$
- Average computational complexity: $O(n^2)$

Figure 2-2-10 shows the flowchart of the basic selection method.

Figure 2-2-10 Flowchart of the basic selection method



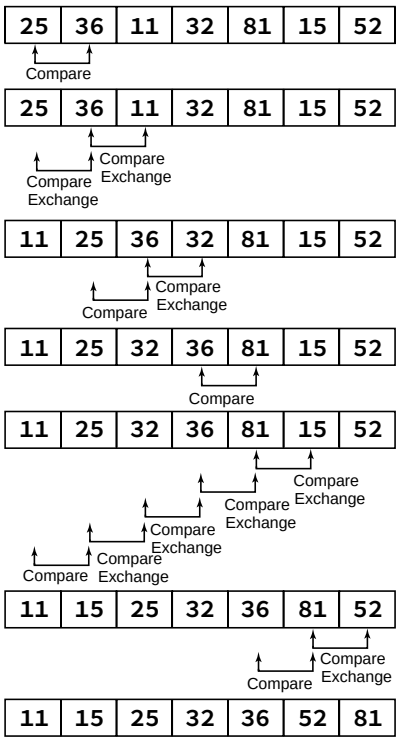
(3) Basic insertion method

In the algorithm of the basic insertion method, while data items are being sorted, an unsorted data item is inserted into a proper position in the sequence of sorted data items.
Figure 2-2-11 shows the algorithm of the basic insertion method.

Figure 2-2-11 Steps of the basic insertion method

Algorithm for sorting data in the ascending order
- Step 1: The first and second elements in a table are compared.
- Step 2: If the first element is smaller than the second, nothing is done.
- Step 3: If the second element is smaller than the first, the first element is exchanged with the second. At this point, the first and second elements are in the correct order.
- Step 4: The second and third elements are compared.
- Step 5: If the second element is smaller than the third, nothing is done.
- Step 6: If the third element is smaller than the second, the second element is exchanged with the third. Then this element is compared with the preceding element according to steps 2 and 3. These steps are repeated till it is placed in the right position.
- Step 7: Steps 4, 5 and 6 are repeated until the last element in the table is inserted into the correct position.

This completes data sorting.

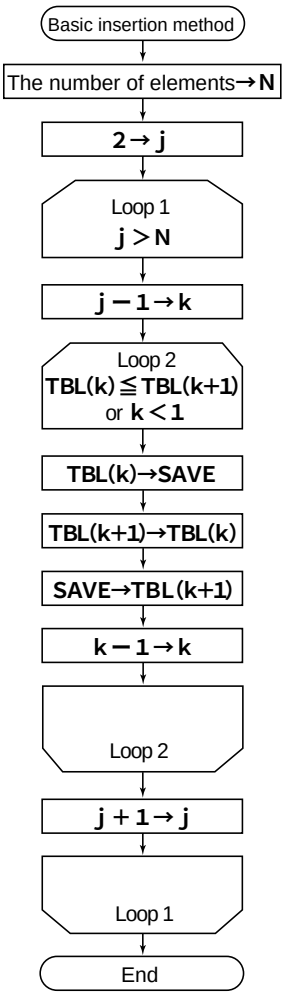


<Characteristics>

- One of the simplest sort methods, like the basic exchange method
- Because preceding data items have been sorted, comparison and insertion speeds are fast.
- If the volume of data is large, the process is time-consuming.

Figure 2-2-12 shows the flowchart of the basic insertion method.

Figure 2-2-12 Flowchart of the basic insertion method



(4) Shaker sort method

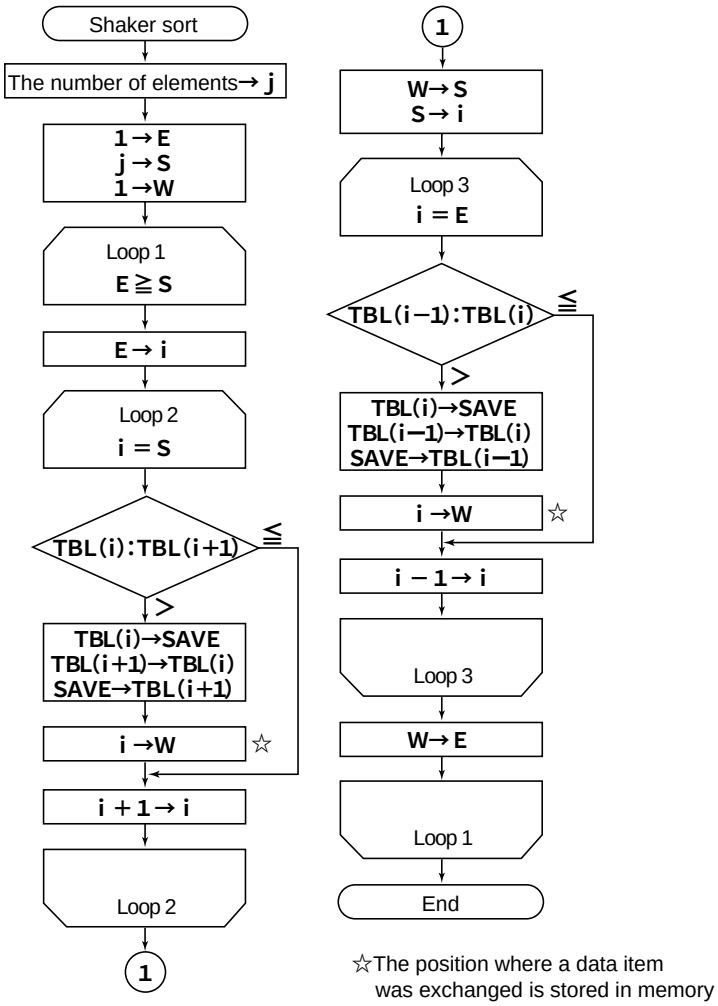
The algorithm of the Shaker sort method is basically the same as that of the basic exchange method (bubble sort). In the algorithm of the basic exchange method, data items are compared from left to right and are sorted in such a way that a maximum (minimum) value is set at the rightmost position. In the algorithm of the Shaker sort method, however, data items are first compared from left to right, then, after a maximum (minimum) value is set at the rightmost position, data items are compared from right to left and a minimum (maximum) value is set at the leftmost position; this operation is repeatedly performed to sort data.

<Characteristic>

- If the volume of data is large, the process is time-consuming.

Figure 2-2-13 shows the flowchart of the Shaker sort method.

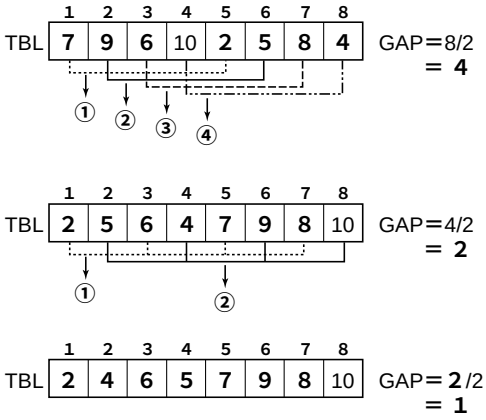
Figure 2-2-13 Flowchart of the Shaker sort method



(5) Shell sort method

The Shell sort method is an extended version of the basic insert method. Two items of data located away from each other at a certain interval are picked out of a data sequence and sorting is executed while the picked data items are compared with each other. The interval is called a gap. This gap is set large at the start of sorting; as sorting proceeds, it is gradually made smaller and finally set to 1. There are various ways of determining this gap. If the number of data is n, a simple way of determining the gap is [n/2], [n/4],··· 1. When the gap is finally set to 1, sorting is executed in exactly the same way as the basic insertion method.
Using the basic insertion method, adjacent elements are compared and exchanged and, therefore, the execution speed is slow. Using the Shell sort method, pieces of data that are away from each other and located in different positions are quickly exchanged so that data items sorted in wrong positions can be resorted to correct positions in the earliest stages of the sorting operation. As sorting proceeds, the gap between data items to be compared is narrowed.
Figure 2-2-14 shows the algorithm of the Shell sort method.

Figure 2-2-14 Steps of the Shell sort method



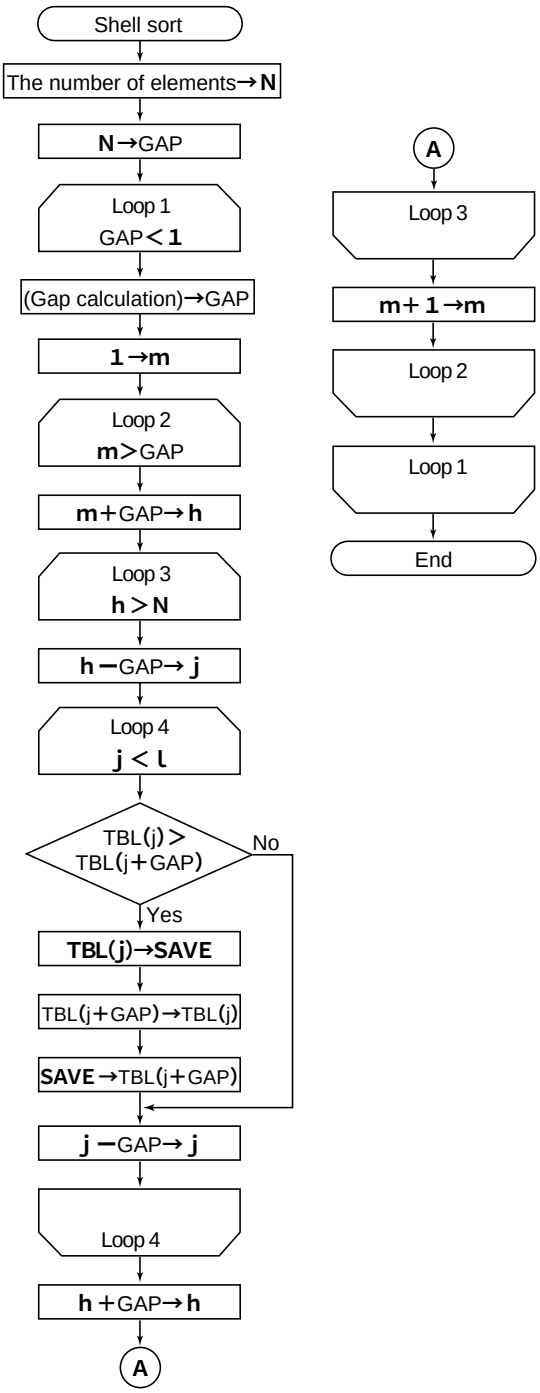
<The basic insertion method is used in this step.>

<Characteristics>

- If part of the data is already sorted, sorting can be completed very quickly.
- A high-speed sorting method that uses the basic insertion method

Figure 2-2-15 shows the flowchart of the Shell sort method.

Figure 2-2-15 Flowchart of the Shell sort method



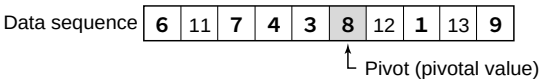
(6) Quick sort method

The quick sort method was designed by Hoare. It is presently the fastest sort method using the recursive call method.

<To sort data in the ascending order>

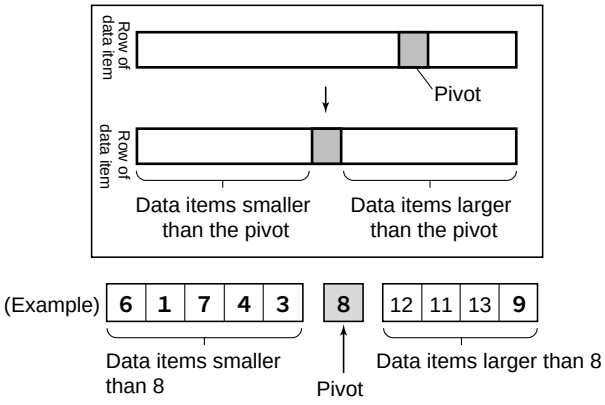
1. A reference value (pivot or pivotal value) is chosen from the data to be sorted. Although various values are used as reference values, a median value or an intermediate value of three elements (right, center and left elements) is usually used. We use a median value in the example of sorting shown below.

Figure 2-2-16 Pivot (pivotal value)



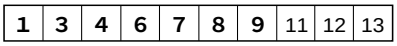
2. Data items smaller than the pivot are moved to the left of the pivot while data items larger than the pivot are moved to the right of it. All the data items are thus divided into two sections (division).

Figure 2-2-17 Moving data



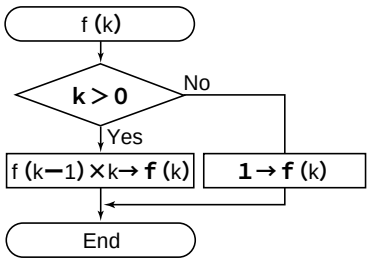
3. A pivot is chosen from each section of data items so that the section is further divided into two sections.
4. This dividing operation is repeatedly performed until only one element remains.

Figure 2-2-18 Data after sorting is completed



A method of dividing a large problem into small problems and solving each small problem individually, like the quick sort method, is called the divide-and-conquer method.
In performing steps 3 and 4 above, the recursive call method of calling the routine itself is generally used. This recursive call method is often used to calculate factorials. (See Figure 2-2-19.)

Figure 2-2-19 Algorithm for calculating factorials



With regard to computational complexity, if an ideal case in which a pivot that can divide data into two sections can always be chosen, the number of comparisons will be very close to O ($n$log$n$). If a maximum (minimum) value in a data sequence is always chosen as a pivot, the number of comparison will become the worst, O (n^2). Although computational complexity usually indicates maximum computational complexity, a case like this may happen in which average computational complexity becomes an important indicator.
After data items are divided into two sections, data items in each section to be subjected to a recursive routine can be processed individually. Therefore, parallel processing can be performed. The average processing time is O (log$n$) if parallel processing is executed.
Figure 2-2-20 shows the algorithm for performing quick sorting with a pivot set at the rightmost position of an array.
Specifically, two pointers are used and the routine proceeds from both ends to the center of a data sequence. A pointer moving from the left end to the right is "i" and one moving from the right end to the left is "j."

Figure 2-2-20 Algorithm of the quick sort method

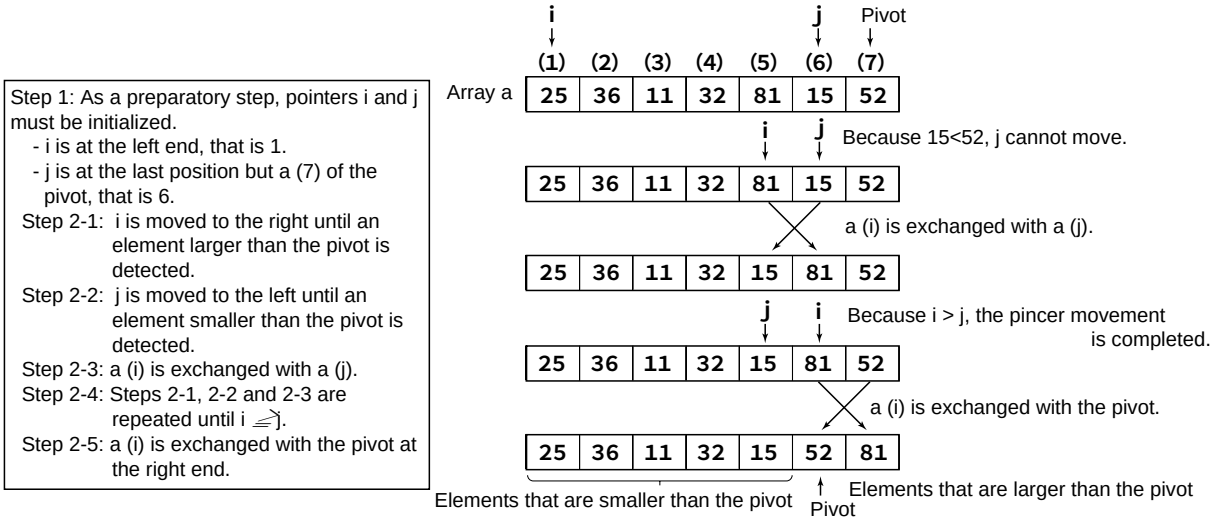
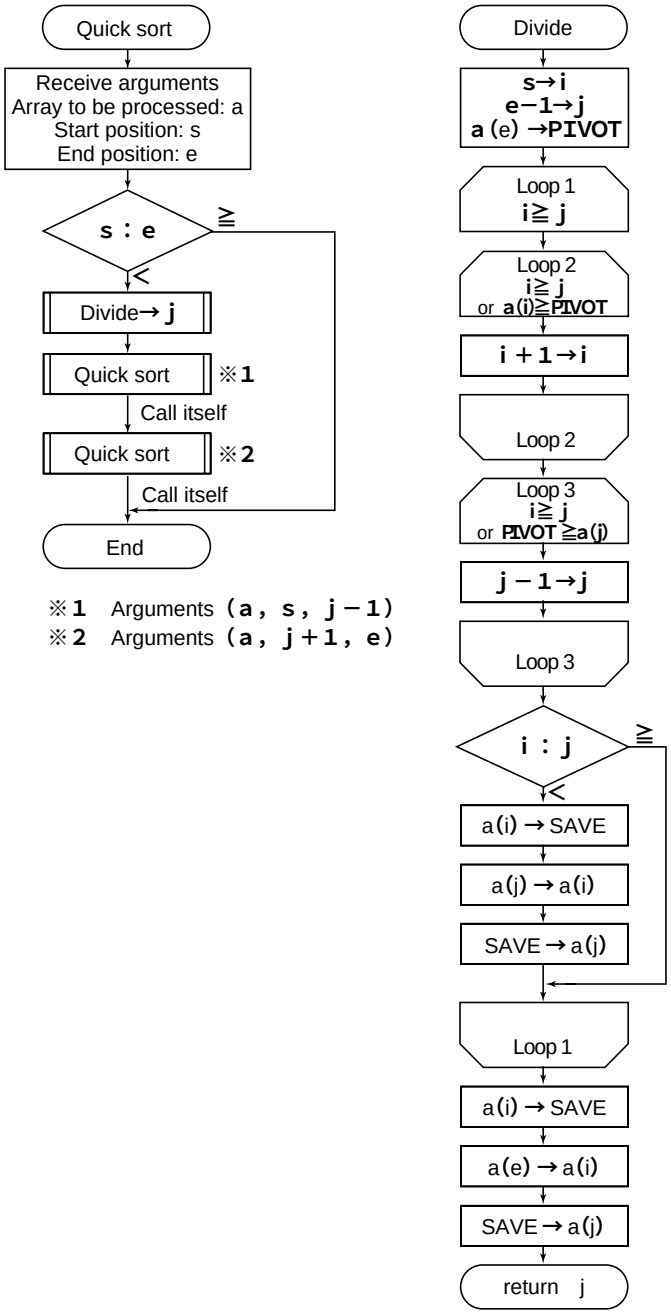


Figure 2-2-21 shows the flowchart of the quick sort method.

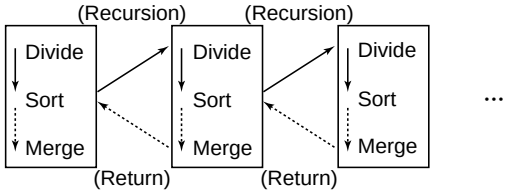
Figure 2-2-21 Algorithm of the quick sort method



(7) Merge sort method

In the algorithm of the merge sort method, all the data items are divided into sections and sorting is executed in each divided section. The sorted sections are merged into a sorted sequence. (see Figure 2-2-22). Merge means comparing data items in two sorted sequences from the head and creating one sequence of sorted data by repeatedly taking out smaller data items in a sequential manner. If the number of data items is 2*n*, repeating the merge by *n* times will complete sorting. If it is not 2*n*, some adjustment is necessary.

Figure 2-2-22 Conceptual diagram of merge sort



<Characteristics>

- The recursive call and divide-and-conquer methods are used, as in the case of the quick sort method.
- Because data sequences are sequentially accessed and sorted, the merge sort algorithm is used for external sorting, for example, storing data on magnetic tapes.

<Procedure>

1. Data is divided into two sections and each divided section is further subdivided until there is only one remaining element in a data sequence.
2. After a data sequence is divided, divided sections are sequentially merged.

Figures 2-2-23 and 2-2-24 show the status of data sequences during merge sort operations and the flowchart of the merge sort method.

Figure 2-2-23 Status of data sequences during merge sort operations

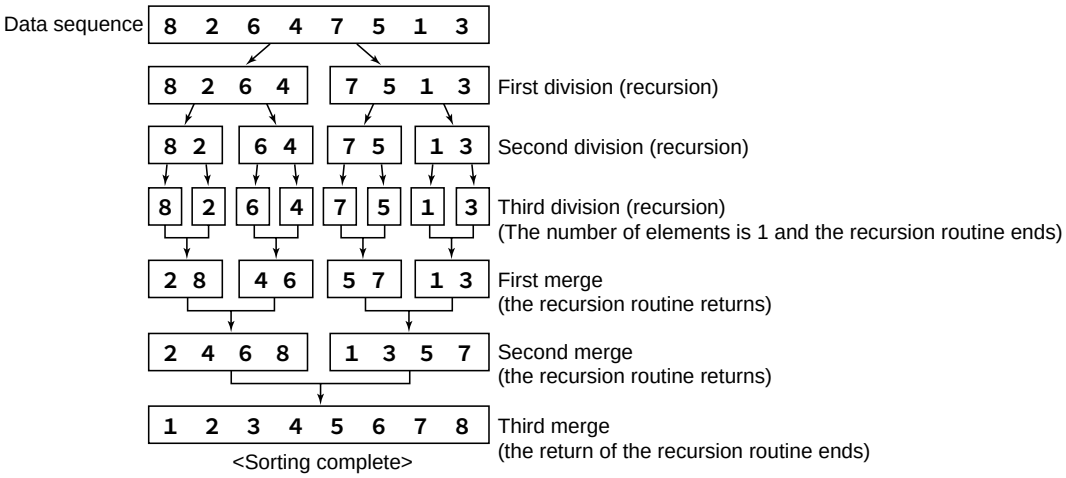
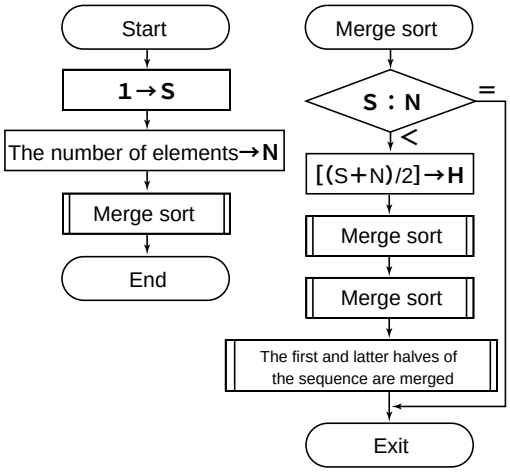


Figure 2-2-24 Flowchart of the merge sort method



Arguments (parameters)
- Data sequence: TBL
- Start position in the data sequence: S
- End position in the data sequence: N

※The first half of divided data sequence is sorted
Arguments
- Data sequence: TBL
- Start position of the data sequence: S
- End position of the data sequence: H-1

※The latter half of divided data sequence is sorted
Arguments
- Data sequence: TBL
- Start position of the data sequence: H
- End position of the data sequence: N

Argument
- Data sequence: TBL
- Start position of the first half of the data sequence: S
- Start position of the latter half of the data sequence: H
- End position of the latter half of the data sequence: N

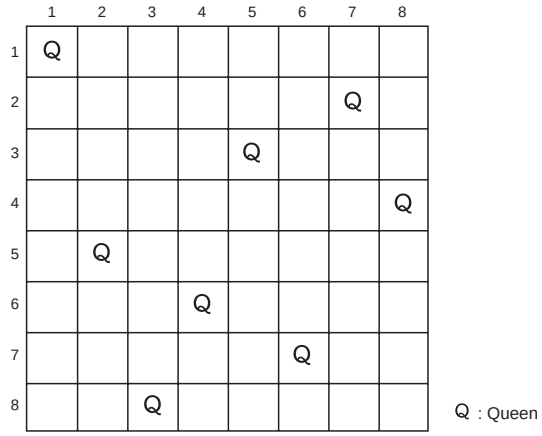
2.2.3 Recursive algorithm

A recursive call is calling a certain routine during data processing. The algorithm designed using a recursive call is the recursive call algorithm. The quick sort and merge sort algorithms are also recursive algorithms.
This section takes up the "eight-queen question" as an example for explaining how the recursive algorithm works.

(1) Eight-queen question

The eight-queen question is a question of how eight queens can be arranged on a chessboard (8 x 8 grid) to prevent pieces from being taken by the opponent. A queen is a chess piece, and it can take a piece that is located on vertical, horizontal or oblique straight lines. In the answer, therefore, queens must be positioned in such a way that they are not located on the same straight lines. Figure 2-2-25 shows one of answers.

Figure 2-2-25 Example of an answer to the eight-queen question

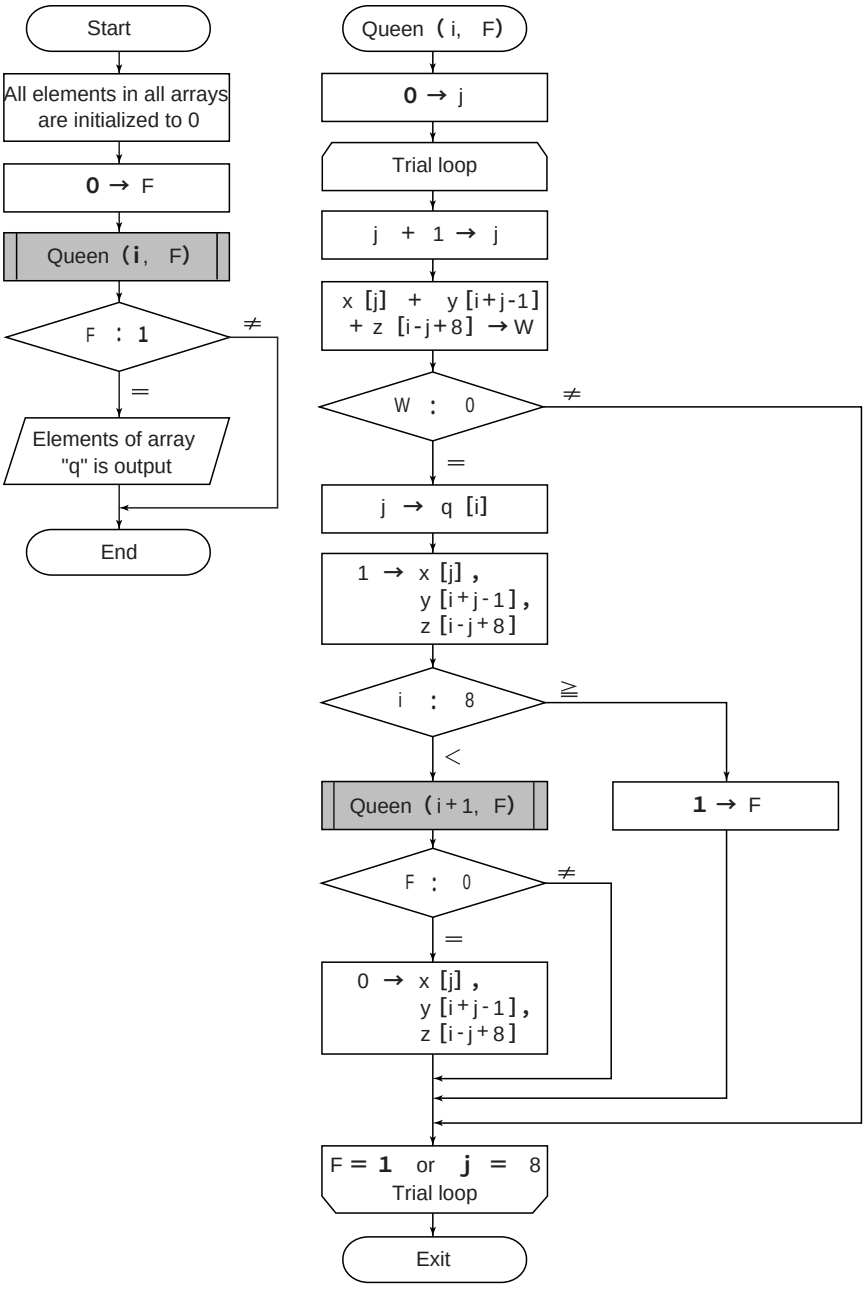


In solving this question, the following four arrangements are used:

- q [i]: The position where a queen is placed in the ith column (i=1 to 8)
  In the above example, q = {1, 5, 8, 6, 3, 7, 2, 4}
- x [j]: To indicate whether or not there is a queen on the jth row (j=1 to 8)
- y [k]: To indicate whether or not there is a queen on the kth left-to-right upward diagonal line.
  On the same left-to-right upward diagonal line, i + j is always the same. This can be used to obtain a subscript "k" with i+j-1 (k=1 to 15).
- Z [l]: To indicate whether or not there is a queen on the lth left-to-right downward diagonal line.
  On the left-to-right downward same diagonal line, i-j is always the same. This can be used to obtain a subscript "l" with i-j+8 (l=1 to 15).
- ∗ For arrays x, y and z, subscripts 0 and 1 mean 'there is no queen' and 'there is a queen' respectively.

Figure 2-2-26 shows the flowchart of the algorithm for solving this question.

Figure 2-2-26 Flowchart of the eight-queen question



2.2.4 Character string processing

Characters are searched, inserted, deleted, exchanged or compressed. This section describes character string search and compression.

(1) Character string search

To search character strings, the algorithm for searching a certain character string pattern and locating it in character strings is used.

① Simple collation

Text is compared character by character sequentially from the head. When a character string on which a search should be performed is detected, the position where it is located is set as a return value. In principle, this comparison is repeatedly executed until the first character of a character string to search matches a character in character strings in text. If there is a match, each remaining character of the matched character string is compared with each of a character string to search.

Figure 2-2-27 Image of the search

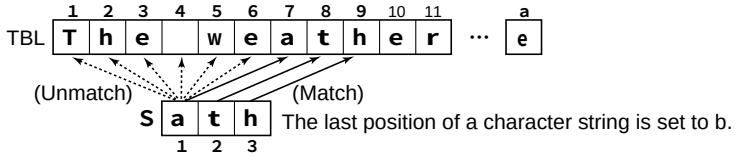
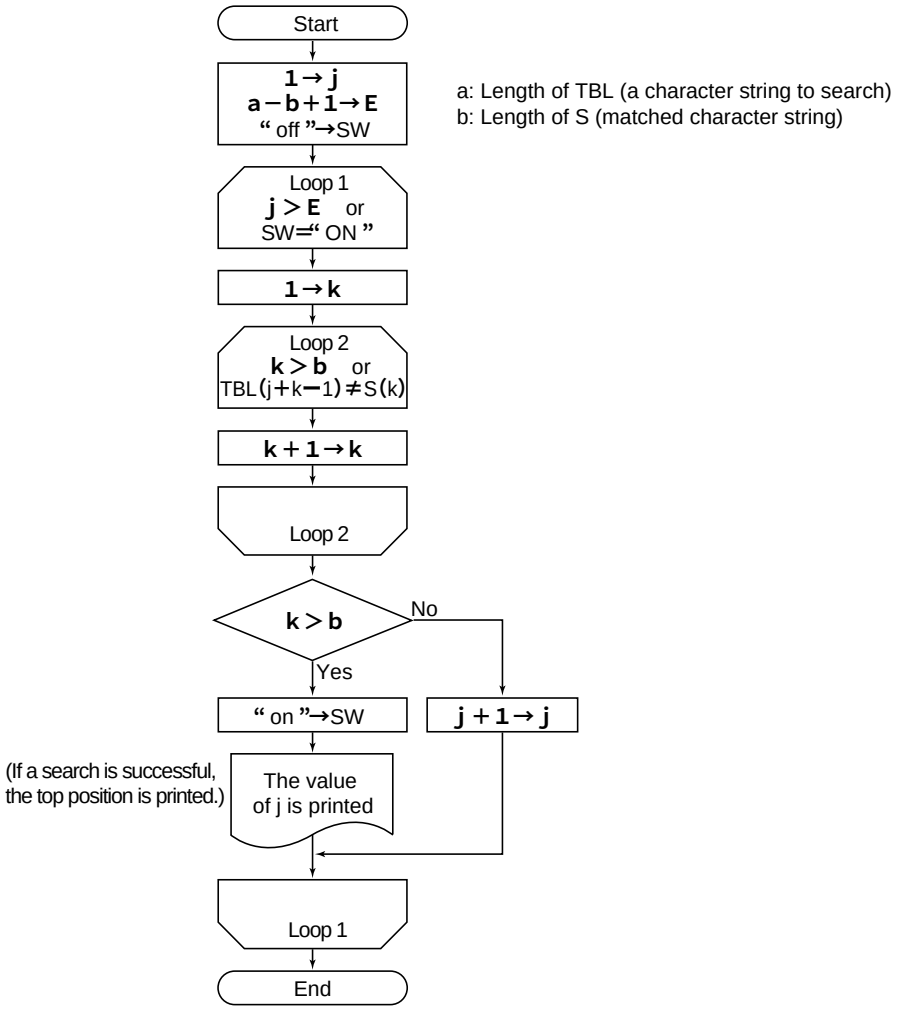


Figure 2-2-28 Flowchart of the simple collation



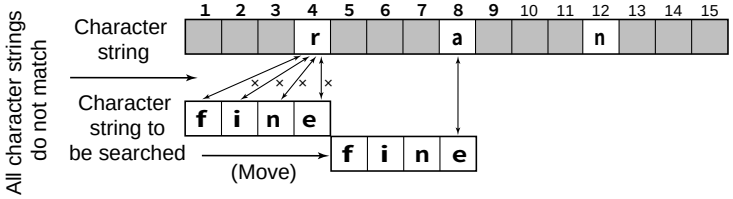
② Boyer-Moore method (BM method)

In the Boyer-Moore algorithm, data is collated while characters in the text are skipped. In this section, only the basic scheme of this algorithm is explained.

a. If there is no character string to search

Figure 2-2-29 shows a comparison of the tail of a character string to search and text patterns.

Figure 2-2-29 BM method (case 1)

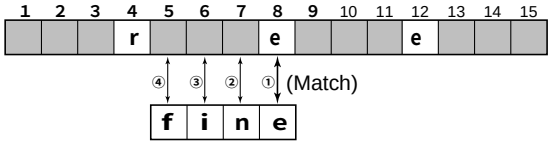


In this case, a character at the tail, and all other characters in the first text portion do not match any character of a character string to be searched. Therefore, a search point is moved by the length of a character string to be searched to allow the next search to start from that point.

b. If there is a match with a character at the tail of a character string to be searched

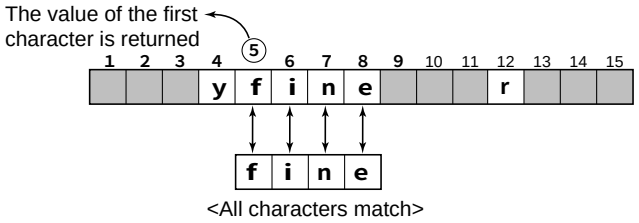
Figure 2-2-30 shows the case in which a character at the tail of a character string to be searched is compared with the text pattern, and a match is found.

Figure 2-2-30 BM method (case 2)



Because a character at the tail of a character string matches a character in the text, characters that precede the matched character must be compared. If all characters match, the value of a subscript of the first character of that matched text pattern is returned.

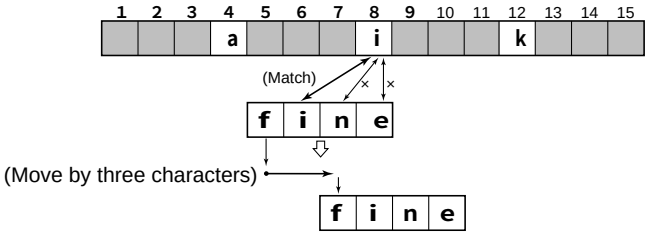
Figure 2-2-31 Search successful



c. If there is a match with one character somewhere in a character string but unmatch with a character at the tail of a character string

In the case shown in Figure 2-2-32, a character string to be searched can simply be moved. What matters is the distance of movement.

Figure 2-2-32 BM method (case 3)



The distance of movement is determined by how characters in a character string to be searched are arranged as shown in Figure 2-2-33.

Figure 2-2-33 Distance of movement

Character	f	i	n	e	others
Distance of movement	3	2	1	0	4

By storing this distance of movement in an array, a character string to be searched can be moved to a correct position.

(2) Character string compression

Making a character string short by replacing consecutive characters or blanks with the number of characters is called character string compression.
This section explains the algorithm for compressing blank characters (spaces).
In the case of the example shown in Figure 2-2-34, character strings are searched from the head, and if there are three consecutive blank characters, they are replaced with special characters (#) and the number of blank characters.

Figure 2-2-34 Image of the character string compression

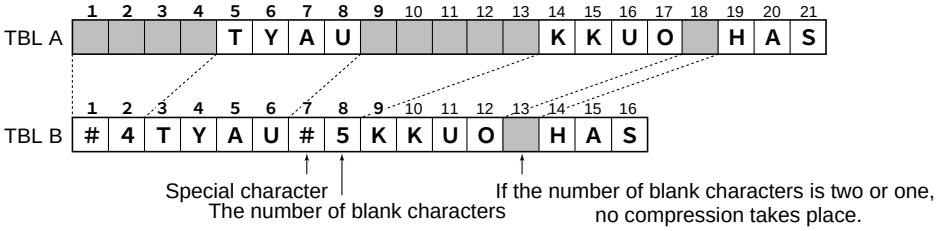
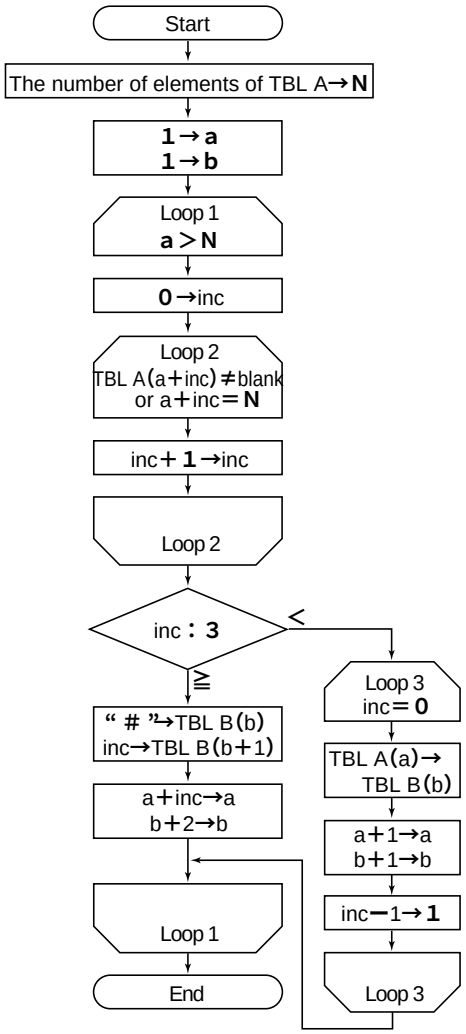


Figure 2-2-35 shows the flowchart of the character string compression.

Figure 2-2-35 Flowchart of the character string compression



2.2.5 File processing

Various files are used to perform paperwork tasks. The characteristic of file processing is processing each record one by one.
A typical file processing algorithm is as follows:

Example
Preparatory processing: Opening files, clearing counters, etc.
Main processing: Calculations, editing, outputting, etc.
End processing: Closing files, etc.

This section describes one algorithm for processing files of the same type and another algorithm for processing files of different types.

(1) Group control in processing files of the same type

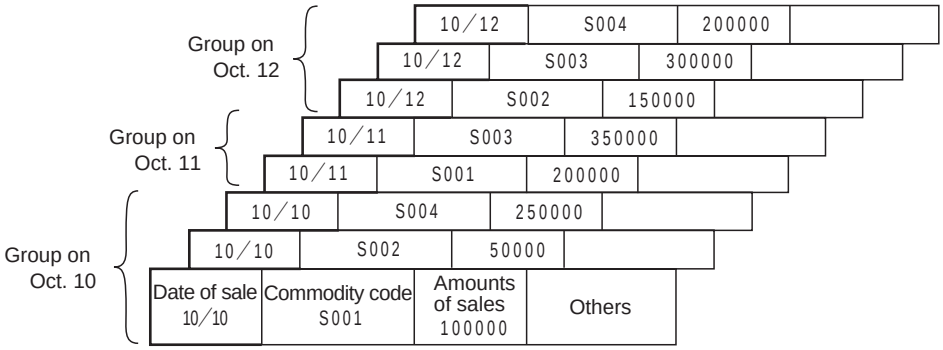
Group control (group total) is to process records with the same key as one lump. In calculating a total of sales for each customer code (key=customer code) or an average mark for each class (key=class code), for example, group control is an indispensable processing routine.
Group control requires that records are sorted according to each key.

Example Algorithm for reading sales files and printing a detailed listing of sales and total amounts of sales on each day

① Sales file layout

Figure 2-2-36 shows the layout of a sales file.

Figure 2-2-36 Layout of a sales file



② Output format (a detailed listing of sales)

Figure 2-2-37 shows the format used to print a detailed listing of sales and total amounts of sales on each day.

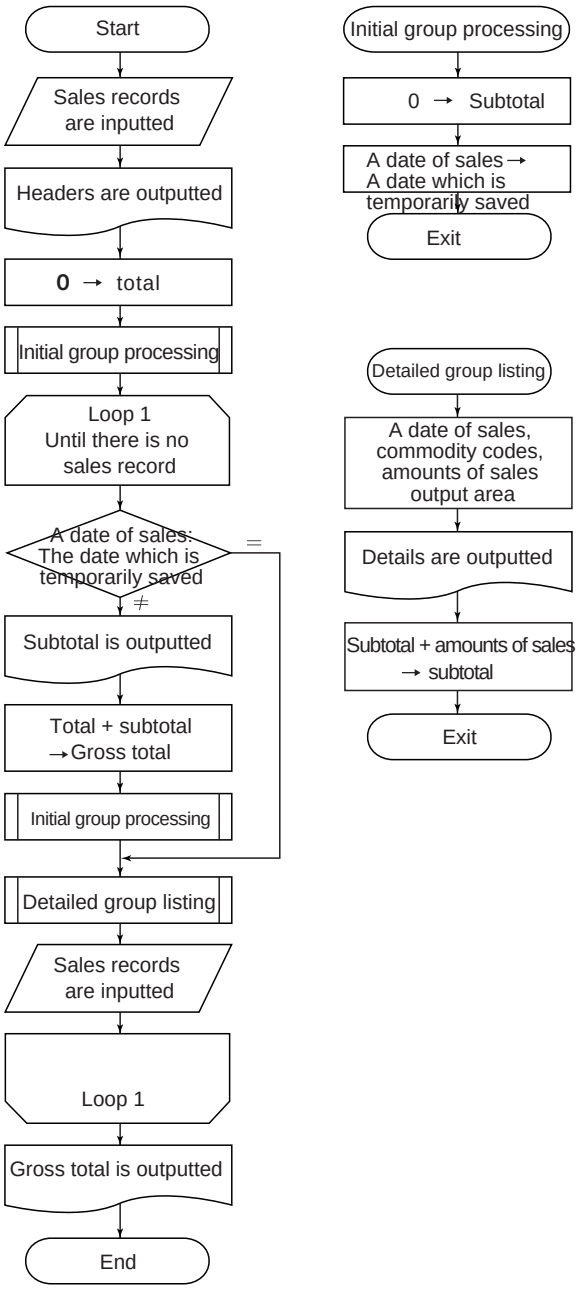
Figure 2-2-37 Format used to output a detailed list of sales

Date of sale	Commodity code	Amounts of sales	
10／10	S001	100,000	Detailed listing on Oct. 10
10／10	S002	50,000	
10／10	S004	250,000	
Total amounts of sales		400,000	····· Group total on Oct. 10
10／11	S001	200,000	Detailed listing on Oct. 11
10／11	S003	350,000	
Total amounts of sales		550,000	····· Group total on Oct . 11
10／12	S002	150,000	Detailed listing on Oct. 12
10／12	S003	300,000	
10／12	S004	200,000	
Total amounts of sales		650,000	····· Group total on Oct. 12
Gross amounts of sales		1,600,000	····· Gross total of amounts on sales files

③ Flowchart and the detailed scheme

To obtain group totals, a division between groups must be discerned. For this particular subject, a point where a date of sale changes becomes a division. Therefore, it is necessary to determine whether or not a date of sales in a newly loaded record matches that in the most recently processed records. To do this, a key (a date of sales) is temporarily saved before the first record in a group is processed, and it is compared with a key (a date of sales) of the past records.
Figure 2-2-38 shows the flowchart of the group control.

Figure 2-2-38 Flowchart of the group control



(2) Updating more than one file

If more than one file is compared using the same criteria by matching, all files must be sorted in the sequence of keys, as in the case of the group control.
File processing tasks are as follows:

- Merging files
- Matching files
- Updating files
- Maintaining files

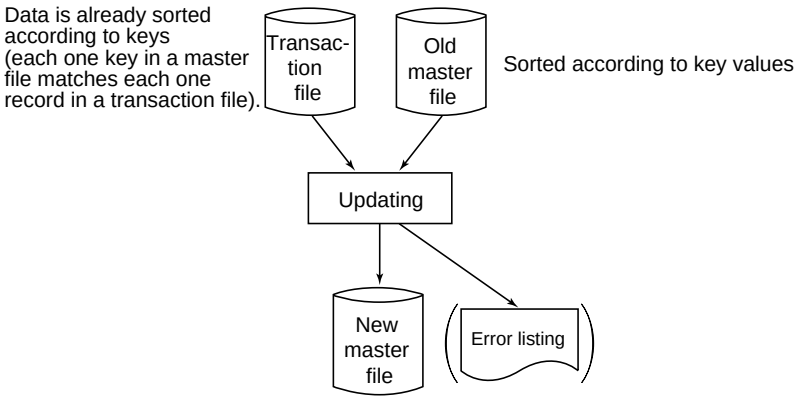
This section describes the task of file updating. File updating is to update master files based on data contained in transaction files. If a master file is a sequential file, it must be completely re-created. If a master file can be accessed randomly, a new master file does not need to be created since records can be retrieved using keys, and updated. Here, we deal with the updating procedure if a master file is a sequential file.

① 1:1 updating

1:1 updating means updating to be executed if each one record in a master file matches each one record in a transaction file. (See Figure 2-2-39.)
Multiple files are read and processing routines are determined by comparing sizes of each key as follows:
Sizes of keys

- TR key = MS key Data is updated.
- TR key > MS key Data is copied (data is written into a new master file).
- TR key < MS key Data is added to a new master file (it may be the case where this status is considered an error and the error routine takes place).

Figure 2-2-39 Image of the 1:1 updating



<Transaction file>

Commodity code	Commodity name	Amounts of sales
S001	Television	134,000
S004	Video recorder	98,000
S006	Personal computer	156,000
S007	Printer	43,000
S009	Scanner	86,000

<Old master file>

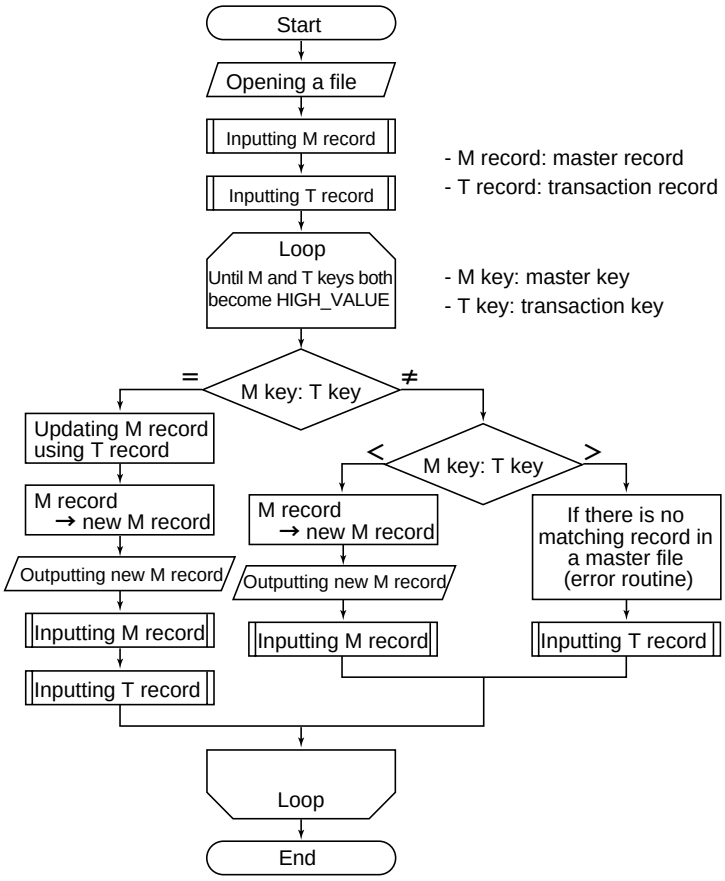
Commodity code	Commodity name	Amounts of sales
S001	Television	656,000
S002	Stereo	423,000
S004	Video recorder	256,000
S006	Personal computer	432,000
S007	Printer	83,000
S008	Digital camera	92,000

Commodity code	Commodity name	Amounts of sales
S001	Television	790,000
S002	Stereo	423,000
S004	Video recorder	354,000
S006	Personal computer	588,000
S007	Printer	126,000
S008	Digital camera	92,000
S009	Scanner	86,000

<New master file>

Figure 2-2-40 shows the flowchart of the 1:1 updating.

Figure 2-2-40 Flowchart of the 1:1 updating



② 1:n updating

1:n updating is the updating routine used if one record in a master file matches more than one record in a transaction file. (See Figure 2-2-41.)
In principle, multiple files are read, as in the case of the 1:1 updating, and processing routines are determined by comparing sizes of keys as follows:
Sizes of keys

- TR key = MS key Data is totaled and the results are stored in a work area.
- TR key > MS key Data is updated.
- TR key < MS key Data is added to a new master file (it may be the case where this is considered an error and the error routine takes place).

Figure 2-2-41 Image of the 1:n updating

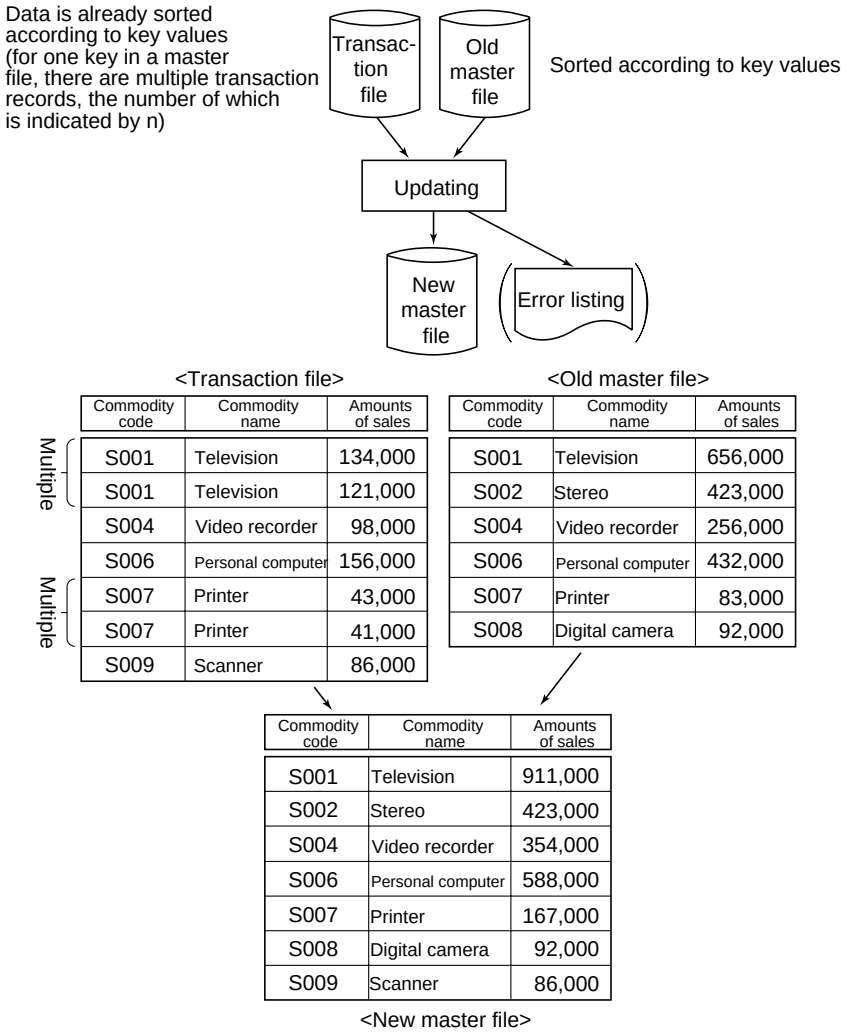
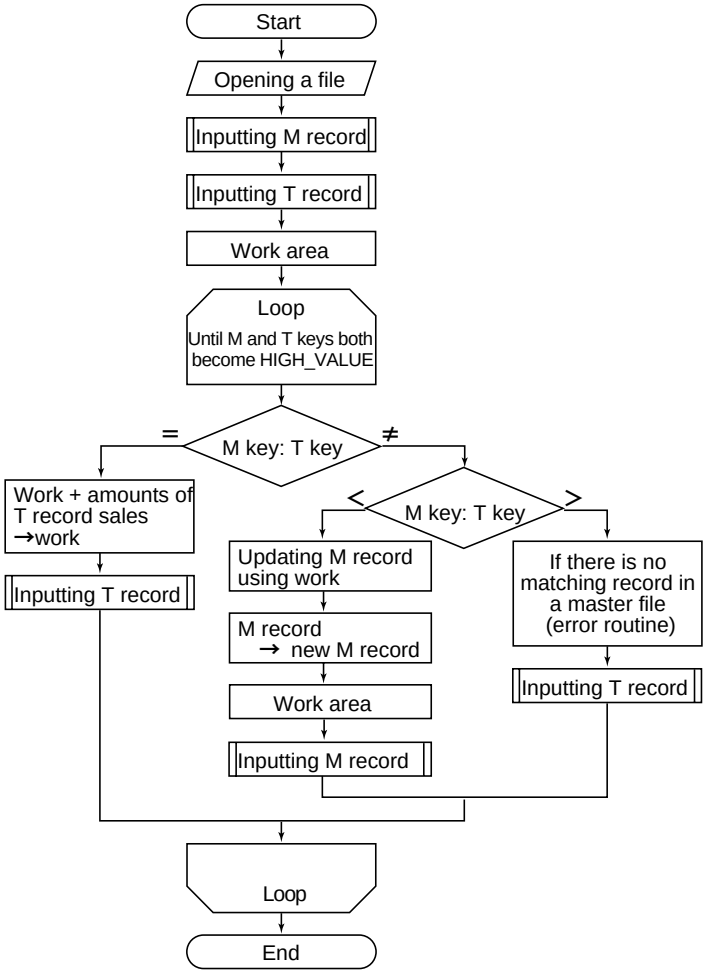


Figure 2-2-42 shows the flowchart of the 1:n updating.

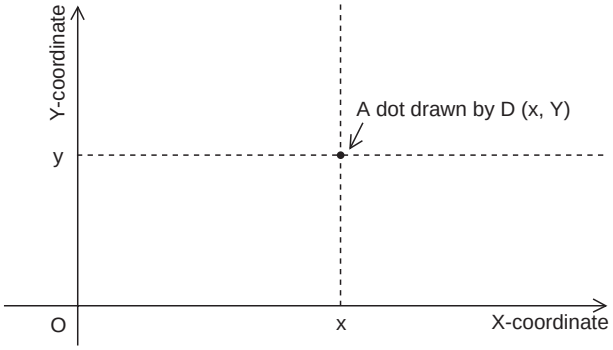
Figure 2-2-42 Flowchart of the 1:n updating



2.2.6 Drawing figures

In the present world of computers, CAD, CG and other figure drawing technologies are used. In an algorithm for drawing figures, a figure is treated as a set of dots. How a simple straight line and a circle can be drawn is explained in this section.
A function D (x, y) is used to draw a dot at an intersection point where a line along the x-coordinate meets a line along the y-coordinate, as shown in Figure 2-2-43.

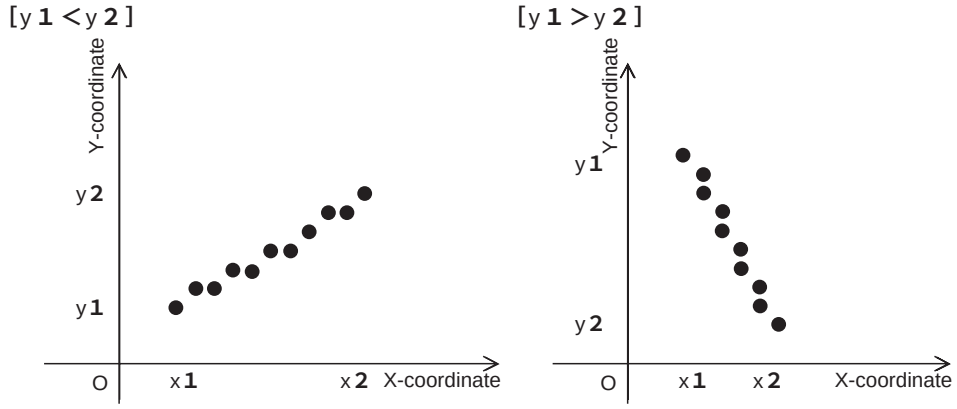
Figure 2-2-43 How the function D (x, y) works



(1) Drawing a straight line

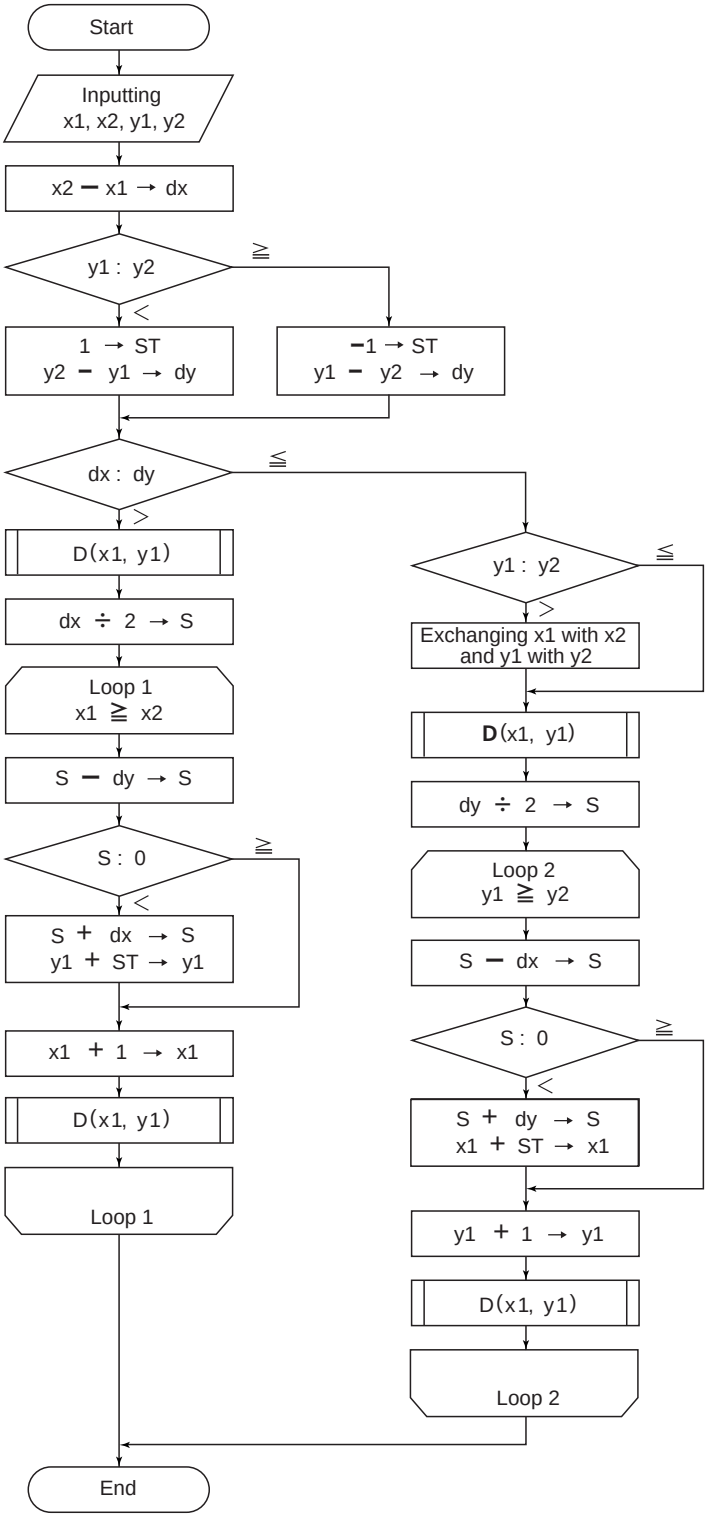
Figure 2-2-44 shows the routine of straight line drawing.

Figure 2-2-44 Examples of straight line drawing



The flowchart shown in Figure 2-2-45 is an algorithm for drawing a straight line that connects two given points in the coordinate, i.e., (x1, y1), (x2, y2) | 0<x1≦x2, 0<y1, 0<y2 | . Although this algorithm can draw straight lines running obliquely in upper or lower right directions by drawing dots on an integer coordinate, it is designed to allow lines to look like straight lines. Also, multiply-divide operations are kept to a minimum and add-subtract operations are mostly used to increase the processing speed.

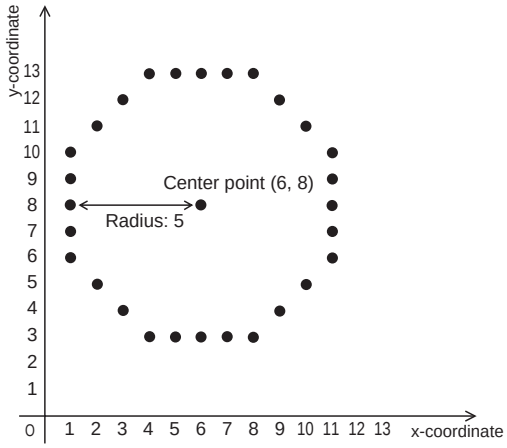
Figure 2-2-45 Flowchart of straight line drawing



(2) Drawing a circle

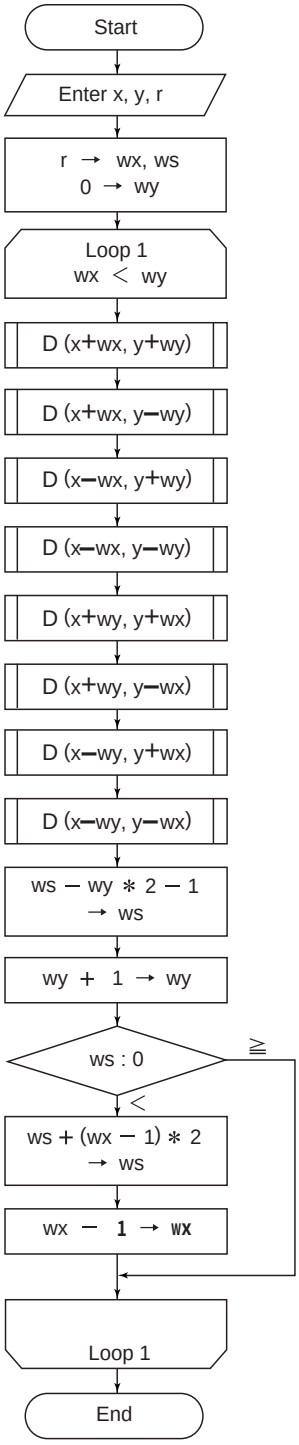
Figure 2-2-46 shows how a circle is drawn.

Figure 2-2-46 How a circle is drawn



The flowchart in Figure 2-2-47 shows the algorithm for drawing a circle based on the coordinate (x, y) of a given center point and the radius r. To draw this circle, a trigonometric function that needs a long execution time is not used.

Figure 2-2-47 Flowchart for drawing a circle



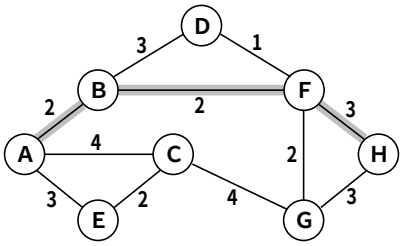
2.2.7 Graph

The graph that we discuss in this section is one made up of arcs formed by connecting more than one point. Basically an undirected graph is assumed. A directed graph can also be used, depending on the type of problem to be solved.
The shortest path problem is described here as one of the representative graph problems.

(1) Shortest path problem

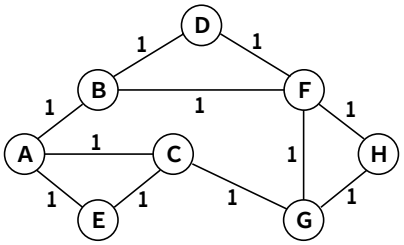
As shown in Figure 2-2-48, there are routes and nodes, and the shortest route that runs from point A to point H must be found. Because the numbers along each route show a distance, you can see at once that the thick line is the shortest route.

Figure 2-2-48 Map (the thick line is the shortest route)



The graph shown in Figure 2-2-48 that has numbers along each route to show a route distance is called a weighted graph. Although we can discern the shortest route by sight, an algorithm must be used to enable a computer to find the shortest route through calculations.
We will here describe three route search methods: the depth-first and the breadth-first search method which assume that the distance of each route is 1, and the Dijkstra's search method which is a major method for finding the shortest route.

Figure 2-2-49 Graph in which the distance of all routes is set to 1



① Depth-first search method

Using the depth-first search method, one route is chosen as a start point and a search starts to find a route to a destination using the stack.

<Procedure> (Figure 2-2-50)

1. A node at the start point is put into the stack.
2. A node is picked from the stack. Nodes adjacent to a picked node are checked and those that have never before been pushed into the stack are chosen. The chosen nodes are pushed into the stack.
3. When the chosen nodes are pushed into the stack, the node picked in step 2 above is stored in the array.
4. Steps 1, 2 and 3 are repeatedly executed until a target node is put into the stack or the stack becomes empty.

Figure 2-2-50 The state of the stack

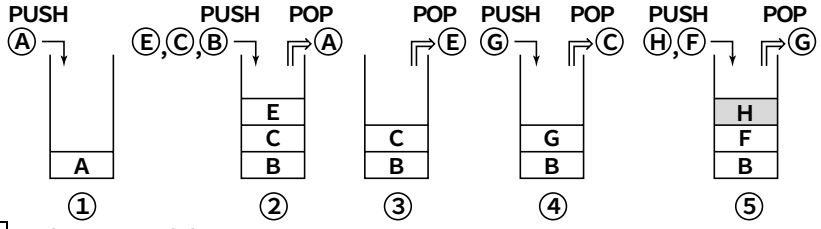
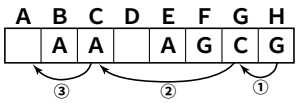


Figure 2-2-51 The state of the array



Before the chosen nodes are pushed into the stack, the node picked in step 2 above must be stored in the array. For example, if Ⓐ is picked and Ⓑ, Ⓒ and Ⓔ are pushed into the stack, A must be stored in the elements of B, C and E. After a target node has been put into the stack, nodes are traced one by one sequentially to find the shortest route. (The same routine is executed in the case shown in Figure 2-2-53.)

<Results>

A search executed with the depth-first search method results in the route A-C-G-H shown in Figure 2-2-51. Although the above Figures show the basic scheme of the search execution routine, what happens during an actual search is more complex. That is, assuming that the shortest route cannot be found, different routes are repeatedly searched, that is, as the search routine reaches the step before the step of putting H, shown in Figure 2-2-50, step ⑤ into the stack, it leaps back to step 1 and repeats all the steps. This routine is repeated until the stack becomes empty so that all routes are counted to find the shortest route.

② Breadth-first search method

Using the breadth-first search method, all possible routes that can lead to a destination are searched using the queue.

<Procedure> (Figure 2-2-52)

1. A node at the start point is put into the queue.
2. One node is picked from the queue. Nodes adjacent to the picked node are checked and those that have never before been put into the queue are chosen. The chosen nodes are put into the queue.
3. The node picked in step 2 is stored in the array.
4. Above steps 1, 2 and 3 are repeatedly executed until a target node is reached or the queue becomes empty.

Figure 2-2-52 The state of the queue

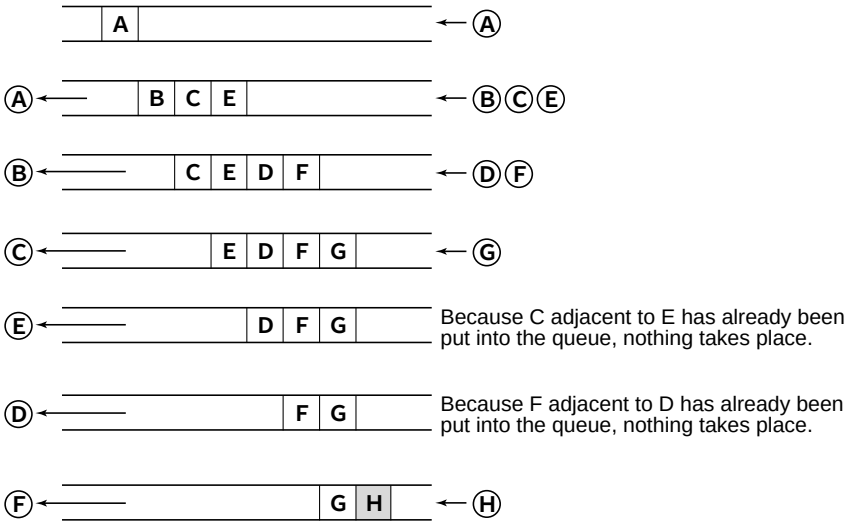
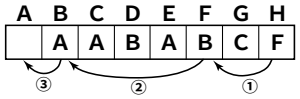


Figure 2-2-53 The state of the array



Before a node is put into the queue, the node picked in step 2 is stored in the array. For example, if A is picked and B, C and E are put into the queue, Ⓐ is stored in the elements of Ⓑ, Ⓒ and Ⓔ.

A search executed using the breadth-first search method results in the shortest route A-B-F-H shown in Figure 2-2-53. All nodes are searched one by one, as shown above, while calculations are being made to find the best route.

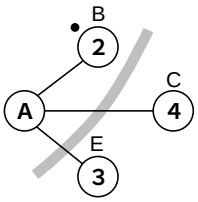
③ Dijkstra's search method

The Dijkstra's search method is a method to which the breadth-first search method is applied.

<Procedures> (Figure 2-2-54)

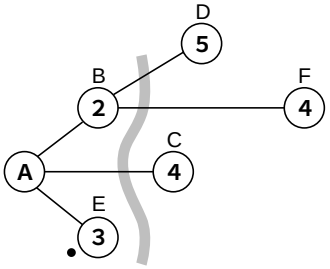
1. The distance to each node adjacent to the start node is measured and the node located at the shortest distance is chosen (this chosen node is defined as X).
2. The distance to each node adjacent to X from the start node as well as the distance to nodes except X from the start node are measured and the node located at the shortest distance is chosen.
3. Step 2 is repeatedly performed on all nodes until a target node is reached.
4. By adding the distance attached to each node, the shortest route can be established.

Figure 2-2-54 Dijkstra search method

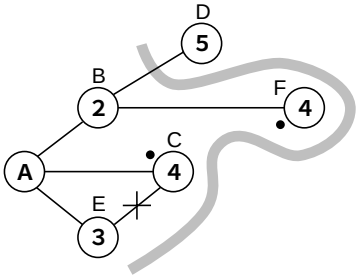


The shaded line is the line that separates nodes or routes that have already been processed.

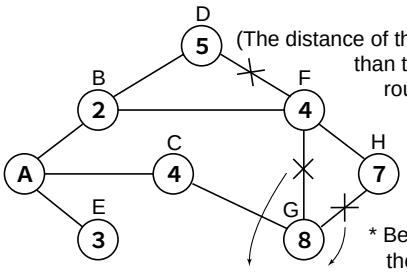
The distance from the start node to each adjacent node is compared and the node located at the shortest distance is marked.



The distance to a node located on the A-to-B route is compared with the distance of the node adjacent to A and the node at the shortest distance is marked.



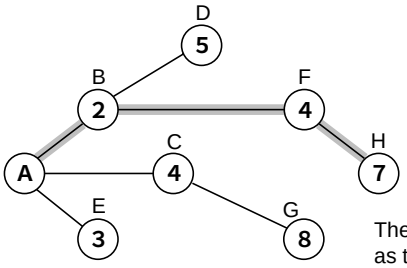
C adjacent to E located at the shortest distance can be reached via A. Because the distance from A to C is shorter than the distance from A to C via E, the E-to-C route is cut off.



(The distance of the B-to-F route is shorter than the distance of the B-to-F route via D. Therefore, the D-to-F route is cut off.)

* Before a route is cut off, the distance is considered.

(Both routes are cut off.)



<The search is completed.>

The result becomes the same as that obtained using the breadth-first search method.

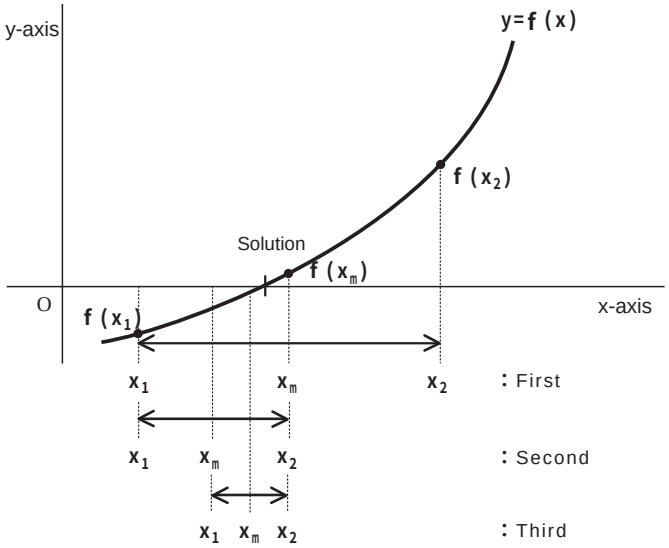
2.2.8 Numeric calculation

(1) Solution of algebraic equations

① Bisection method

The bisection method is the simplest solution of higher-degree equations. A commonly used higher-degree equation is defined as y = f(x) and various values are assigned to x. The line is plotted by assigning values into x. The value of x when the line intersects y = 0 (x-axis) becomes the root of this equation.
Figure 2-2-55 is a graph plotted based on y = f(x).

Figure 2-2-55 Graph plotted based on y = f(x)



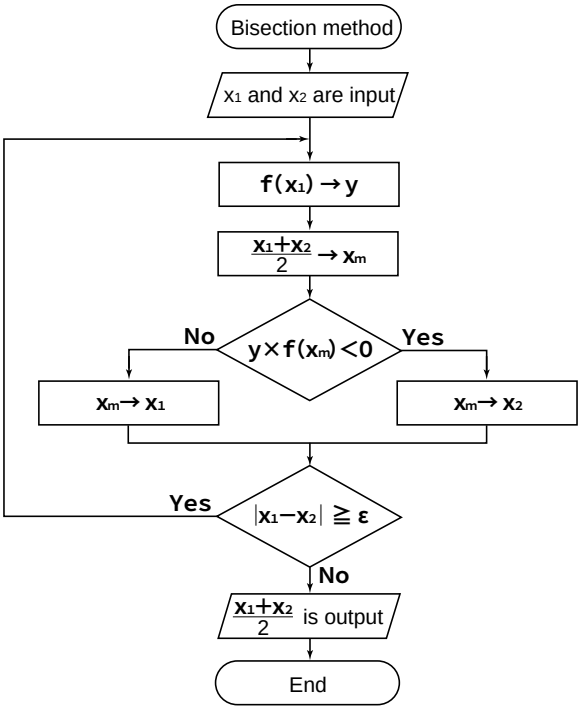
$f(x_1) \cdot f(x_2)<0$ shows that there is at least one solution within the $[x_1, x_2]$ region in which the line intersects the x-axis. Using the bisection method, this $[x_1, x_2]$ region is divided into two sections. To obtain an approximate solution, the routine of dividing a region into two sections is repeatedly executed to choose a section (indicated by ↔) where there is a solution.

Figure 2-2-56 Algorithm of the bisection method

Step 1	The point xm that divides a region into two sections is calculated.
Step 2	Each of the divided sections [x_1, x_m] and [x_m, x_2] is checked to determine in which section there is a solution. If $f(x_1) \cdot f(x_m)<0$, there is a solution in the left section. If not, there is a solution in the right section.
Step 3	If $f(x_1) \cdot f(x_m)<0$, $x_m \rightarrow x_2$. If not, $x_m \rightarrow x_1$.
Step 4	A judgment made -- whether the section length $\lvert x_1-x_2 \rvert$ is smaller than the convergence judgment value **ε** or not -- is used to determine whether the approximate solution has been obtained or not. If $\lvert x_1-x_2 \rvert$ is larger than **ε**, the routine goes back to step 1 and repeats steps 1 through 4.

Figure 2-2-57 shows the flowchart of the algorithm for obtaining the solution of f(x) = 0 using the bisection method.

Figure 2-2-57 Flowchart of the algorithm for obtaining the solution of f(x) = 0 using the bisection method



② Newton's method

Newton's method assumes that an approximate solution of a higher-degree equation is already known. The approximate solution is repeatedly corrected to obtain a true solution. Newton's method is superior to other methods in that the convergence speed is faster and both real and imaginary solutions can be obtained.
Figure 2-2-58 is a graph plotted using Newton's method.

Figure 2-2-58 Newton's method

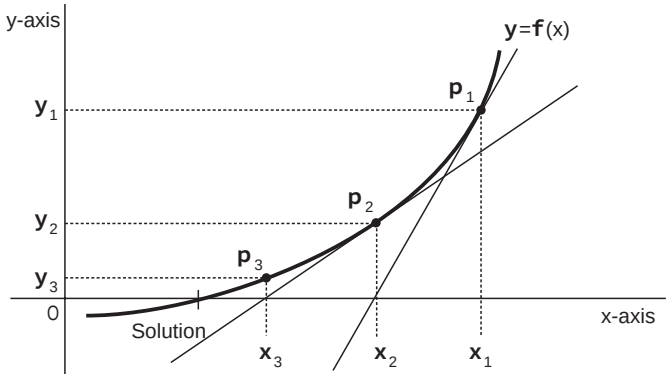


Figure 2-2-59 Algorithm of Newton's method

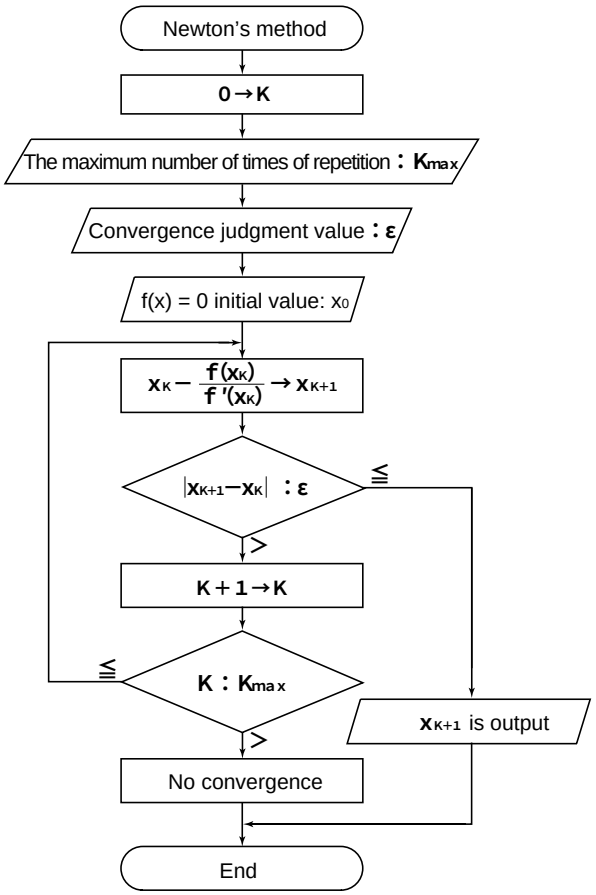
Step 1	A tangential line y = f(x) at point p_1 of the coordinate x_1, y_1 is drawn and point x_2 where the tangential line intersects x-axis is obtained.
Step 2	A tangential line y = f(x) at point p_2 of the coordinate x_2, y_2 is likewise drawn and point x_1 where the tangential line intersects x-axis is obtained. As this step is repeatedly executed the tangential line moves closer to a solution.
Step 3	The difference between adjacent approximate values obtained in step 2 is compared with a predetermined convergence judgment value precision. Steps 1 and 2 are repeatedly executed until this difference becomes smaller than the predetermined convergence judgment value.

Because the equation for a tangential line at point p1 is $y-f(x_1) = f'(x_1)(x-x_1)$, point x_2 where a tangential line intersects x-axis can be obtained using the following equation:

$$x_{i+1} = x_1 - \frac{f(x_i)}{f'(x_i)} \quad (i = 0, 1, 2, ...)$$

Figure 2-2-60 shows the flowchart of the algorithm for obtaining the solution of f(x) = 0 using Newton's method.

Figure 2-2-60 Flowchart of the algorithm for obtaining the solution of f(x) = 0 using Newton's method



(2) Numerical integration

Numerical integration methods are used to calculate the area of a figure enclosed by curved lines. This section describes the trapezoidal rule and Simpson's method of all commonly used numerical integration methods.

① Trapezoidal rule

Figure 2-2-61 shows the basic concept of the trapezoidal rule.

Figure 2-2-61 Basic concept of the trapezoidal rule

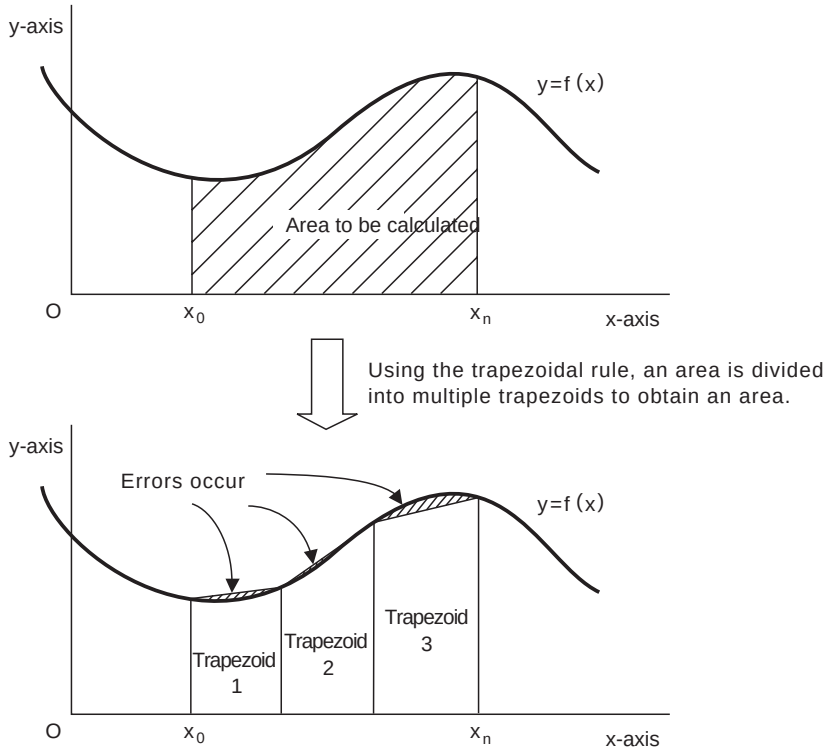


Figure 2-2-62 shows the procedure for calculating an area enclosed by the curved line y = f(x), a region along the x-axis and a portion enclosed by the x-axis.

Figure 2-2-62 Procedure for calculating an area using the trapezoidal rule

Step 1 Vertical lines are drawn in an area at regular intervals to divide it into parts.
Step 2 Assuming that the curved line of each divided area is a straight line, each divided area is regarded as a trapezoid.
Step 3 Each divided area is regarded as a trapezoid and each area is calculated.
An area of one trapezoid = (upper base + lower base) x height÷2
Step 4 Areas of all trapezoids are totaled to obtain an area.

If an area is calculated using the trapezoidal rule, errors occur with respect to shaded portions, as shown in Figure 2-2-61, since a true area is different from a trapezoidal area. To reduce errors, the number of trapezoids is increased. Figure 2-2-63 shows how an area is divided into trapezoids.

Figure 2-2-63 An area divided according to the trapezoidal rule

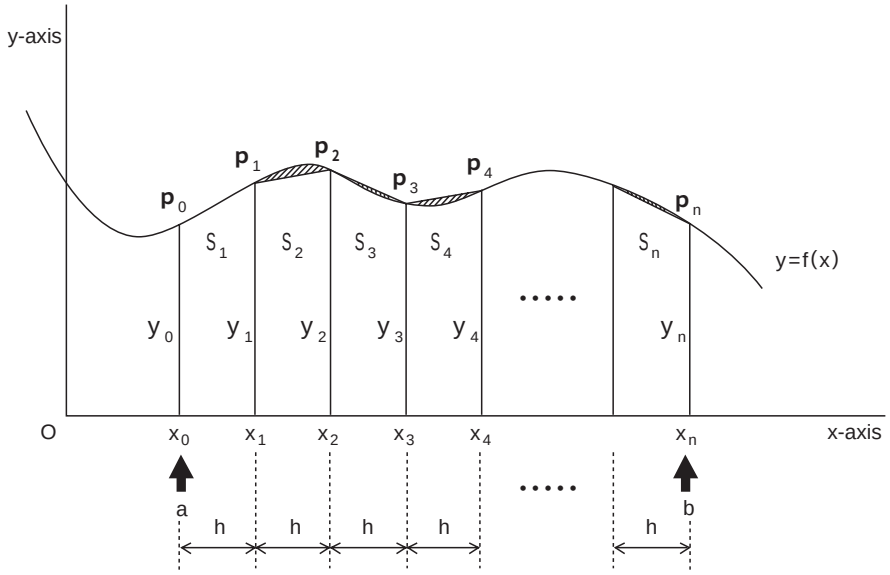


Figure 2-2-64 shows the algorithm for calculating an area using the trapezoidal rule based on Figure 2-2-63.

Figure 2-2-64 Algorithm of the trapezoidal rule

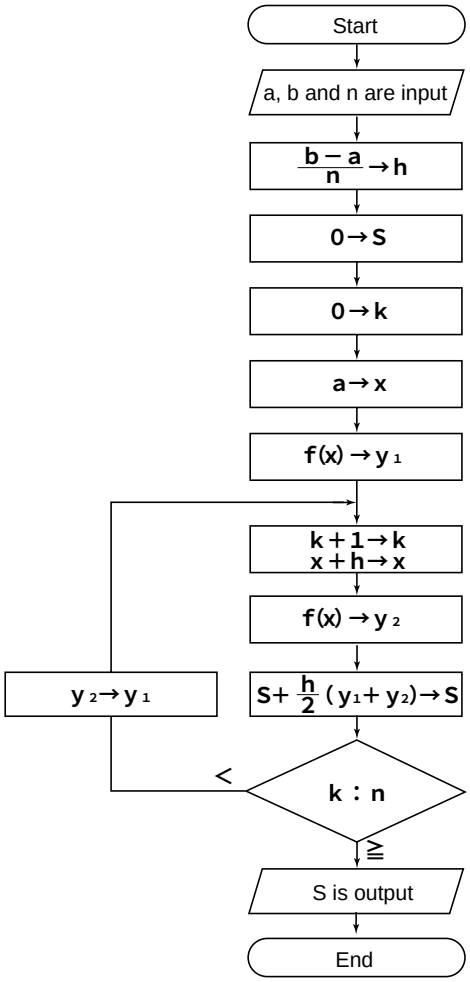
Step 1 The portion where integration takes place, i.e., [a, b] of y = f(x), is divided into parts, the number of which is defined as n.
Step 2 Points where lines set at regular intervals intersect x-axis are defined as x_0, x_1, x_2, ... x_n from left to right.
Step 3 The values of functions at above intersection points are defined as y_0, y_1, y_2, ... y_n.
Step 4 Points where lines set at regular intervals intersect curved lines are defined as p_0, p_1, p_2, ... p_n. With x, y and p points connected, a trapezoid is formed.
Step 5 Providing that an area of each trapezoid is S_1, S_2, S_3, ... S_n,

$$S_1 = \frac{h}{2}(y_0 + y_1)$$
$$S_2 = \frac{h}{2}(y_1 + y_2)$$
$$\vdots$$
$$S_n = \frac{h}{2}(y_{n-1} + y_n)$$

Each area is calculated using the above equations.
Step 6 An area enclosed by the curved line can be obtained by totaling areas of each divided area: $S = S_1+S_2+S_3+ ...+S_n$

Figure 2-2-65 shows the flowchart of the algorithm for calculating an area using a computer and the trapezoidal rule.

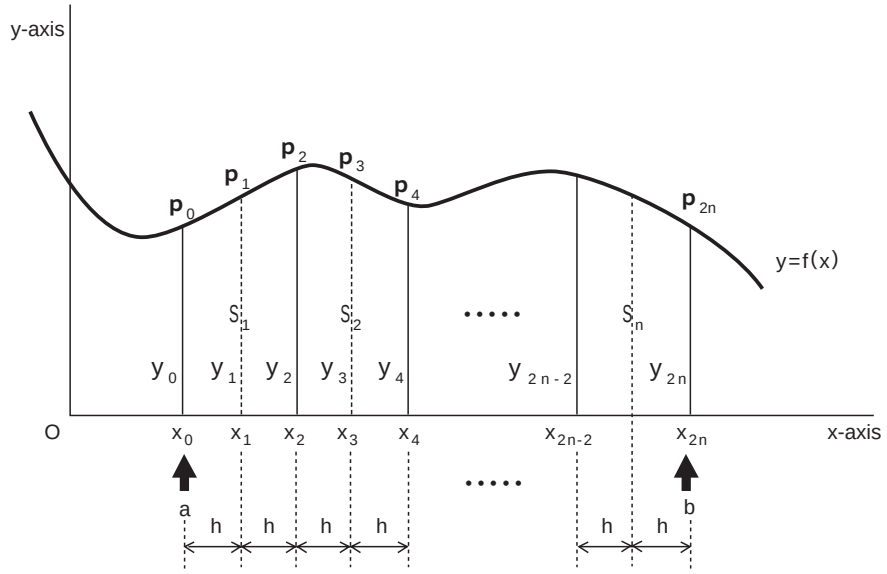
Figure 2-2-65 Flowchart of the approximate area calculation using the trapezoidal rule



② Simpson's method

Using the trapezoidal rule, a curved line is divided at regular intervals and intersection points on the curved line are connected to form trapezoids. Areas of each trapezoid are totaled to obtain a whole area. Although errors can be reduced by increasing the number of trapezoids, it still holds that the greater the number of trapezoids, the longer it takes to execute totaling. Furthermore, because the method is based on the scheme wherein an area enclosed by three straight lines and one curved line is regarded as a trapezoid, there is concern over the accuracy of the result obtained.
As a solution, Simpson's method is used as shown in Figure 2-2-66. Using this method, a curved line is approximated to an easy-to-handle parabola to calculate an area.

Figure 2-2-66 Area divided using Simpson's method



To calculate an area S1 enclosed by y = f(x), x = x_0, x = x_2 and x-axis shown in Figure 2-2-66 using Simpson's method, f(x) is considered a parabola that runs through p_0, p_1 and p_2, and S_1 is approximated as follows:

$$S_1 = \frac{(y_0 + 4y_1 + y_2)h}{3}$$

This method is quite different from the trapezoidal rule in that an area is equally divided into 2n (equal, even-number division), not into n.

Figure 2-2-67 shows the algorithm for calculating an area shown in Figure 2-2-66 using Simpson's method.

Figure 2-2-67 Algorithm for calculating an area using Simpson's method

Step 1 The section [a, b] of the function y = f(x) where integration takes place is equally divided into 2n (equal, even-number division). H

Step 2 Points where dividing lines intersect x-axis are defined from the left as x_0, x_1, x_2, ... x_{2n}.

Step 3 Function values at each intersection point are defined as y_0, y_1, y_2, ... y_{2n}.

Step 4 Points where dividing lines intersect a curved line are defined as p_0, p_1, p_2, ... p_{2n}.

Step 5 Three points $p_{2i-2}(x_{2i-s}, y_{2I-2})$, $p_{2i-1}(x_{2i-1}, y_{2i-1})$ and p_{2i} (x_{2i}, y_{2i}) in two sections are approximated to a parabola to calculate an area Si.

$$S_1 = \frac{(y_0 + 4y_1 + y_2)\,h}{3}$$

$$S_2 = \frac{(y_2 + 4y_3 + y_4)\,h}{3}$$

$$\vdots$$

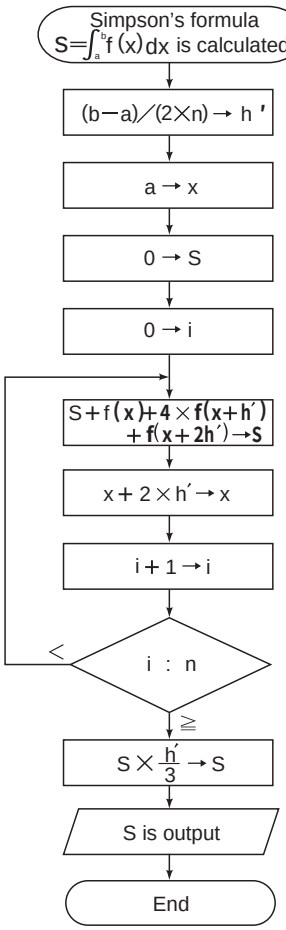
$$S_n = \frac{(y_{2n-2} + 4y_{2n-1} + y_{2n})\,h}{3}$$

Step 6 Areas Si in each section are totaled to obtain an area S enclosed by a curved line.

$$S = S_1 + S_2 + \cdots + S_n$$

Figure 2-2-68 shows the flowchart of the algorithm for calculating an area using a computer and Simpson's method.

Figure 2-2-68 Flowchart of the area approximation calculation using Simpson's method



2.2.9 Collation algorithm

In the collation algorithm, values stored in the array are compared to obtain a solution. A search of character strings described under Section 2.2.4, Character string processing, also uses one of the collation algorithms.
This section describes the stable marriage problem to explain one of the representative collation algorithms.

(1) Stable marriage problem

In solving the stable marriage problem, stable pairs of men and women are determined.
Stable means a state in which men and women feel more fondness for their present partners than others. Specifically, supposing that there are three men and three women, men are named A, B and C and women are named a, b and c. The levels of fondness (high to low levels in the order 1, 2 and 3) are shown in the table below:

<Men>

	1	2	3
A	b	a	c
B	c	a	b
C	c	b	a

<Women>

	1	2	3
a	A	C	B
b	C	B	A
c	A	B	C

If they are paired as
P = { (A, a), (B, b), (C, c) },
A feels fonder of b than a, who is the present partner. b feels fonder of B, who is the present partner, than A.

Therefore, there should be no problem. B feels fonder of a and c than b, who is the present partner. Because a feels fonder of A, who is the present partner, than B, there should be no problem. However, c feels fonder of B than C who is the present partner. This state is called an unstable state. By changing partners, they can be paired as

P = { (A, a), (B, c), (C, b) }.

This state of pairing can be analyzed as follows:

①A feels fonder of b than a, who is the present partner. → b feels fonder of C, who is the present partner, than A.
② B feels fonder of c who is the present partner.
③ C feels fonder of c than b, who is the present partner. → c feels fonder of B, who is the present partner, than C.
④ a feels fondest of A who is the present partner.
⑤ b feels fondest of C, who is the present partner.
⑥c feels fonder of A than B, who is the present partner. → A feels fonder of a, who is the present partner, than c.

In this pairing, no pairs have an unstable factor. This result is called stable or stable matching.

Stable matching may not necessarily be determined uniquely. There may be a case where stable matching cannot be achieved using the approach described above. The stable marriage problem to be described in the following, is designed to achieve stable matching by specifying the conditions for determining pairs:

[Stable marriage problem]

N men and N women are looking for stable pairs.

- Men's and women's levels of fondness are preset in array M and F (the number of elements: N+1).
- As levels of fondness, element numbers 1 through 5 (high to low) are assigned to the numbers of each partner.

Example If there are five men and five women

[Combination M: Levels of fondness seen from the side of the men]

	1	2	3	4	5	6 (PT)
M (1)	2	1	4	3	5	0
M (2)	1	3	4	2	5	0
M (3)	1	4	5	2	3	0
M (4)	5	4	1	3	2	0
M (5)	4	2	3	1	5	0

* A partner is entered in PT.
0 is set here as the initial value.

[Combination F: Levels of fondness seen from the side of the women]

	1	2	3	4	5	6 (PT)
F (1)	3	1	5	2	4	0
F (2)	2	1	4	3	5	0
F (3)	3	2	5	4	1	0
F (4)	5	2	1	3	4	0
F (5)	5	4	2	1	3	0

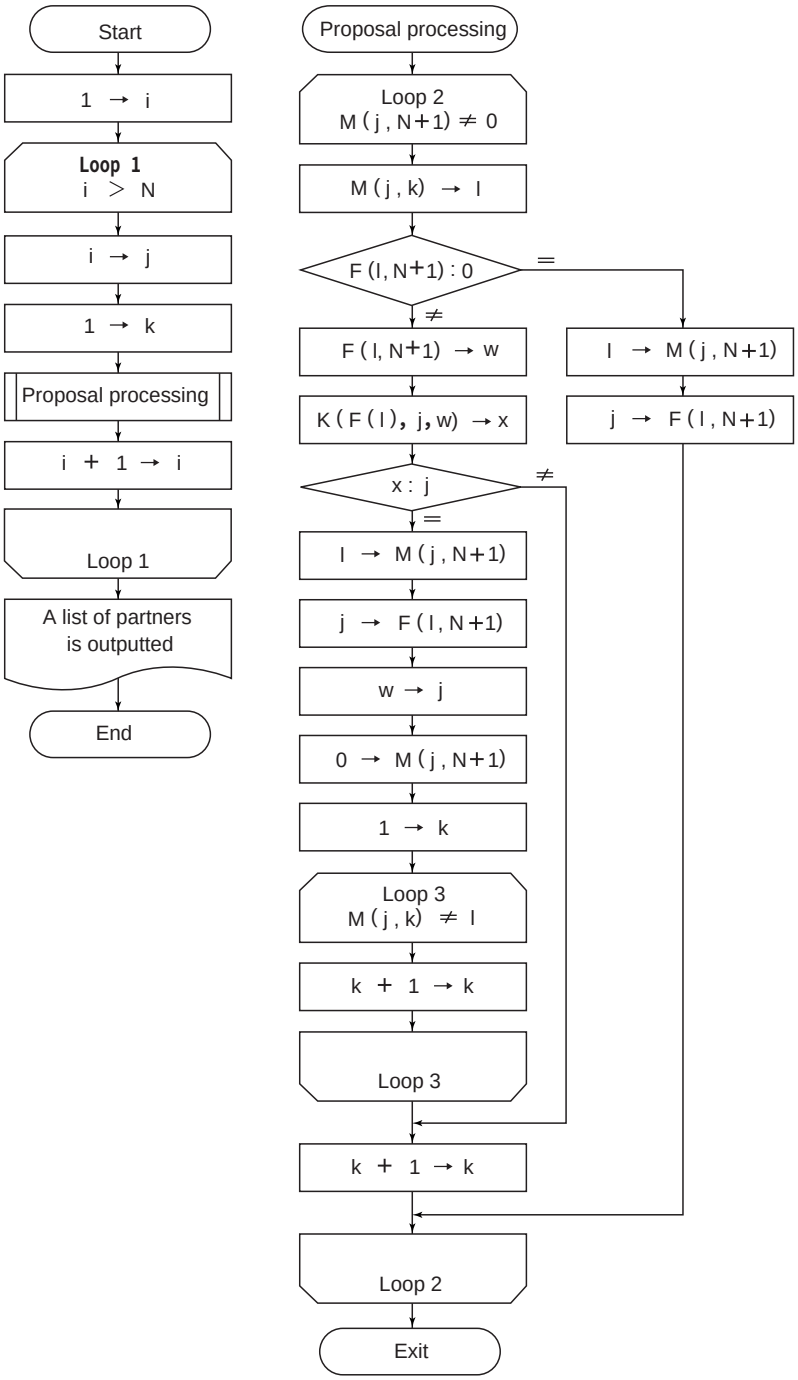
* A partner is entered in PT.
0 is set here as the initial value.

- Proposals are made in the order of M (1), M (2).... M (N).
- Men make proposals to women in high-to-low order of fondness. They repeatedly make proposals until they obtain OKs. It may happen, however, that the OKs thus obtained are canceled.
- Women give OK or NO on the following criteria:
  ① If one receives a proposal for the first time, she gives an OK and secures him as a partner.
  ② If she already has a man, she compares her fondness for the present partner and that for the one who has made the proposal.
- If she feels fonder of the present partner, she gives a NO.
- If she feels fonder of a man who has made a proposal, she cancels the present partner and secures the man who has made the proposal as a partner (OK).
- A man who obtained an OK but was canceled later begins making proposals to women in high-to-low

order of fondness. He repeatedly makes proposals until he obtains an OK.

In the flowchart shown in Figure 2-2-69, the function K (p1, p2, p3) is used to compare levels of fondness.
Function K (p1, p2, p3): This function is for returning either p2 or p3 in the array element p1, whichever is higher in the level of fondness. K (F(2), 1, 3) → 1
Figure 2-2-69 shows the flowchart of the algorithm of the stable marriage problem.

Figure 2-2-69 Flowchart of the algorithm of the stable marriage problem



2.2.10 Approximate and probability algorithms

Algorithms are generally used to obtain true values or solutions. There are problems that require a very long time to solve, and there are problems for which no algorithm is available. In this case, algorithms for obtaining the solutions whose errors or whose possibility of containing mistakes are very small are used.

(1) Approximation algorithms

Approximation algorithms are used to obtain an approximate solution in those cases where obtaining a true solution to a problem is impossible or it takes a very long time. Both Newton's method and Simpson's method described in Section 2. 2. 8 fall under the approximation algorithm category.
This section describes the knapsack problem as one representative approximation algorithm.

[Knapsack problem]

You have n goods which are different in weight and value. You want to sell them in a town but you cannot put all of them into your knapsack. Find what combination of goods maximizes their value.

We apply the following values to explain how the algorithm works:
The number of goods (items) : 5
Weight of each item (kg) : {2, 3, 5, 7, 8} which apply to item 1, item 2,.....item 5 in that order
Value of each item (10,000 yen) : {2, 4, 8, 9, 10} which apply to item 1, item 2, ... item 5 in that order
Capacity of the knapsack : 10 kg
One way to solve this problem is to consider each combination of goods that would bring the total weight to 10 kg and compare the total value of goods in each combination. For this particular problem, the total value becomes highest if goods 1, 2 and 3 are packed into the knapsack, as shown in the table below:

Goods chosen	Total weight	Total value	
Goods 1, 2 and 3	2 + 3 + 5 = 10	2 + 4 + 8 = 14	← Maximum
Goods 1 and 5	2 + 8 = 10	2 + 10 = 12	
Goods 2 and 4	3 + 7 = 10	4 + 9 = 13	

Making the total weight equal to the knapsack capacity is not always the best solution. For example, if there were a sixth item, 9 kg in weight and ¥150,000 in value, the maximum value could be obtained by packing this item alone into the knapsack. To find the best solution, all possible combinations of goods must be considered. As goods increase in number, the number of combinations becomes enormous, and it takes a great deal of time to find the solution. As a solution to this, the approximation algorithm can be used.
The knapsack problem can be formulated as follows:

There are n positive integers a_1, a_2, a_3, ..., a_n, and b_1, b_2, b_3, ..., b_n, and a positive integer, c. Find a combination of x_1, x_2, x_3, ..., x_n that maximizes the total sum of b_ix_i (i = 1-n) where x_i ={0, 1} and the total sum of a_ix_i (i = 1-n) is equal to or smaller than c.

If this formula is considered in analogy to the problem previously discussed, a is the weight of goods, b is the value of goods, and c is the capacity of the knapsack. x is whether the good is packed into the knapsack or not. 0 means that the good is not packed and 1 means that the good is packed. Therefore, the knapsack problem and the solution can be expressed as follows:
a = | 2, 3, 5, 7, 8 |
b = | 2, 4, 8, 9, 10 |
c = 10
x = | 1, 1, 1, 0, 0 |
Using this formula, a search is made the 2^n times to check all possible combinations of 0, 1 in array x if the approximation algorithm is not used.
Using the approximation algorithm, however, unit values of all goods are first identified. A unit value means the value per weight and it is given by value (b_i)÷weight (a_i).

Unit values k = | 2/2, 4/3, 8/5, 9/7, 10/8 |
= | 1.00, 1.33, 1.60, 1.28, 1.25 |

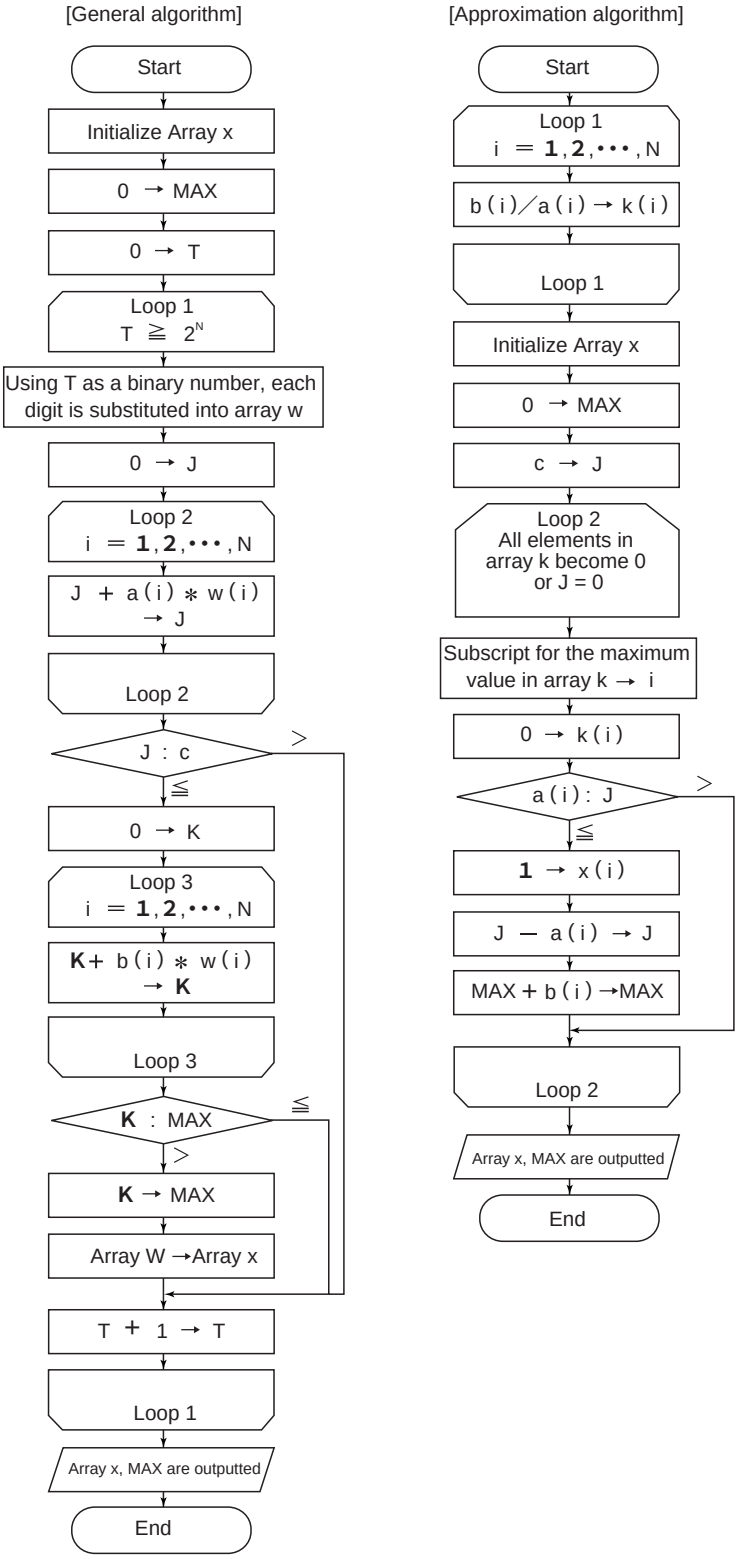
After all unit values are identified, goods are packed into the knapsack in the order of high to low unit values. Because goods cannot be divided, goods that exceed the unused capacity of the knapsack cannot be packed.

① Item 3 with the highest unit value is packed: Unused capacity = 10-5 = 5
② Item 2 with the second-highest unit value is packed: Unused capacity = 5-3 = 2
③ Item 4 with a unit value less than that of item 2 cannot be packed: Unused capacity <the weight of item 4
④ Item 5 with a unit value less than that of item 4 cannot be packed: Unused capacity <the weight of item 5
⑤ Item 1 with a unit value less than that of item 5 is packed: Unused capacity = 2-2 = 0

This way to obtain a solution does not necessarily give the best solution. If values are defined as {2, 4, 8, 11, 10}, the approximation algorithm gives the solution: (item 1, item 2, item 3) = ¥140,000, though we already have the best solution: (item 2, item 4) = ¥150,000. The approximation algorithm, however, is used in many different fields as a method of quickly obtaining an approximate value very close to the best solution.

Figure 2-2-70 shows the flowchart of the algorithm for solving the knapsack problem.

Figure 2-2-70 Flowchart of the algorithm for solving the knapsack problem



(2) Probability algorithm

The probability algorithm uses a probabilistic approach to find a solution using random numbers. A probabilistic approach is one in which the appropriateness of a solution is examined in terms of probability, or that a solution is found based on the probability of occurrence of a certain event.
This section describes the primarity test problem as an algorithm for examining the appropriateness of a solution in terms of probability, and the case of obtaining a circular constant π as an algorithm for finding a solution based on the probability of occurrence of a certain event.

① Primarity test problem

Primarity test problem is a process of checking given N (= 2^Sd+1, d is an odd number) and determining whether it is a prime number or not. To solve the primarity test problem, given N is divided using integers 2, 3, ..., $\sqrt{N}$, and it is considered a prime number if it is not divisible without a remainder by any of the integers. Although a correct solution can be obtained using this approach, a much longer time period is required to arrive at a solution if the value of N is large. As a solution, Rabin's algorithm, designed with the following theorem, is used:

[Theorem of prime numbers]

If N (= 2^Sd+1, d is an odd number) is a prime number, either of two conditions shown below stand for the arbitrarily chosen positive integer, a (>1):
- $a^d = 1 \pmod N$
- $a^{2kd} = -1 \pmod N$ where $0 \le k \le S-1$

If the arbitrarily chosen integer, a, does not meet the above conditions, N is considered a composite number, not a prime number. The probability that the arbitrarily chosen integer, a, for a composite number N can meet the above conditions is equal to or lower than 1/4. Therefore, if the arbitrarily chosen integer, a, meets the above conditions, the probability of the correctness of the judgement that N is a prime number is equal to or higher than 3/4.

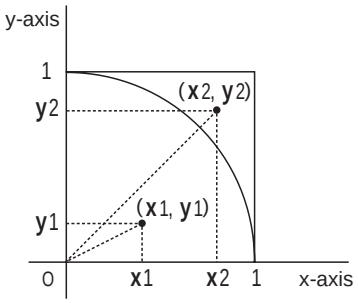
	Probability that the conditions can be met	Probability that the conditions cannot be met
Prime number N	100%	0%
Composite number N	25% or lower	75% or higher

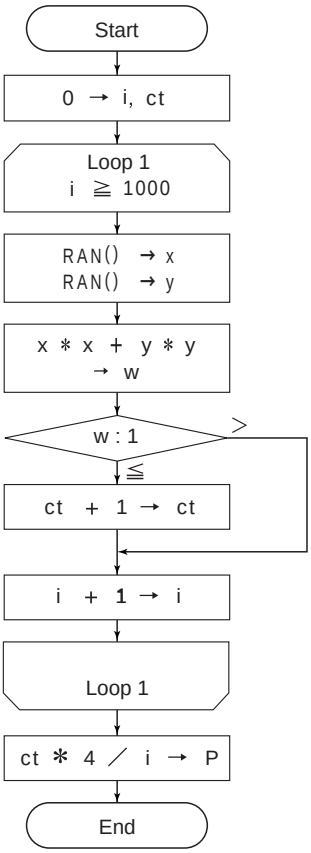
If this algorithm is used, the possibility remains that the solution given by the algorithm is incorrect. This type of algorithm is called the probability algorithm with bounded errors. For the above particular example, since the probability that errors will occur is sufficiently low, the solution given by the algorithm should be considered dependable. In addition to the probability algorithm with bounded errors, the probability algorithm with no errors is sometimes used. Although this algorithm can theoretically assure the correctness of a given solution, it sometimes takes too long to execute the algorithm process, or it ends up giving no definite solution.
A probability algorithm with no errors is the probabilistic quick sort algorithm. This algorithm is designed to improve efficiency by randomly rearranging input data using random numbers.

② How to find a circular constant π

Figure 2-2-71 shows a circle of 1 in radius. The area enclosed by the axes and arc is a fourth of that of the complete circle having a radius 1. Therefore, it is π/4 (= 1 × 1 × π/4). On the other hand, the area of a square enclosed by four lines, x = 0, x = 1, y = 0 and y = 1 is 1. If points on this square are designated in a random manner, the probability that these designated points are inside the circle is π/4.
In the flowchart shown in Figure 2-2-71, 1,000 points are generated using the function RAN ( ) that can generate random numbers between 0 and 1. Whether generated points are inside the circle or not is determined by measuring the straight distance from (0, 0) to each point. To simplify the calculation, the root $\sqrt{}$ calculation is omitted. In the case of the example shown in Figure 2-2-71, it is judged that (x1, y1) is inside if $(x1^2+y1^2) \le 1$ and that (x2, y2) is outside if $(x2^2+y2^2) > 1$.

Figure 2-2-71 Flowchart of the algorithm for finding a circular constant π





Because a circular constant obtained in this way contains errors resulting from the characteristics inherent in methods used to generate random numbers, it is generally used in the representation containing a standard error (solution ± standard errors).
An algorithm like this one, that uses random numbers to solve mathematical problems, is called the Monte Carlo method.

2.3 Evaluation of algorithms

Algorithms should be evaluated and selected based on certain criteria. If the most appropriate algorithm can be found to solve a problem, the work of programming can be done efficiently and a high-quality program can be created.
This section describes the following three algorithm evaluation criteria:

- Evaluation in terms of computational complexity (a criterion for evaluating efficiency)
- Evaluation in terms of validity (a criterion for evaluating reliability)
- Evaluation in terms of representation (a criterion for evaluating elimination of redundancy and improvement of processing speed)

2.3.1 Evaluation by computational complexity

Computational complexity, which is also called computational measure, quantitatively shows the workload needed to perform calculations.
Computational complexity is a measure used to mathematically clarify how much time and memory area a computer requires to perform calculations.
Computational complexity is expressed in *O* (order).
Example: If an algorithm deals with data in such a way that the data increases twofold, threefold, fourfold....etc., the execution time likewise becomes twofold, threefold, fourfold...etc., computational complexity of this algorithm is $O(n)$.

(1) Types of computational complexity

There are two types of computational complexity.

- Time complexity measure: The maximum time that an algorithm needs to process all data
- Space complexity measure: The maximum area that an algorithm needs to process all data

In general, computational complexity assume the worst case. Computational complexity discussed in this section also assume the worst case, if not otherwise indicated. If average computational complexity is $O(n\log n)$ and if maximum computational complexity is $O(n^2)$, as in the case of quick sorting, average computational complexity should be considered more important.
Computational complexity of an algorithm is represented as data sizes (for example, a square of a size n) are not restricted by hardware or software limitations.

(2) Computation example

We now take up linear search (sequential search) and binary search algorithms as examples. Assuming that the data size is n, the number of times of comparison and computational complexity are as follows:

① Linear search (sequential search)

- Minimum number of times of comparison: 1 (computational complexities: $O(1)$)
- Average number of times of comparison: $n/2$ (computational complexities: $O(n)$)
- Maximum number of times of comparison: n (computational complexities: $O(n)$)

② Binary search

- Minimum number of times of comparison: 1 (computational complexities: $O(1)$)
- Average number of times of comparison: $[\log_2 n]$ (computational complexities: $O(\log_2 n)$)
- Maximum number of times of comparison: $[\log_2 n]+1$ (computational complexities: $O(\log_2 n)$)

2.3.2 Evaluation by validity

The validity of an algorithm is a criterion for making a judgment about whether or not an algorithm satisfies program specifications (module design document, etc.).
The validity can be further divided into three categories:

- Partial validity: Whether segments or functions satisfy specifications or not
- Terminability: Whether a program can terminate after a specified number of execution steps, as defined by an algorithm; because an infinite loop must not occur.
- Total validity: Whether the whole algorithm satisfies specifications or not.

Although various methods are used to verify the validity of an algorithm, it is difficult to verify it completely. If the range of data is specified, the validity can be verified easily by detecting trouble that may occur when data outside the specified range is inputted.

2.3.3 Evaluation by representation

Algorithms must be evaluated in terms of not only computational complexity and validity, but also algorithm representation, which is called the elaboration of algorithm representation.

Example

- The same step is repeatedly executed: If repeated steps are defined as a subroutine, they can be described as one single step so that the flow of steps in an algorithm can be presented in a simple, easy-to-understand manner.
- It is necessary to increase the speed of an algorithm: Attention should be paid to steps that are repeated most frequently, and ways to increase the speed should be studied (if quick sorting is used, it may be necessary to quit the use of recursive calls and to work out an alternative).

2.4 How to design algorithms

Based on data obtained by analyzing a problem and the results of evaluation described in the previous section, an algorithm is designed with the most attention given to how a solution can be had with the highest level of efficiency. There are several methods used to design an algorithm. Representative methods are:

- Dynamic programming method
- Greedy algorithm method
- Reduction method

(1) Dynamic programming method

In the dynamic programming method, one problem is considered to be a set of more than one sub-problem. Steps to follow are:

1. A problem is divided into some sub-problems.
2. A solution to each sub-problem is found.
3. Some sub-problems are put together to make a slightly larger partial problem.
4. Above steps 2 and 3 are repeated until a solution method to the original problem is found.

(2) Greedy algorithm method

In the greedy algorithm method, which is used to find a solution to an optimization problem, the values of variables are changed by degrees depending on each present condition. For example, to find how the number of coins can be decreased to a minimum to pay a specified amount, the numbers of large- to small-value coins are determined using the greedy algorithm method.

Example

To pay ¥574,
One 500-yen coin - ¥74 remains
One 50-yen coin - ¥24 remains
Two ten-yen coins - ¥4 remains
Four 1-yen coins - Nothing remains

(3) Reduction method

In the reduction method, an algorithm originally designed has complexity of $O(n)$ is subjected to a reduction process of time length cn, so that it can be reduced to a smaller size to create a new algorithm of $O(n/x)$. If $O(n)>cn+O(n/x)$ cannot be satisfied, this algorithm is meaningless. Therefore, if n is very small, this algorithm cannot produce the expected results.

Exercises

Q1 **What is the term used to show a algorithm that searches the elements sequentially from the head to the foot of a table?**

a. Linear b. Binary c. Hash d. Heap

Q2 **Values in the array containing n elements, are sequentially compared with data X which is data to be searched. If data X matches a certain value in the table, "exist" is shown. Data X is stored in the position designated as index n+1.**

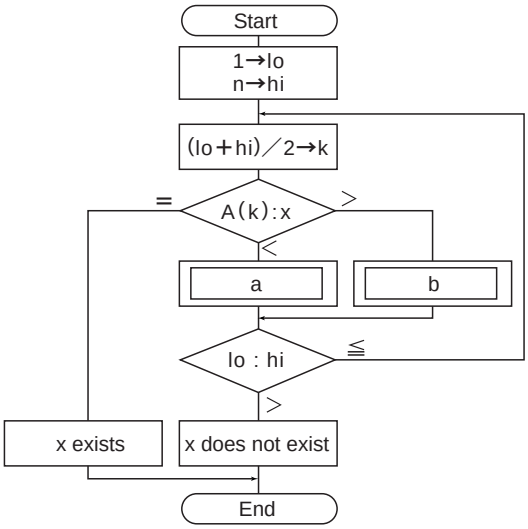
Subscript	1	2	3	...	i	...	n	n+1
Value	a_1	a_2	a_3	...	a_i	...	a_n	X

In the linear search algorithm shown below, what condition should be entered in the space [ ] ?

Step 1 1 is set to index i.
Step 2 If [ ], the routine jumps to step 5.
Step 3 1 is added to index i.
Step 4 The routine jumps to step 2.
Step 5 If index i is n or less, "exist" is shown.
Step 6 End

a. $i \geq n$ b. $i \neq n$ b. $i < n$ d. $X = a_i$ e. $X \neq a_i$

Q3 **There is array A which contain n data item sorted in the ascending order. The following flowchart shows the routine for retrieving data x from array A using the binary search method. Choose a correct combination of operations and enter them in the spaces a and b. Decimal numbers of a value obtained by division should be truncated.**



	a	b
a	k + l → hi	k - l → lo
b	k - l → hi	k + l → lo
c	k + l → lo	k - l → hi
d	k - l → lo	k + l → hi

Q4 **There is a table that has 2,000 different elements sorted in the ascending order of the keys. Using a key input from outside, this table is searched using the binary search method, and the elements that match the key are retrieved. What is the maximum number of comparison times needed before all matching elements are found? It is assumed that the table contains the matching key.**

a. 10 b. 11 c. 12 d. 13

Q5 **Which of the following comments made on search methods is <u>wrong</u>?**

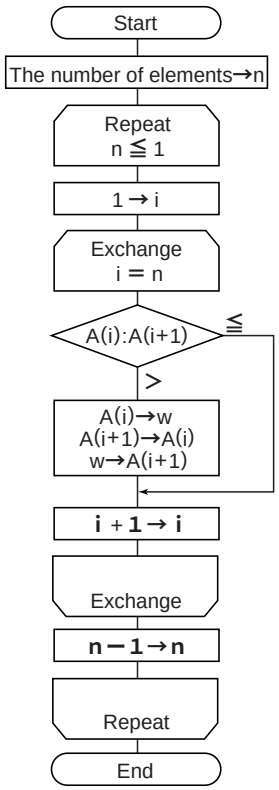
a. To use the binary search method, data must be sorted.
b. To search 100 data using the binary search method, the maximum number of comparison times that is needed to find the target data is 7.
c. If the linear search method is used, the number of comparison times does not necessarily decrease even if the data is already sorted.
d. If the number of data is 10 or smaller, the average number of comparison times that the linear search method requires, is smaller than that of the binary search method.
e. If the number of data is increased from 100 to 1,000, the number of comparison times increases to 10 times as large as when the linear search method is used. Using the binary search method, the number increases twice or less.

Q6 **Concerning data sort and merge, choose the proper combinations of words and enter them in the space [ ].**

Sorting data in the order of small- to large-value queues is referred as [A] [B]. If a target data sequence is in an auxiliary storage, this operation is called [C].
Integrating two or more files [D] in certain order into one file is called [E].

	A	B	C	D	E
a	descending	sort	external sorting	sorted	merging
b	ascending	merge	external merge	merged	sorting
c	descending	merge	internal merge	merged	sorting
d	ascending	sort	external sorting	sorted	merging
e	ascending	merge	internal merge	merged	sorting

Q7 **What sort of algorithm does the following flowchart show?**



a. Quick sort b. Shaker sort c. Shell sort d. Insertion sort e. Bubble sort

Q8 **It took 1.0 seconds for a certain computer to sort 1,000 data using the bubble sort method. How long will it take to sort 100,000 data of the same type? The time that it takes for a computer to bubble sort data is proportional to a square of the number of data that is defined as n.**

a. 1 b. 10 c. 100 d. 1,000 e. 10,000

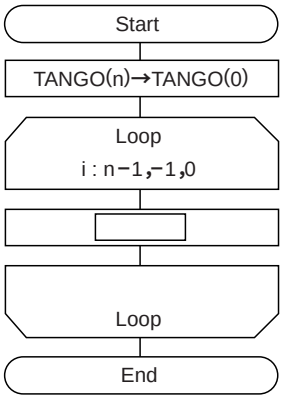
Q9 **Concerning data sorting methods, which of descriptions given below is correct?**

a. Quick sort is a method of sorting data in sub-sequences consisting of data items fetched at an interval and sorting smaller sub-sequences consisting of data items fetched at a smaller interval.
b. Shell sort is a method of sorting data by comparing a pair of adjacent elements and exchanging them if the second element is larger than the first.
c. Bubble sort is a method of sorting data by setting a median reference value, allocating elements with values larger than the reference value in one section and those with values smaller than the reference value in the other section and repeating this routine in each individual section.
d. Heap sort is a method of sorting data by representing an unsorted area as a subtree, fetching the maximum or minimum value from the unsorted area, moving the maximum or minimum value to a sorted area and repeating this routine to narrow down unsorted areas.

Q10 Which is an appropriate description of quick sort?

a. Compare and exchange are executed for two data away from one another at a certain distance. This distance is gradually and continuously narrowed to sort all data.
b. The first minimum value is found in data. The second minimum value is found in data in which the first minimum value is not included. This routine is repeatedly executed.
c. Data is divided into one group of data smaller than a reference value and the other group of data larger than a reference value. In each group, a new reference value is then selected and data is likewise divided into two groups based on the reference value. This routine is repeatedly executed.
d. Adjacent data are repeatedly compared and exchanged to allow smaller data to move to the end of a data array.

Q11 There is array TANGO whose index number starts with 0. n words are contained in TANGO (1) through TANGO (n). A flowchart below shows the steps for reorganizing the word table by shifting words in TANGO (1) through TANGO (n-1) backward, by one word, to put an n-th word in TANGO (1). Enter a proper step in the space [ ].



Note: Names of variables (initial value, increment and termination value) are shown as loop conditions.

a. TANGO (i) → TANGO (i+1)
b. TANGO (i) → TANGO (n-i)
d. TANGO (i+1) → TANGO (n-i)
e. TANGO (n-i) → TANGO (i)

Q12 From the descriptions made on Newton's method, which is known as an algorithm for obtaining an approximate solution of the equation f(x) = 0, choose the most appropriate one.

a. Although a function, f(x), cannot be differentiated, an approximate solution can be obtained.
b. As seen from the geometrical viewpoint, the method is to obtain an approximate solution by using a tangential line of y = f(x).
c. Two different initial values must be provided.
d. Whatever initial values are provided, an approximate value can always be obtained.

3 Internal Design

Chapter Objectives

Internal design is a process of designing a system based on an external design document. It deals with how a computer should operate to perform given tasks, while external design focuses on the functionality, efficiency and ease of use from the standpoint of users.
This chapter describes tasks of internal design to be done to design products.

① Understanding the purposes, important points and procedures of internal design
② Understanding the content and meaning of each internal design process
③ Understanding the creation and reuse of parts as well as the practical means for materializing the creation and reuse of parts

Introduction

The purpose of internal design is to determine the functions of software from the viewpoint of the developers. Internal design is a very important process, since a basic system design plan is formulated, and external design must be implemented in this process.
Although external design does not deal with a program, internal design concerns a program in which necessary functions defined through the development of subsystems in the external design stage, must be implemented. A program has a great influence, not only on program design and testing in the next process, but also on efficiency when a system becomes operational. Therefore, subsystems must be divided very carefully.
In this chapter, we learn how each item is done and what techniques are used, so that we can understand an external design document well, and are able to prepare reliable internal design work ourselves.

3.1 What is internal design?

Internal design is the design for invisible parts. It is the design as seen from a computer specialist's or system developer's point of view. In external design, a system is designed from the viewpoint of users. In internal design, hardware and other restrictions, as well as software requirements are considered in order to enable a computer to perform required functions.

3.1.1 Purpose of internal design and points to note

(1) Purpose of internal design

There are two purposes of internal design:

- Determining the functions that software should perform from the standpoint of developers
  The internal design process in all system development processes is where a system is first determined from the standpoint of developers. Because determined functions have a great influence not only on program design and testing in the next process, but also on efficiency when a system becomes operational, they must be determined very carefully.
- Ensuring the independence of each phase, so that one phase can be clearly distinguished from another
  The waterfall model has long been used as a means of system development. Having been refined and improved, it is the methodology still being used. Although some overlapping may occur between adjacent phases, each phase must be basically independent.
  Take the case where the work of program design was done as part of internal design, for example.

Example Partitioning a module was considered more important; so it was partitioned into programs and each program was further partitioned into modules.
As a result, the program is biased; it is "module partitioning" oriented rather than "process" oriented, causing internal design itself to lose flexibility. This may affect testing, operation and even maintenance stages.

(2) Points to note when doing the work of internal design

In doing the work of internal design, "why," "what" and "how" must always be kept in mind.

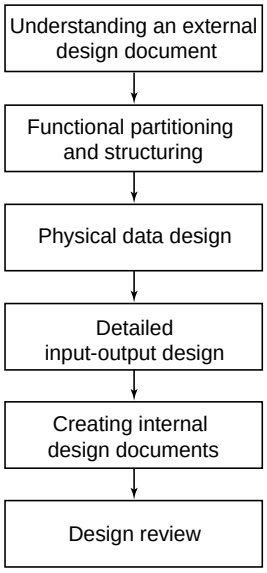
- Why: Defining subsystems
  Why each subsystem was divided in external design must be clarified.
- What: Defining the functions of subsystems
  What functions each subsystem performs in the overall system flow, must be examined from the viewpoint of relationships between subsystems.
- How: Defining details of subsystems in a program
  How input conditions, functional processing, and output conditions for the functioning of subsystems can be implemented in a program must be clarified.

In clarifying "why" and "what," as a principle, factors clarifying "how" must be left out of consideration.

3.1.2 Internal design procedure

Figure 3-1-1 shows the internal design procedure.

Figure 3-1-1 Internal design procedure



Details of each step are described below:

(1) Understanding an external design document

In the internal design stage, the content of an external design document (products or documents of the previous process) must be understood and organized and reflected as a whole in the system.
An external design document contains the following information:

- System overview (system flow, subsystem flow, etc.)
  This information is used as a base for dividing a subsystem into functional parts.
- Input and output design
  This information is used as a base for preparing detailed input output design.
- Logical data design
  This information is used as a base for preparing physical data design.
- System configuration and performance
  A system configuration (hardware, software, network, etc.), and estimated values of system

performance provided by external design, must be validated when a program is defined and physical data design is prepared in the internal design stage.

(2) Functional partitioning and structuring

Functions that were defined through subsystem development in the external design stage must be defined and detailed design for each operation must be prepared.

① Partitioning functions

Step 1: Why: Defining a subsystem
Why each subsystem was divided in external design must be clarified.

Step 2: What: Defining the functions of a subsystem
- A function of each subsystem is broken down into multiple functions.
- Functions are grouped.
- A function that each group performs, or a theme for each group is defined.

Step 3: How: Defining a program
The procedure for developing a theme is determined.

② Structuring functions

Step 1: Detailed functions of a program are reviewed.
Functions of a program are reviewed.

Step 2: Interface between programs
Data to pass between programs is clarified.

Step 3: Determining a process flow
The program processing procedure is determined.

(3) Physical data design

In this step of the internal design procedure, ways to access a database and files, and edit the files as well as the layout, are designed based on the data of logical data design created in the external design process.

<Physical data design procedure>

1. Analyzing data characteristics
2. Determining the logical organization structure of files
3. Determining the physical organization structure of files
4. Determining data storage media
5. Designing the layout of data items

(4) Detailed input-output design

In this step, the detailed data input and output are designed using special-purpose forms (spacing chart and other forms) based on the image prepared in the external design process with due consideration to hardware restrictions.
The graphical user interface (GUI) was introduced in this design work years ago. The optical character reader (OCR) and optical mark reader (OMR) are still widely used to simplify the work of data input.
Data output using a special-purpose form must be designed differently from that using a general-purpose form.

<Points to note when designing detailed input and output>

- Relevance of one data input or output task to another
  Input and output must be designed with consideration of the ease of performing each task.
- Choosing media with consideration given to the ease of data handling and management
  Input media include optical character readers, floppy disks, barcode readers, etc. Output media include printers, displays, etc. Proper media must be chosen with consideration given to the convenience of data handling and management.

- Taking hardware-related restrictions into account
  It is pointed out that an impact printer has a copy function, but it is noisy and the speed is slow, or that an optical character reader and optical mark reader are hardware-dependent due to design reasons. Hardware-related restrictions such as these must be taken into account.
- Handling errors of input data
  It is vital to check errors of input data. How errors can be checked and corrected must be determined.
- Ensuring the ease of maintenance, including proper control of input and output media
  Design must be prepared with considerations for the ease of maintenance, and proper control of input and output media in mind.

(5) Creating an internal design documents

An internal document is created as a product of the work shown above:

<The content of an internal design document>

- Internal design policy
- System configuration
- Program functions
- Screen layout
- Input and output layout
- File organization
- System performance
- Integration test plan

(6) Design review

The objectives of a design review in the internal design stage are:

- Validating that user requirements are satisfied
- Verifying that the consistency with external design is maintained, and that the prepared internal design data can be handed over to the program design stage

With these two objectives in mind, a design review is conducted in a thoroughgoing manner. A design review is important in any stage; program design and other subsequent processes are greatly affected by whether a design review conducted for internal design is successful or not.

<Points to note when conducting a design review>

- Products generated in the internal design stage are verified.
- The content of internal design is reviewed to confirm that it is consistent with the content of an external design document, and that all functions set forth in an external design document are implemented.
- Implemented functions are reviewed to confirm that they are appropriately divided and structured.
- Interfaces between programs are reviewed to confirm that they are properly designed.
- The appropriateness of physical data design is reviewed.
- Detailed input and output are reviewed to confirm that they are designed with the ease of user operation in mind.
- Missing functions and inappropriate points are indicated.

3.2 Functional partitioning and structuring

Functional partitioning and structuring is the work of partitioning the functions of subsystems into programs after reviewing an external design document and clarifying the functions of each program, the flow of operations involving more than one program, and the relevance of one program to another.
The process of structuring is indispensable to both internal design and system development.

3.2.1 Units of functional partitioning and structuring

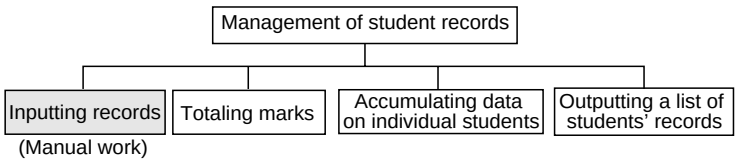
In the external design stage, a subsystem is partitioned into functions. In the internal design stage, a subsystem is partitioned into programs based on its functions. How the former is different from the latter is described in the following:

(1) Functions and programs

<External design>

A subsystem is developed to partition it into functions. In doing the work of partitioning, each function is considered the smallest, partitioned unit; programs and manual work are not considered.

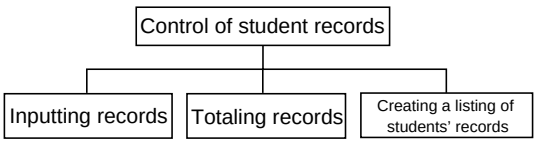
Figure 3-2-1 Developing a subsystem in external design



<Internal design>

Functions of a subsystem are scrutinized and they are partitioned into a program which is a unit of computer processing.

Figure 3-2-2 Functional partitioning and structuring in internal design



(2) Order

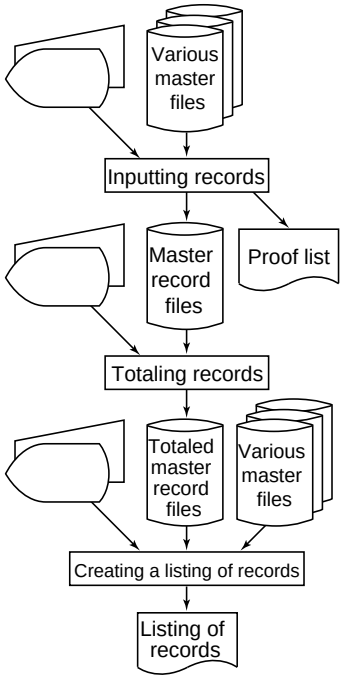
<External design>

A subsystem is developed, but functions of a subsystem are not ordered.

<Internal design>

The order of program execution is defined to implement functions and to execute operations efficiently. It is usually represented in the form of a process flowchart (see Figure 3-2-3).

Figure 3-2-3 Process flowchart



3.2.2 Procedures of functional partitioning and structuring

Procedures of functional partitioning and structuring are explained here by referring to the case shown below.

<Case> Monthly payroll calculation

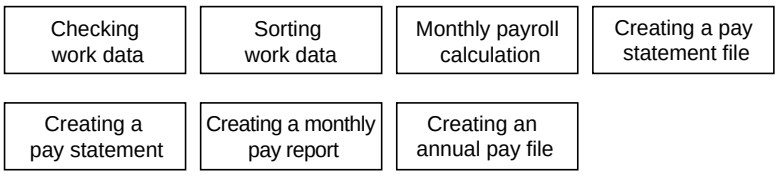
A pay statement, a monthly pay report, and an annual pay file are updated based on each employee's work and pay data.

1. Hours of overtime and absence in each month are input for an employee based on work data.
2. Input data is checked. If it contains errors, an error listing is outputted. In this case, input data is corrected and data is inputted again.
3. After errors are corrected, work data is sorted according to department and employee codes.
4. Various allowances and deduction amounts are calculated based on a master pay file, and other calculations are made.
5. A pay statement and a monthly pay report are printed based on a pay statement file which contains data obtained in steps 3 and 4 above.
6. An annual pay file is updated based on a pay statement file.

(1) Scrutinizing functions

Functions are scrutinized based on a new work flowchart (new physical DFD) and a work sub flowchart.
All functions that must be implemented in a subsystem are first scrutinized. In the case of external design, manual work that a computer does not execute is included in the functions. Functions discussed in this section, however, must be implemented into a program.
In this particular case, seven functions shown in Figure 3-2-4 are conceivable.

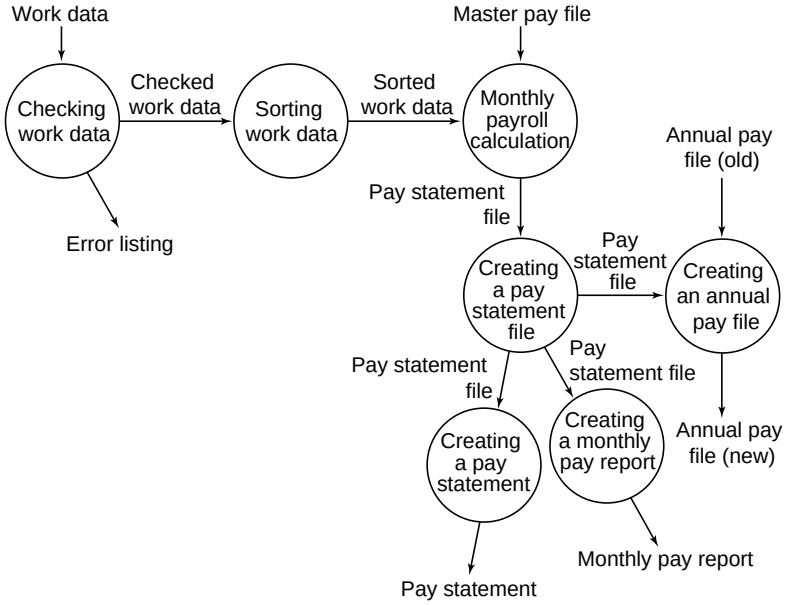
Figure 3-2-4 Scrutinizing functions



(2) Clarifying data flow

To partition functions, the flow of data to be processed must be clearly defined. Data flow is represented using logical data in the form of a data flow diagram (DFD) or a bubble chart.

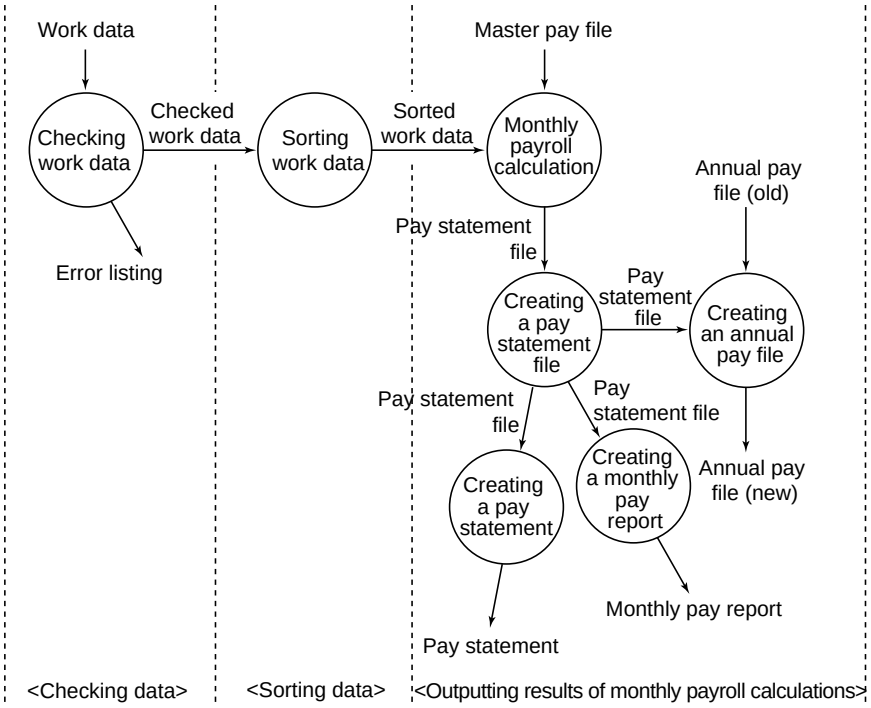
Figure 3-2-5 Clarifying the data flow (representing it in the form of a bubble chart)



(3) Grouping functions

Based on the flow of data shown above, required functions are grouped to allow a computer to perform each given task. This step might well be called a preparatory step, that will lead to the creation and reuse of parts. Functions in the flow shown under (2) above can be grouped as shown in Figure 3-2-6.

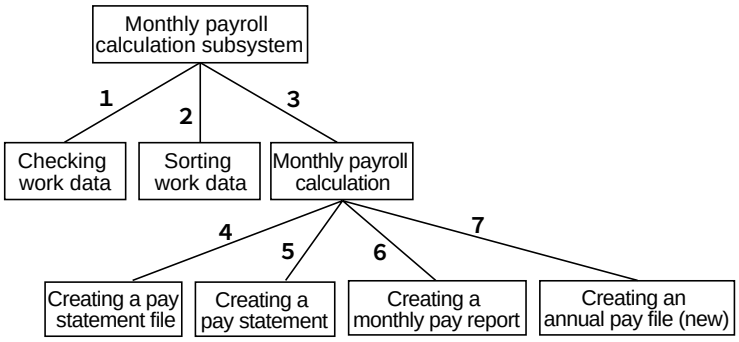
Figure 3-2-6 Grouping functions



(4) Hierarchical structuring

With attention given to the data flow, required functions are built into a hierarchical structure in stages. As shown in Figure 3-2-7, a program-to-program interface chart (parameter table) is prepared to streamline the flow of input and output data in a program.

Figure 3-2-7 Hierarchical structuring and a program-to-program interface



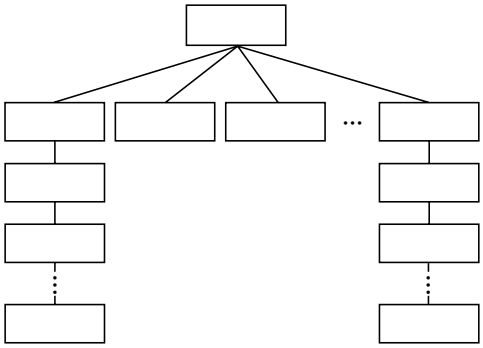
<Parameter table>

	Input	Output
1	Work data	Checked work data Error listing
2	Checked work data	Sorted work data
3	Sorted work data Master pay file	Pay statement Monthly pay report Annual pay file (new)
4	Sorted work data Master pay file	Pay statement file
5	Pay statement file	Pay statement
6	Pay statement file	Monthly pay report
7	Pay statement file Annual pay file (old)	Annual pay file (new)

<Points to note when building functions into a hierarchical structure>

- A deep hierarchy should be avoided. If a hierarchy is made too deep, the region of program design may be trespassed.
- Caution should be used to keep the number of programs in one hierarchy to a minimum.

Figure 3-2-8 Example of an improper hierarchical structure



(5) Determining program functions

For the lowest-order hierarchy shown under (4), details of the routine of a program are determined with attention given to the data flow. If possible, software parts can be assembled at this point.

Figure 3-2-9 Determining program functions

Checking work data
• Data is inputted and checked. • If work data contains errors, it is outputted to an error listing. • Data that contains no errors is outputted as checked work data to a file in auxiliary storage. • Data output to an error listing is corrected, and again outputted as checked work data.
Sorting work data
• Checked work data is sorted according to department and employee codes to create sorted work data (monthly data).

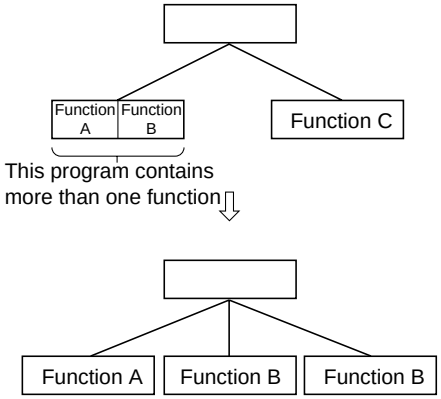
(6) Evaluating the results of partitioning

Partitioned functions are considered program structures and they are evaluated as programs. The results of the work described under (5) above are reviewed.

<Object of evaluating the results of partitioning>

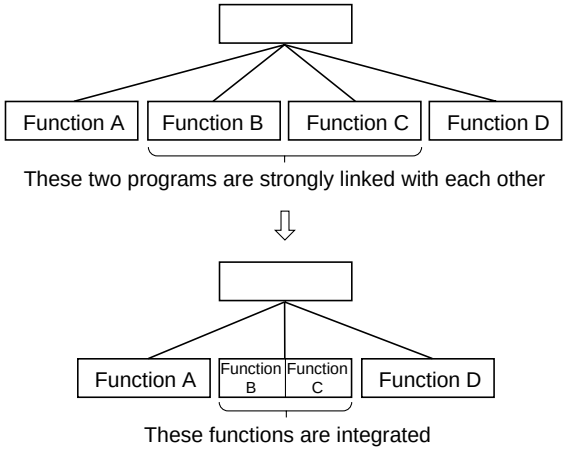
① If a program consists of two or more functions (increasing the strength of functions)

Figure 3-2-10 Increasing the strength of functions



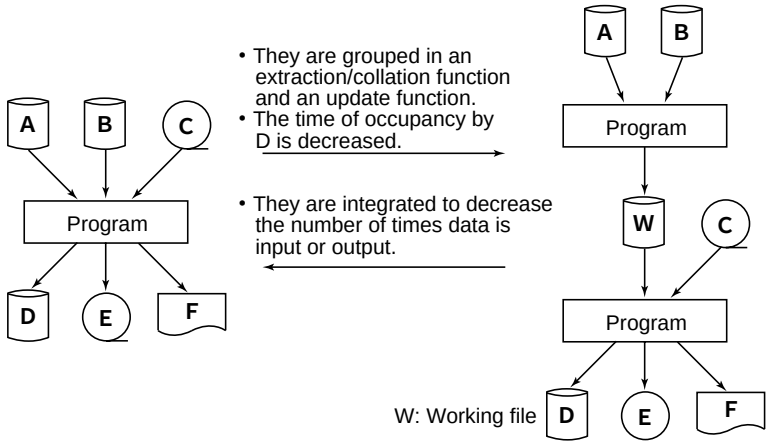
② If partitioned programs are closely linked with each other (integrating functions)

Figure 3-2-11 Integrating functions



③ If it is necessary to reconsider partitioned programs apart from the aspect of processing efficiency

Figure 3-2-12 From the aspect of processing efficiency



The independence of functions is used as a measure for evaluating the appropriateness of hierarchies and contained functions. In evaluating the independence of functions, two indicators shown below are used. This will be explained in more detail in section 4.2.3.

- Functional strength : Functional strength should be increased as much as possible.
- Functional linkage: Functional linkage between functions should be decreased as much as possible.

(7) Documenting functional specifications

Functions partitioned into programs are described in documents. These documents form part of an internal design document. They can be prepared in the form of a data flow diagram, a flowchart, hierarchy plus input process output (HIPO), or by any means that is designated as an internal standard.

<Documents>

Figures 3-2-13, 3-2-14 and 3-2-15 show the subsystem structure, program-to-program interface and program functions respectively in the case presented here.

Figure 3-2-13 Subsystem structure

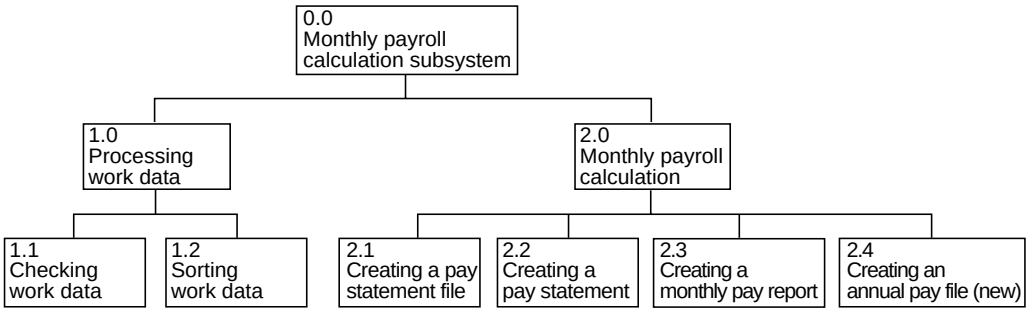


Figure 3-2-14 Program-to-program interface

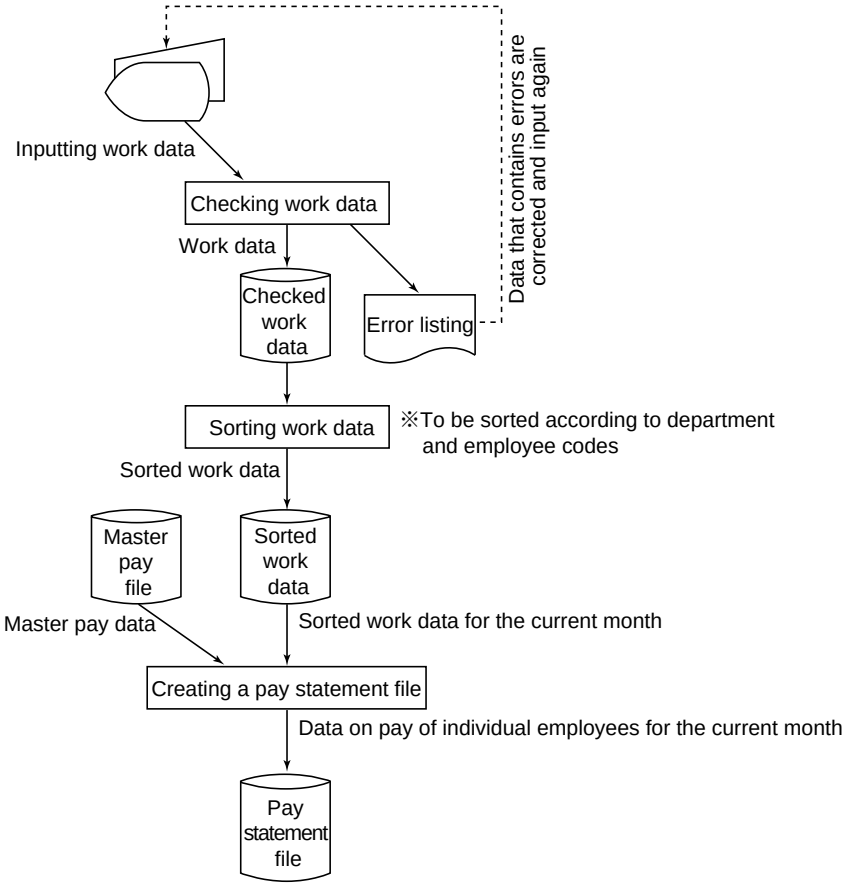


Figure 3-2-15 Program functions (to be created for each program)

①	Details of the task
②	How to perform the task in a program
③	Explanation of input and output parameters
④	Handling of errors
⑤	List of messages
⑥	Special remarks

3.2.3 Structured design method

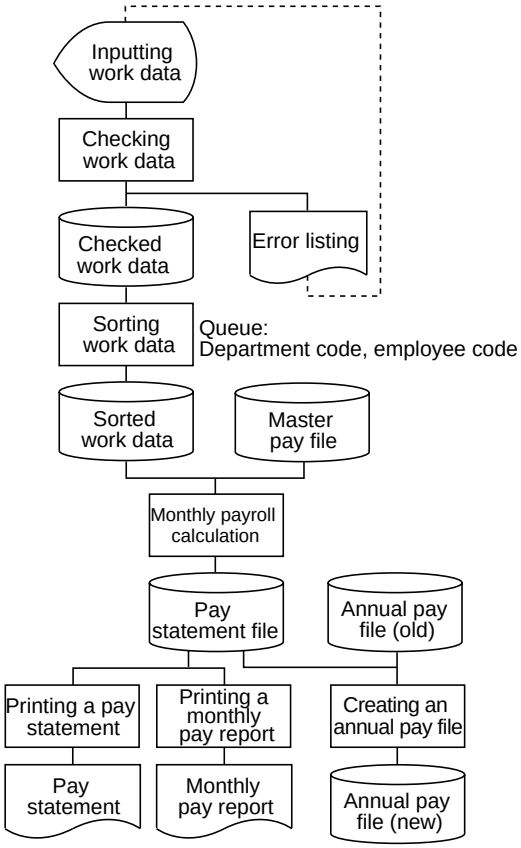
Techniques or tools used for structured design are as follows:

- Flowchart
- Data flow diagram (DFD)
- Hierarchy plus input process output (HIPO)
- Structured chart
- State transition diagram
- Bubble chart

(1) Flowchart

A flowchart is also called a process chart. (Flowcharts are usually used for downstream processes.)
A flowchart is created using symbols which are standardized, by defining and analyzing complex problems and task processing procedures.

Figure 3-2-16 Flowchart (process chart)



(2) Data flow diagram (DFD)

A data flow diagram (DFD) is used to express the flow of task processing or system operation schematically. In writing a data flow diagram, attention should be given to where data being used is generated, how it is processed, and where it is stored. A data flow diagram is an easy-to-use technique, and effective in preventing an overview of user requirements. It has also recently been used to show the data flow in a subsystem.

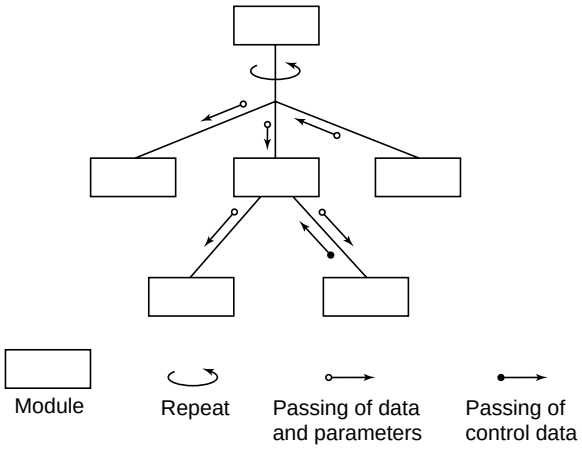
(3) Hierarchy plus input process output (HIPO)

A hierarchy plus input process output (HIPO) is a documentation support tool, used as an auxiliary means for supporting the work of design.
Because it systematically divides system or program functions into parts in sequential order, it can meet various different needs of managing personnel, designers, departments using a program, personnel responsible for development or maintenance, etc.

(4) Structured chart

A structured chart is used to represent the functions of each program in an easy-to-understand manner. Using a structured chart, master-slave relations between programs can be represented as a hierarchical structure.
A structured chart is very easy to understand when used to represent the interfaces between programs comprising a system or the structure of each program.

Figure 3-2-17 Symbols used in a structured flowchart



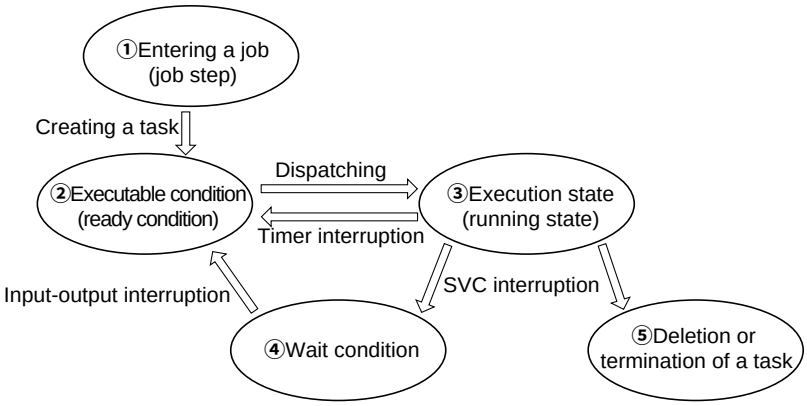
(5) State transition diagram

A state transition diagram is used to show how a state changes (transition). It is suitable for organizing or expressing the state of an operating system or a communication control program. (See Figure 3-2-18.)

<How to write a state transition diagram>

- States that can possibly take place are enclosed within a circle or a rectangle.
- States are connected with an arrow in the order they take place.
- Conditions that cause a specific state to take place are written alongside an arrow.

Figure 3-2-18 State transition diagram



(6) Bubble chart

A bubble chart is used to analyze or divide a designed system or program.
The points to note when analyzing a system or program:

- What data is generated where?
- How does data flow?
- How is data processed?

The work of structured design can be streamlined by using the techniques or tools (1) through (6) in combination.

3.3 Physical data design

In the physical data design process, items and characteristics (usage and increase rates) of data in a database, files or the table structure in memory which are determined in the logical data design process, are reviewed and the physical organization and layout of data are designed.
Data and storage media should be examined with attention given to their respective characteristics, and the most appropriate storage media must be selected so that data characteristics can be used to advantage, and records can be organized into a high-efficiency layout.

3.3.1 Physical data design procedure

The physical data design procedure was described in Section 3.1.2. This section describes the content of the physical data design work, important points to note, and various characteristics in greater detail.

(1) Analyzing data characteristics

Characteristics of data to handle are closely analyzed, and design is prepared in such a way that characteristics of respective data can be used to advantage. The following points must be kept in mind during design:

<Important points to note>

- Characteristics and uses of data
  - Is it a master file or a transaction file?
  - How long must it be retained (retained for a long time or retained temporarily)?
  - Is it used as backup data or retained as an update record?
- Adding, deleting or changing data
  - Amounts of data that are added, deleted or changed during a specified period of time
  - The content of a task to be executed (in the order of keys or randomly)
- Updating
  - Is it updated daily, monthly, annually or according to a specified period of time?
- How data is used
  - Is it used for batch processing?
  - Is it used for online processing?
- Maintainability
  - How data can be restored if it is destroyed?

(2) Determining a system for organizing data into a logical structure

If a storage medium or a physical organization is adopted before a system for organizing data into a logical structure is determined (a master or transaction file, etc.), the efficiency of operation, the work of design change, etc., will be adversely affected.

- Acceptable case
  A system for organizing data into a logical structure is first determined, then a physical organization is determined.
  Need: A commodity file must be designed as a master file.
  Result: Commodity files are stored on a magnetic disk and used as indexed sequential files.
- Unacceptable case
  A physical organization is first determined, then a system for organizing data into a logical structure is determined.
  Need: Files must be stored on magnetic tape and used as indexed sequential files.

Result: Commodity files should be used as master files, but magnetic tape is inconvenient to use. What should we do?

It is recommended that a system for organizing data into a logical structure be first determined, then a physical organization can follow.

<Points to note>

- Scope of data use
  Is data intended for use only in a system being developed? Or must it be used in a different system as a master file? Or was it used only for programming?
- Temporary data or data to save
  Is processed data used only temporarily? Or should it be handled as data to save?
- Is data serially processed and does it increase gradually or quickly?
  Is it the type of data that increases gradually and does it require the retention of update records?

(3) Determining data storage media

① Determining storage media

Physical data storage media include:

- Magnetic disk (hard disk)
- Magnetic tape
- Flexible disk (floppy disk)
- Magneto-optical disk (MO)
- ZIP drive
- Streamer (to back up data on a hard disk)

A magneto-optical disk, ZIP drive and streamer are storage media that have recently made an appearance. They have a large storage capacity (tens of megabytes to hundreds of gigabytes) and are suitable for storing multimedia data.

The following points should be considered in selecting the most appropriate storage medium:

- Storage capacity
- Characteristics (data access method)
- Access speed
- Maintenance, handling, price, etc.

If you want to directly access data using keys, only a storage device that allows direct access, such as a magnetic disk, can be used.

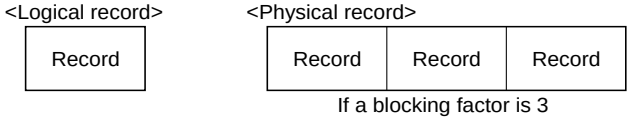
② Calculating storage capacity and access time

In calculating storage capacity and access time, attention should be given to a difference between logical and physical records, and a blocking factor.

<Types of record>

- Logical record: A unit of data to be processed by a program; it is one of the objects to be dealt with by field design.
- Physical record: A unit of data to be read from and written into storage media

Figure 3-3-1 Logical and physical records



<Points to note>

- A blocking factor 3 means that one block contains three logical records.
- A program handles logical records, while data input into, and output from actual storage media are physical records.
- There is an interblock gap (IBG) between blocks.

Figure 3-3-2 shows how storage capacity of a magnetic disk and access time can be calculated.

Figure 3-3-2 Calculating storage capacity of a magnetic disk and access time

1. Storage capacity of a magnetic disk

<Specifications of a magnetic disk>

The number of cylinders per disk pack	800 cylinders
The number of tracks per cylinder	19 tracks
Track capacity	24,000 bytes

(How to calculate)
The storage capacity of a magnetic disk pack can be calculated as follows:
Storage capacity of a track x the number of tracks x the number of cylinders
Example: 24,000 bytes per track x 19 tracks per cylinder x 800 cylinders per disk pack
= 364,800,000 bytes per disk pack

2. Access time of a magnetic disk

<Specifications of a magnetic disk>

Effective storage capacity per track	15,000 bytes
Average seek time (average positioning time)	25 milliseconds
Rotation speed	3,000 rotations per minute
Length of one record	15,000 bytes

(How to calculate)
To obtain access time, three elements are required: average seek time, average search time and data transfer time. Because only the average seek time is described in a specification, average search time and data transfer time must be calculated.
Average search time can be easily calculated from the rotation speed of a magnetic disk. If the rotation speed is 3,000 per minute, the time that it takes for a disk to make one rotation is 20 milliseconds. Because average search time is half the time that it takes for a disk to make one rotation, it can be calculated as follows:
20 milliseconds÷2=10 milliseconds
Because 50 rotations per second can be deduced from 3,000 rotations per second, data transfer speed can be calculated as follows:
50 tracks per second x 15,000 bytes per track = 750 x 10^3 bytes per second
The time that it takes for one record to transfer data of 15,000 bytes can be calculated from the above data transfer speed as follows:

$$\frac{15\times10^3 \text{ bytes}}{750\times10^3 \text{ bytes per second}} = 0.02 \text{ seconds} = 20 \text{ milliseconds}$$

Therefore, access time can be calculated as follows:
25 milliseconds + 10 milliseconds + 20 milliseconds = 55 milliseconds

(4) Designing the layout of a record

In the record layout design process, data items are laid out.

① Field (item) design (logical record design)

In preparing field design, the following points should be noted:

- Order of items
  - Items should be sequentially arranged, starting with key items. If a key item is comprised of two or more fields, these fields should be laid out in succession.
  - Items should be arranged in the order of high- to low-level importance or high to low frequency of use.
  - Items should be arranged in an easy-to-use manner.
- Field size and data types
  - A uniform date format should be used (yymmdd, yyyymmdd, etc.). this is the International date and time standard it should be designed so that any one element (a year, a month or a day) in the date format can be handled individually and, at the same time, all elements can be used as a group.
  - Numerical items should be packed decimal numbers.
- Margin fields
  - A reserve area should be established to provide for future expansion (to increase the number of fields).
  - Margin fields should be used to match the length of records (to match 80 bytes or 256 bytes).

- Margin fields are established in two ways:
  a. Free space is established at the tail end
  b. Free space is established immediately after a file that is expected to be used in future expansion. This free space should have the same size as the current data field.

- The ease of coding should be taken into consideration.

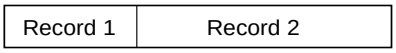
② Record types

There are the following record types:

a. Fixed length record
In the case of a fixed length record, each record comprising a file has the same length. If the records in this type of file are blocked, the length of each block becomes the same.
A fixed length record is widely used because of its easy-to-use characteristic.

b. Variable length record
In the case of a variable length record, the length of each record comprising a file is not the same. For a file created in the variable length record format, each record contained in the file must be prefixed with a record descriptor that shows the length of the record. When blocking records, each block must be prefixed with a record descriptor that shows the overall length of the block.

Figure 3-3-3 Variable length record



c. Undefined length record
In the case of an undefined length record, the length of each record is determined by a program, since the length of each individual logical record is different.

③ Record layout

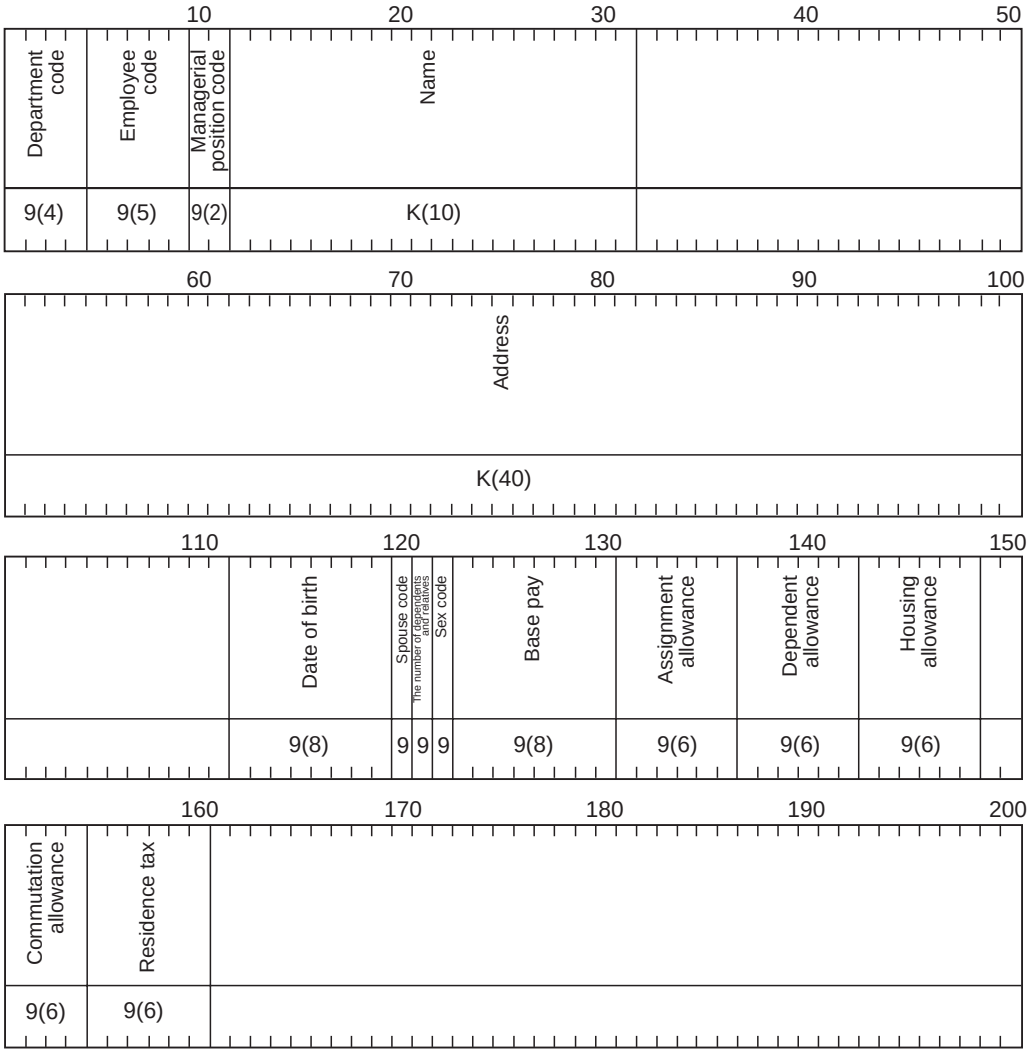
The following points should be considered when designing the layout of a record using a record (file) layout form:

- As a record name, an easy-to-understand character string that can represent the contents of data must be used.
- X (alphanumeric character), 9 (numeral) and K or G (Japanese type) must be entered to indicate the type of attribute.
- The number of digits used to form a data item must be entered. One digit is one alphanumeric character (one byte). In the case of Japanese type, one character corresponds to two bytes, and it is therefore recommended that the number of digits as well as the number of characters be entered to make each data item easy to understand.

Figure 3-3-4 Example of record layout

Record layout	Date	/ /	Created by		Approved by	
System name	Pay calculation system		Subsystem name	Pay master calculation, monthly calculation, bonus calculation, year end adjustment calculation		

File name	Pay master file	Record name	Pay master record
Record length	160	Blocking factor	6	Record type	Fixed length



3.3.2 Physical data organization

(1) Types of file organization

Types of physical file organization are determined based on the results of data characteristics analysis, logical organization methods, program functions, etc.
Main types of physical file organizations are:

- Sequential organization file
- Direct organization file
- Indexed sequential file
- Partitioned organization file
- Virtual storage organization file (VSAM file)
- Table

A type of physical file organization most suitable for a given purpose should be selected.

① Sequential organization file

In a sequential organization file, records are sequentially stored in consecutive positions on a storage medium.
Records are normally arranged in ascending or descending order with a certain data item designated as a key, or in the order records are created.

<Characteristics>

- This type of file can be created on any type of storage media, a magnetic disk, magnetic tape, etc.
- File reading and writing starts with the head of records; random access is not supported.
- Because records are stored physically and successively, a new record cannot be added or inserted. Therefore, to update a certain file on magnetic tape, a new file must be created. In the case of a magnetic disk, however, data can be rewritten to an original file.
- The access speed is fast.
- This type of file is suitable for batch processing or data backup.

② Direct organization file

Direct organization files can only be used on direct access storage devices (DASD), such as magnetic disks.

<Characteristics>

- The address of data is calculated using a special address calculation method called the hash method.
- The same address may occur when address conversion is executed. Such an address is called a synonym.
- Direct access is supported. In some cases, sequential access is also supported.
- Storage efficiency is high.
- Processing speeds are fast.
- This type of file is suitable for online real-time processing that requires high-speed access.

③ Indexed sequential file

A file that allows reference to a data storage position index as well as the reading of specific data is called an indexed sequential file. This type of file can only be used on direct access storage devices (DASD).

<Characteristics>

- A data memory area consists of an index area, a prime data area and an overflow area.
- An index area consists of a master index, a cylinder index and a track index. These indexes are designed to match the keys of each record so that data can be searched and retrieved. Therefore, key values must be arranged in ascending order.
- Both sequential and direct access are possible.
- As an index area must be first searched, the level of storage efficiency is low.
- Processing speeds are slow.
- This type of file is suitable for a master file for paperwork tasks.

④ Partitioned organization file

In a partitioned organization file, a sequential file is partitioned into subfiles called members, and a directory is created to show the start position of each member so that each individual member can be directly accessed.
This type of file can only be used on direct access storage devices (DASD), such as magnetic disks, as in the case of the direct organization file.

<Characteristics>

- Although each member can be directly accessed, records contained in a member are sequentially accessed.
- Access speeds are almost the same as those of the sequential organization file.
- The level of storage efficiency is lower than that of the sequential organization file since directories are created.
- This type of file is suitable for a program file and various libraries.

⑤ Virtual storage organization file (VSAM file)

A virtual storage organization file can be used with an operating system that has a virtual memory function. This type of file can only be used on direct access storage devices (DASD), such as magnetic disks, as in the case of the direct organization file.

<Characteristics>

- This file is designed by integrating three access methods used for the sequential, direct and indexed sequential organization files.
- Because the type of file is controlled by an operating system, a system developer does not need to keep block lengths or record lengths in mind.
- This type of file is further divided into three types:
  Key sequence data set : KSDS (equivalent to the indexed sequential organization file)
  Entry sequence data set : ESDS (equivalent to the sequential organization file)
  Relative record data set : RRDS (equivalent to the direct organization file)

⑥ Table

A table is a work area established on a main memory unit. In the case of COBOL, the following entries are defined in the Data Division:

- Table name
- The number of tables
- Name of data item in a table
- Type of data item (alphanumeric data, numeric data, etc.)

<Characteristics>

- Because the table resides in memory, the access speed is very fast.
- Both sequential access and direct access are possible due to the program structure.
- If a computer is powered off, data is lost. Therefore, the table cannot be used as a master file.
- The table is generally used as an area for performing a high-speed search on data, or totaling data.

(2) Processing mode

How data is formatted and retained affects system usability. In developing a data structure in a storage medium, various factors must be considered, i.e., the access method, access efficiency, space efficiency, localization of failure, efficiency of buffer amounts in memory, etc. With all these factors in mind, handling of files and use of a database should be examined.

Performance, price, troubleshooting and other factors associated with general system design should be determined along with a consideration of system requirements. It is difficult however to achieve access efficiency and storage capacity, or search efficiency and update efficiency (write efficiency) at the same time. These efficiency-related factors must be determined so that they can be kept in balance.

Also files are interpreted differently, depending on the operating system in which a created software program runs. It is necessary to verify the file management service functions provided by the operating system. According to the interpretation of UNIX, one of the major operating systems, the content of a file should be recognized by a program and does not need to be recognized by the file system of UNIX. According to the interpretation of MVS however, a logical group of data is recognized as a record and a base unit of file operation, and the logical data structure designed by a programmer is supported by the file system.

3.4 Detailed input-output design

In the external design stage, the screen and report are designed to allow a user to recognize the image of screen and output. Precise arrangement of items, font sizes, color, icon types and other details that are subject to hardware- or system-related constraints are not created in the external design stage.
In the internal design stage, such details are based on the image created in the external stage. Internal design is prepared with consideration given to hardware-related constraints. Because internal design is based on the work of external design, the main theme is to achieve ease of use for users.

3.4.1 Detailed input data design

The data input form is one of main input data. This section describes the detail design of the data input form.

(1) Design objectives

A user uses the data input form. To design a well thought-out, refined data input form, the conditions of a user and the speed, accuracy and ease of data input must be kept in mind.
Objectives that must first be achieved in the design of the data input form are:

- Ease of writing
- Ease of computer (keyboard) input

(2) Design

The size of a form, the number of copies, the colors used in a form and the framework of the data input form must be determined.

① Ease of data input

- The amount of writing should be kept to a minimum
  The following points should be considered to save a user the time and labor of data input:
  - Selective method (a user is asked to make a choice and encircle one item)
  - Minimum number of input items
  - Duplicating
  - Fixed items to be printed on the data input form
  - Less frequently used input items to be grouped in the remarks space
- Input items
  Input items should be arranged according to the sequence of thought.
  - A user should be guided to first enter data on main input items.
  - A user should be guided to enter data from the upper left to the lower right.
  - Input items should be legible and easy to understand.

Figure 3-4-1 How each item can be made legible and easy to understand

① Make clear what a user is expected to write

Date_____	This is rather ambiguous
Date: year-200_, month-_____, day-__	This is easy to understand

② Guide a user to prevent him/her from making an input mistake

Amount _____	This is rather ambiguous
Amount due \ ____, _____, _____ (not including consumption tax)	This prevents a user from making an input mistake

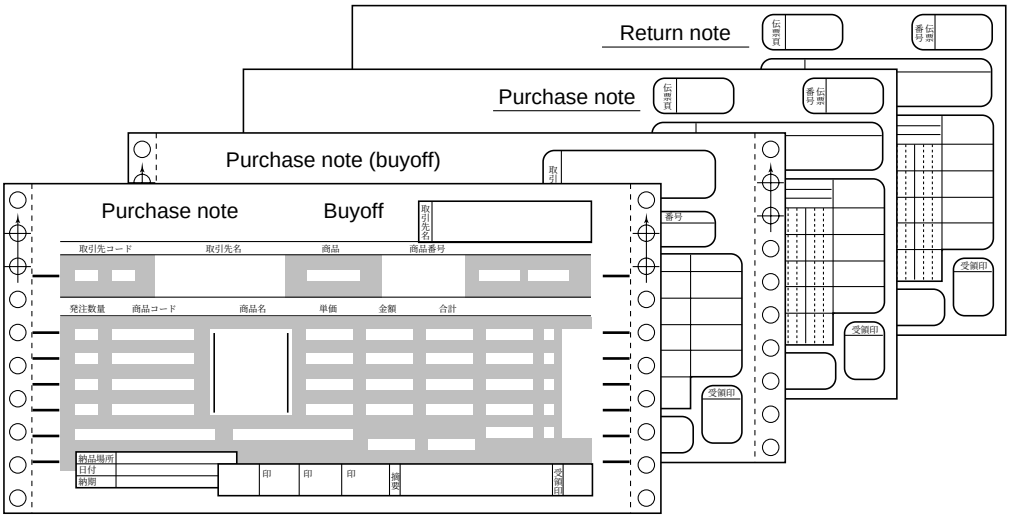
② Ease of computer (keyboard) input

The accuracy of data can be ensured if data is input where it is generated (at a workplace). Therefore, the data input form must be designed based on the purpose of each data processing task, and the content of new system operations (data flow) on the assumption that it is used at each workplace.

- Colored paper should be used as the data input form to allow a user to recognize it on sight.
- Each input item should be enclosed in a thick line to make it easily discernible.
- A separate item should be established to allow a user to distinguish the data input form that has already been input from the one that is not.

A data input form is shown as follows.

Figure 3-4-2 Data input form (exclusively for OCR)



③ Designing an OCR form

Using an OCR (optical character reader), you can directly input data by feeding a form into a reader. At the current stage of OCR technology development, misreading often occurs even if data is written correctly and legibly.

There are some compromise measures taken: care is exercised to keep the OCR form clean (to prevent misreading) or explanations about character styles are given.

(3) Checking input data

Input data must always be checked since the probability is high that a user made input mistakes, or that an OCR or OMR misread the input data. Mistakes in input data can lead to not only incorrect results, but also the abnormal shutdown or runaway of a system.
Data itself may contain mistakes. Input data is past a specified input time limit or some data is missing. All these problems are considered input mistakes.

① Correcting data errors

When a data error is found, it must be corrected immediately. The actions to be taken when errors are found must be clearly defined.

- Mistakes found before data is input to a computer
  If a mistake is found when the data is entered, the data input form must be returned to the department that filled it out. An operator must never correct the mistake. If an operator corrects it on his or her own judgment, there is the possibility that an unexpected, serious problem may happen later. Mistakes found at this stage have minor effects on data processing operations.
- Mistake detected by a program
  If a mistake is found when a program has rejected input data, data processing is already under way. Therefore, at what point a mistake should be corrected must be determined.

② Handling of data errors

- Ignoring data errors and continuing data processing
  If there are only a few data errors in tens of thousands of data being processed for a statistical purpose, such errors have only a minimum effect on the overall processing operations, and can be ignored. If all data must be registered, or if the ratio of erroneous data increases, this method of error handling can't be used.
- Performing data processing after all the data errors are corrected
  This method of error handling is used if it is handled under very stringent conditions and not one mistake is permitted (data registration, for example). The work involved to correct data errors this way however, is very time-consuming.
- Use only correct data and continue data processing
  Using only correct data, data processing is continued. After it is completed, data errors are corrected, and merged with correct data. The main process, starts when all data are made available.

③ Data check methods

Figure 3-4-3 shows main data check methods.

Figure 3-4-3 Data check methods

Data check methods		Points to check
Validity check	Format check	Data is created in a specified format
	Numeric check	Data other than numeric characters is not input in numeric items
	Limit check	Data does not reach or exceed specified limits
	Range check	Values stay within specified limits
	Overflow check	Data does not exceed a specified data length
	Check digit check	Check digits are attached to codes to verify that read numbers are correct
File check	Sum check	The result of calculations performed using a desk calculator is compared with the result of calculations performed by a computer.
	Sequence check	Data is sequentially arranged in accordance with a specified standard
	Balance check	In a pair of numbers, one number matches the other, as in the case of numbers in debit and credit sides
	Count check	The actual number of data matches the number of data calculated by a computer
Sight check		Output data is visually checked to find data errors

3.4.2 Screen design

Screen design is one of the important factors that a user considers to determine whether a system is well or poorly designed, as explained in sections dealing with external design. The screen is the face of a system for a user. The screen must be designed with the highest priority given to the ease of use.
Because the graphical user interface (GUI) techniques using a visual language have become mainstream computer techniques, this section describes the screen design using the GUI techniques.
The screen design using the GUI techniques is carried out by prototyping. The screen design procedure is shown below.

<Screen design procedure>

1. A standard layout of the screen, requirements that must be met to perform specified functions, and the level of users' proficiency are clarified.
2. A prototype screen is created using a visual language.
3. Users evaluate the prototype screen. Modifications and improvements are made to complete the prototype screen.

(1) Graphical user interface (GUI)

The GUI uses intuitive icons (pictorial symbols), pop-up menus, drop-down menus, etc., to allow tasks to be performed. (See Figures 3-4-4 and 3-4-5.) Specifically, an arrow is moved using a pointing device (a mouse and other devices) to a desired position on the screen, where an icon or an item in a menu is selected to instruct a computer to perform a given task.

Figure 3-4-4 Icons and a pop-up menu

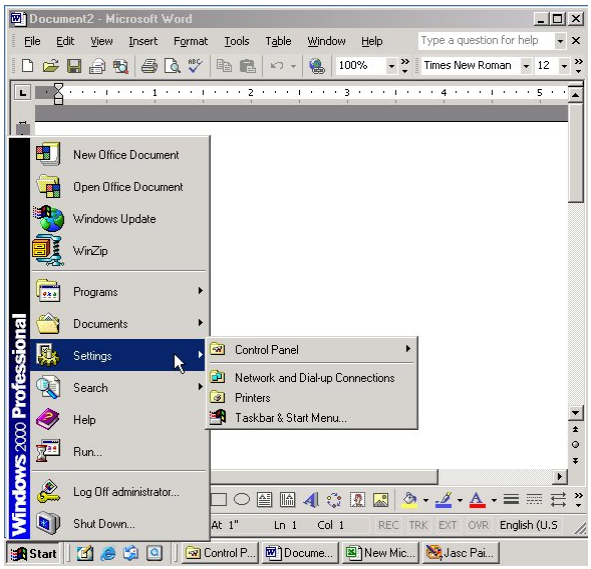
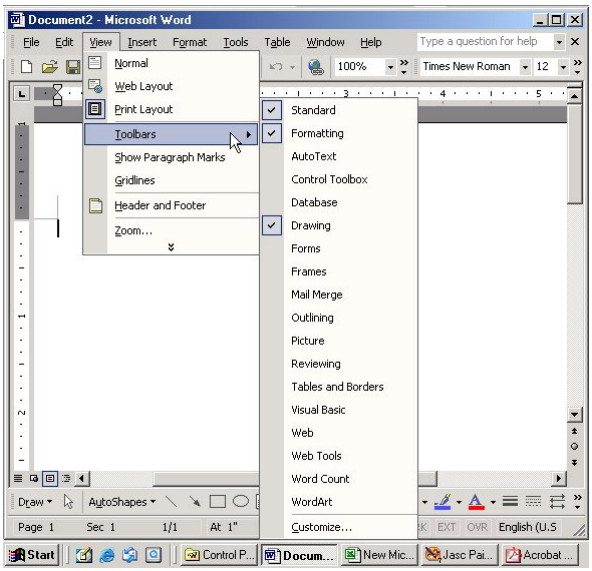


Figure 3-4-5 Drop-down (pull-down) menu



The GUI functions are built into the operating system of a personal computer. In the case of UNIX, however, the GUI functions must be installed additionally. The representations, meanings and operations on icons are different depending on the GUI used. Because standardization has been done for each GUI, design productivity has improved and the number of errors has been reduced.

(2) Terminology related to the GUI environment (Windows)

Before describing about screen design, the terminology related to screen design etc. in the GUI environment is briefly explained.

① Event-driven program

All applications used in the GUI environment (Windows, etc.) are event-driven programs. Applications developed before introduction of the GUI were designed with the sequential control scheme. With the event-driven programs, such actions as clicking a mouse or dragging a cursor can cause a specific event, which in turn activates a corresponding predefined procedure.

② Windows

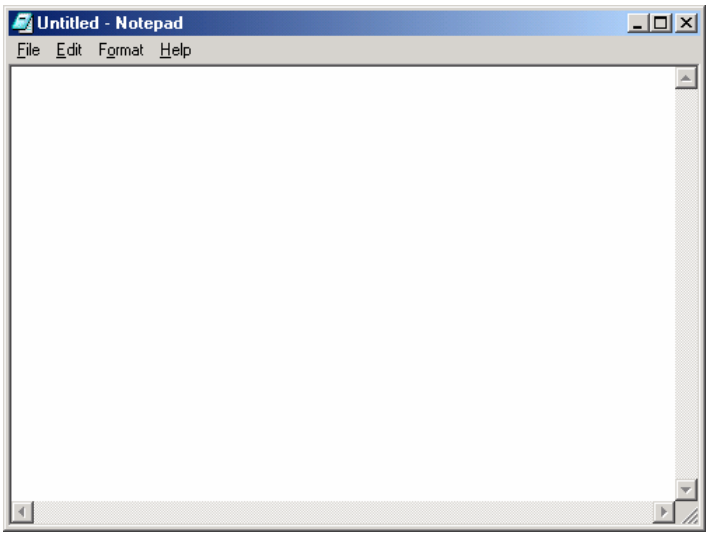
The base of the GUI is windows. They vary from application to application.

③ Documents

The contents shown in one window are called documents. The types of the documents are called document interfaces and are classified broadly into two types:

a. Single document interface (SDI)
A single document interface (SDI) shows a single document inside one application. (See Figure 3-4-6.)

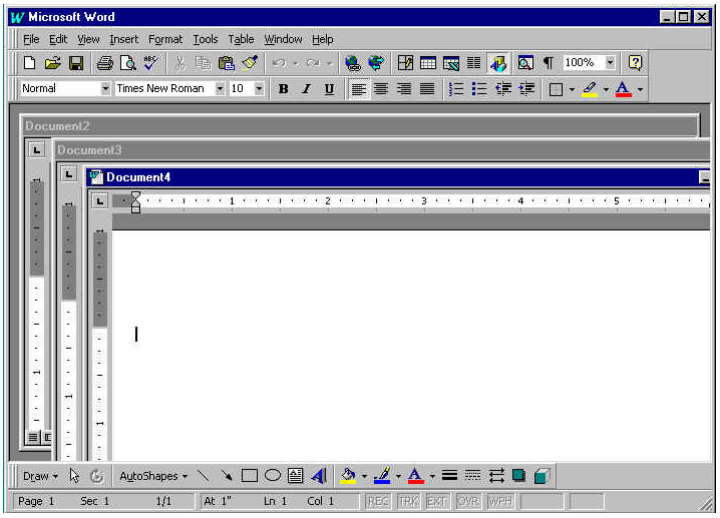
Figure 3-4-6 Single document interface (SDI)



b. Multiple document interface (MDI)

A multiple document interface (MDI) shows more than one document inside one application.

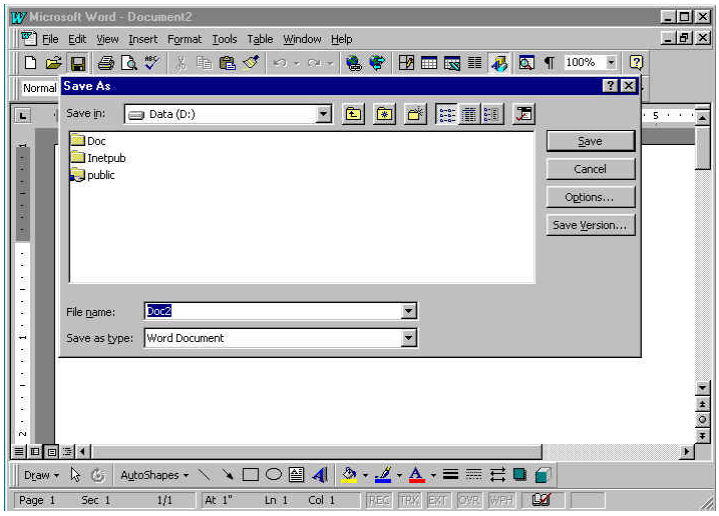
Figure 3-4-7 Multiple document interface (MDI)



④ Dialog box

A dialog box is used to show feedback information to the user and to prompt the user to respond. (See Figure 3-4-8.)

Figure 3-4-8 Dialog box



(3) Procedures and tasks for screen design using GUI

The screen design is done in six steps shown below:

① Standardization

For the layout in the screens and interaction, standard rules are determined.

Figure 3-4-9 Screens standardization

Elements to be standardized	Content
Operability	The operating range is shown so that a user can make a choice and give instructions. It is desirable that the screen have a direct operation feature that allows an immediate response.
Uniformity	Consistency or uniformity must be ensured in the way the same functions are displayed and operated on (icons, for example).
Display	Operating status should be always displayed and necessary information is adequately provided sot that the user can have initiative feelings.
Functions	Online help functions, restore functions, short-cut keys, etc., should be built into the screen with consideration given to the level of the users' proficiency.
Terms, colors	Terms that a user can easily understand should be used. Colors that do not irritate or fatigue a user should be chosen.

② Preparing a general view of GUI screens

A figure showing the layout of the whole screen, and relationships between the elements on the screen is prepared. Basic elements on the screen include main and subsystem menus, and individual operations that are performed.

③ Creating GUI screen flow

The interaction procedures using the screens are designed.

<Important points>

- After the current screen disappears, the next screen opens.
- The screen position is fixed.
- When the "close" button is pressed, the invoked screen is shown.
- To use a screen only to prompt a user to input, a message box is used and the current screen remains open.

- A warning message box appears to get the user's attention. if the "close" button is pressed without updating (saving) the data.

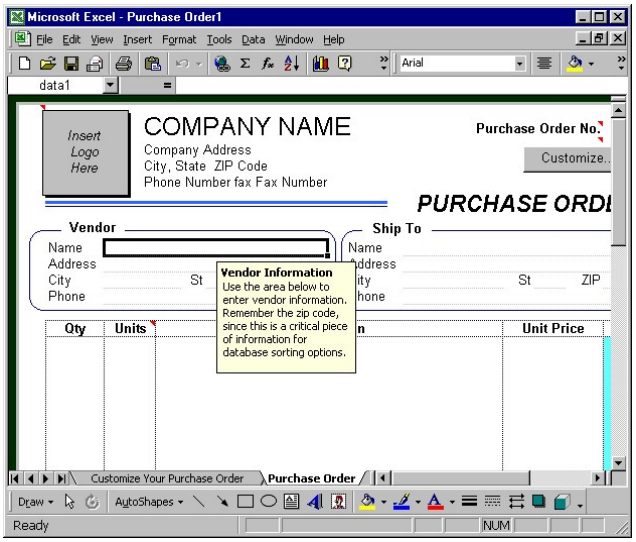
④ Determining the GUI screen input method

The GUI screen input method is determined.

<Important points>

- If the user is a beginner, a selective method such as a pull-down menu is recommended.
- If a user is proficient, a direct input method (inputting text directly) is desirable to save time.
- Items should be arranged in the same order that they are arranged in the data input form, to minimize input mistakes.
- Items should be arranged from top to bottom and from left to right.
- An alarm should be raised when an input mistake occurs while, at the same time, a message box should appear to gain a user's attention.

Figure 3-4-10 Example of the input screen



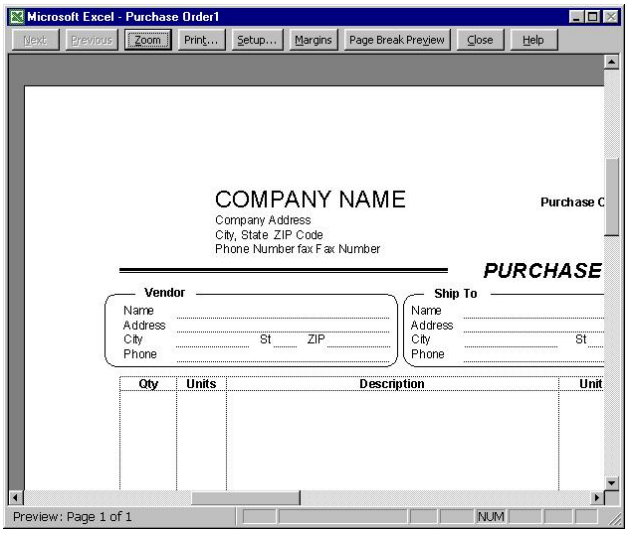
⑤ Determining the GUI screen output method

On the GUI screen, a preview function that allows data to be shown in a printout format should be supported. Also, a search result display function should be supported. (See Figure 3-4-11.)

<Important points>

- The "back" and "next" buttons should be placed, since it may happen that all the data cannot be shown on the screen due to the limited size of the screen.
- A "print" button should be established to allow a user to start printing the output screen.
- Colors should be used and amounts of data displayed on the screen should be limited so that displayed data can be clearly visible.

Figure 3-4-11 Example of the output screen



⑥ Designing the GUI screen layout

One method of designing the screen layout is drawing the screen layout on a special-purpose form. Another method is using a visual language and making a prototype; a prototype is evaluated, modified and improved to complete the screen layout closest to an ideal screen image.

(4) Points to consider when designing the screen

The following points must be considered when designing the screen:

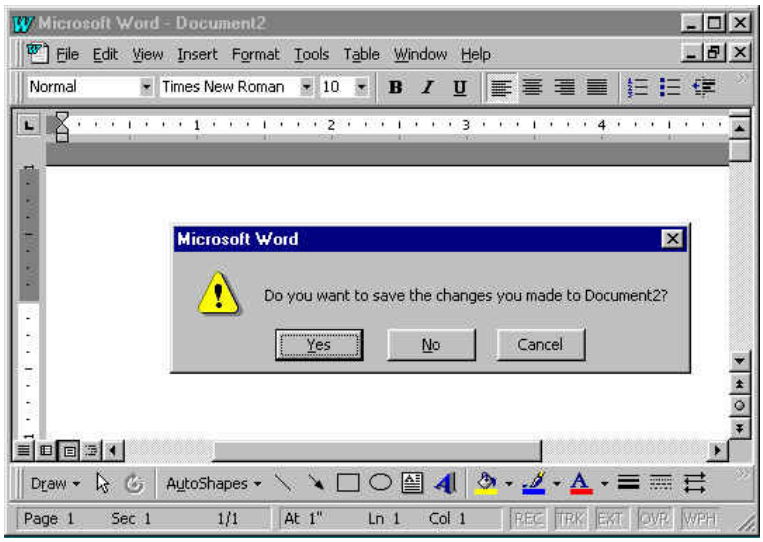
① Standardizing the screen

Because the GUI environment allows a high level of design freedom, completed screen layouts are varied and many, especially if more than one person does the work of design. So many men, so many minds, therefore, fundamental parts of the screen must be standardized as follows:

Example

- Header area and data input positions
  A title and a date must be entered in the header area.
- Button area
  The positions of buttons that specify next functions should be determined.
- Positions and contents of messages (including color messages)
  Basic requirements of a message box should be determined. (Only OK buttons or retry and cancel buttons are needed?)

Figure 3-4-12 Message box



- Flow of process patterns
  A basic flow must be determined; a main menu → a sub-menu → individual processes, for example.
- Use of PF (program function) keys
  Whether PF keys are used or not must be determined.
- Clicking of a mouse
  Standard rules on double clicking and right-button clicking must be established.
- Use of metaphors
  Metaphors should be used to have the user understand easily.

Figure 3-4-13 Metaphors

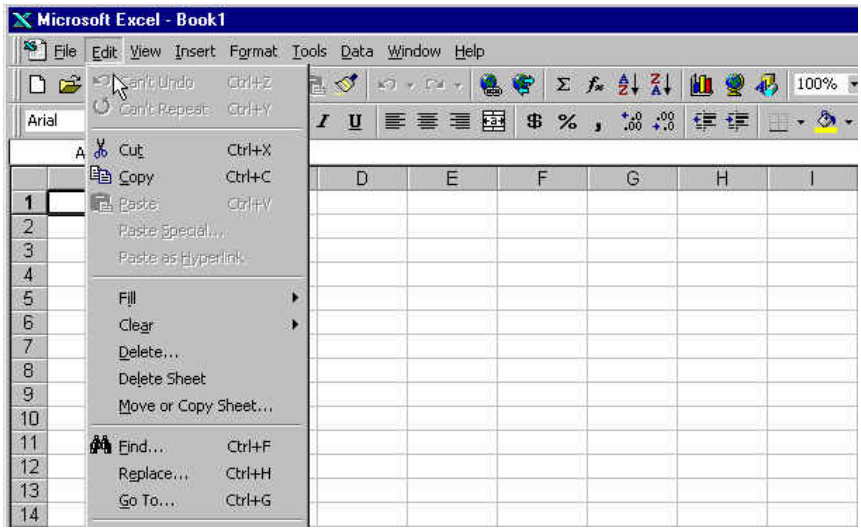
- Standardization of the interaction process
  Interaction using a message box, etc. should be standardized.
- Color
  The number of colors should be kept at five or less. Red and other bright colors should be used to draw attention to a small area, while light blue and other light colors should be used to show a large area.

A format standardized as described above will increase ease of use.

② Short-cut keys

In preparing the overall view of screens, not only step-by-step menu display and select characteristics, but also direct select characteristics (short-cut keys) must be provided according to the level of the users' proficiency. This lessens the input work load, and shortens the time for input, while reducing the overall time and labor. (See Figure 3-4-14.)

Figure 3-4-14 Short-cut keys



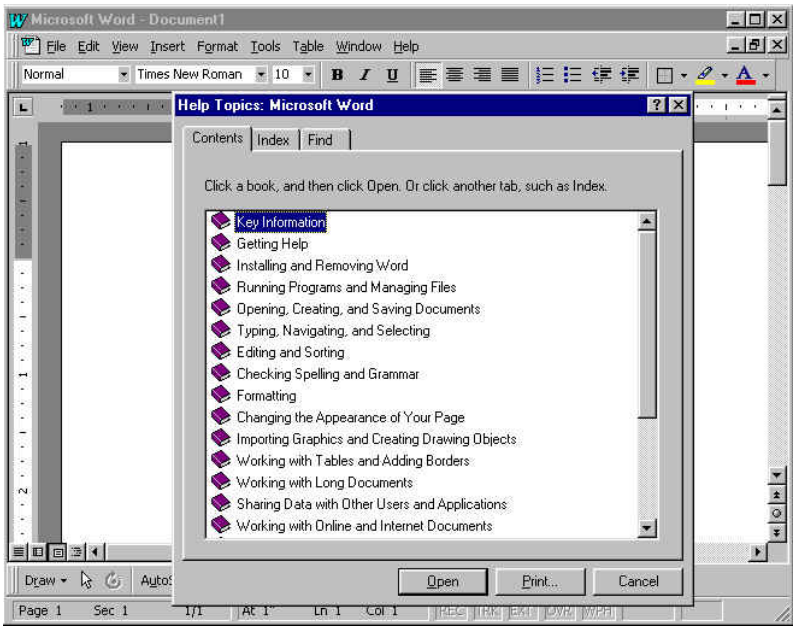
③ Actions to be taken when the screen is locked, or input mistake is made, etc.

The screen must be designed to cope with emergency with a forced termination function that can be used to exit a program if the screen is locked, as well as with a message box that shows a warning message.

④ Screen patterns

Because more than one screen (multi-windows, etc.) must be used to execute a task, the flow of screen patterns must be designed to allow a user to view different screens efficiently. A function for canceling the flow of consecutive patterns (cancel button) as well as an online help function should also be supported.

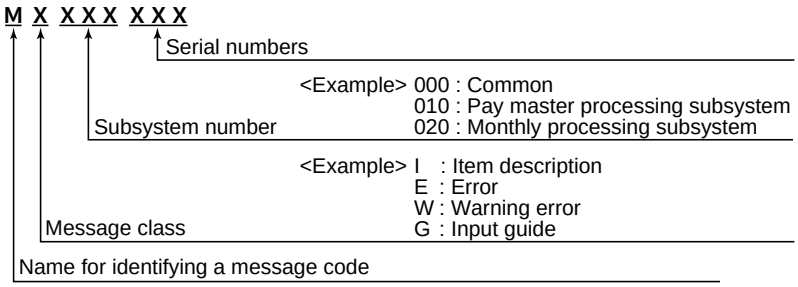
Figure 3-4-15 Online help functions



(5) Designing messages

If a message that reports system conditions is shown on the screen, the system operation efficiency will improve. The screen should be designed to display the internal status, such as a printing error, the pressing of an invalid key, and data errors. (Errors should be shown in red while an alarm is raised at the same time.)
Because codes are easier to handle, codes should be designed (internal codes).
Figure 3-4-16 shows an example of message design.

Figure 3-4-16 Message design



Message code	Message	Description
ME000001	The pressed key is an invalid function key	A wrong function key was pressed
MG000002	After checking input data, press the execution key	To prompt a user to check data
ME010001	The key already exists.	A wrong key is established used?
M I 000002	Set paper on a printer	Paper is not set in printer
ME000003	Data highlighted in reverse video contains errors	Input data contains errors

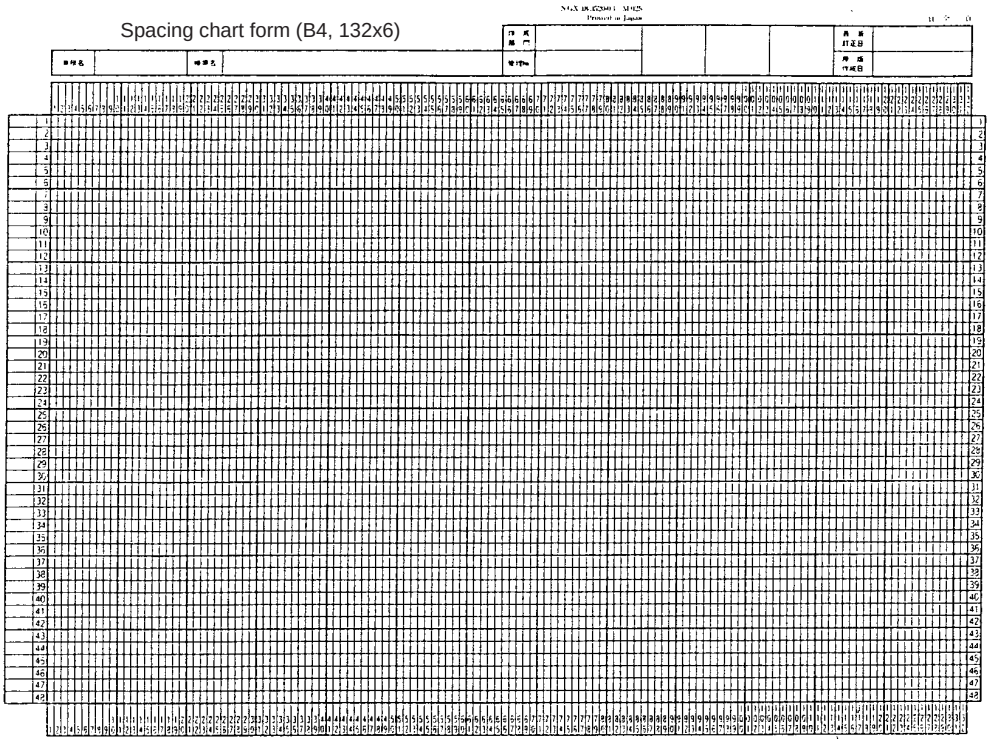
3.4.3 Detailed output data design

A user inputs data, has a computer process it, and outputs it on the display or on paper to check the results of data processing. The data output form must be designed so that it is easy to understand and used for data processing operations.

(1) Report design procedure and tasks

Because the outline of the output report is established in the external design stage, the data output form must be designed using a special-purpose form (spacing chart) based on data provided from external design. (See Figure 3-4-17.)
If a special data output form is to be used, it should be designed in greater detail based on the outline of a form created in the external design stage.

Figure 3-4-17 Spacing chart (example)



The following points should be considered when preparing a design using a spacing chart:
- One line has 132 character positions (in the case of Figure 3-4-17).
- One page has 48 lines (in the case of Figure 3-4-17).
- In principle, one character should be entered in one cell. In the case of a Chinese character or kanji (two-byte Japanese character), one character is entered in two cells.

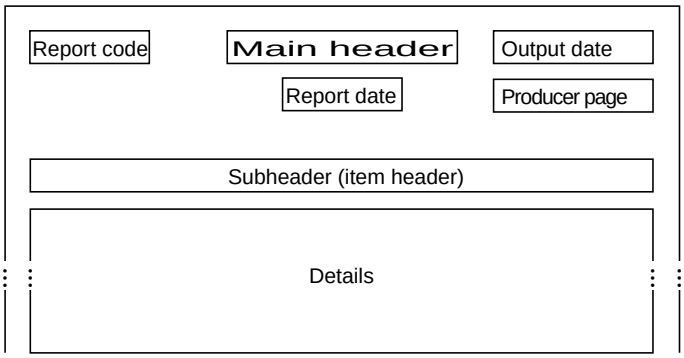
The data output form design is a three-step procedure shown below:

① Determining the title position and the detail lines

The positions of titles (large, medium and small titles) and the specification of detail lines are determined.

- Positions of titles
  Position of the name of the output report, the pagination (usually entered at the upper right), the date when the report is prepared (usually entered at the upper right), and the subtitle for detailed descriptions are determined. The position of the subtitle in particular must be carefully determined since it affects the number of character positions in which each data item must be entered. (See Figure 3-4-18.)

Figure 3-4-18 Determining the position of titles



- Lines where detailed data are entered
  Line-to-line spacing and the position of the last line are determined. In Figure 3-4-17, characters can be printed on 48 lines per page. The layout should be designed so that data entered on these lines is legible and easy to understand for a user.

② Determining the positions of data items and the number of digits

The positions of data items that are printed on each line and the number of digits are determined.

- Positions of data items
  Data items of the same type are printed adjacent to each other. Also, a user can figure out a total easily if data items are arranged from left to right.
- Determining the number of digits
  The number of boxes should be determined with some margin allowed. When a total is printed, a maximum number that can be presumed should be established.
  For numbers, forward zero suppression editing should be carried out to make them clearly visible.

Figure 3-4-19 How the numbers of digits for each data item is shown

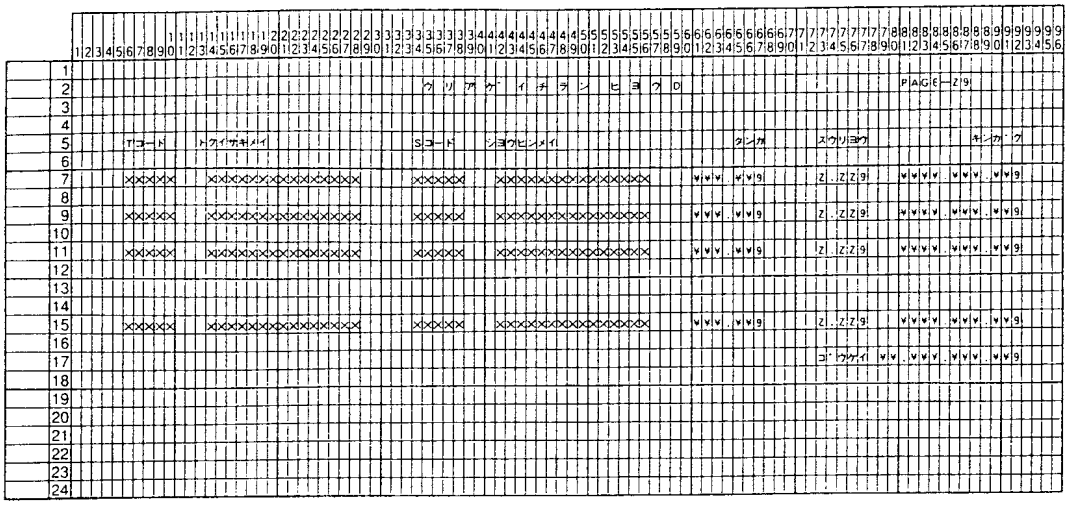
The number of digits can be edited as follows to make the numbers clearly visible.

----,--9	→ -54,000
-$$$,$$$	→ -$54,000
-zzz,zz9	→ - 54,000

③ Designing the report layout

With all considerations described above in mind, data must be entered in a spacing chart to design the data output report. (See Figure 3-4-20.)

Figure 3-4-20 Designing the data output report

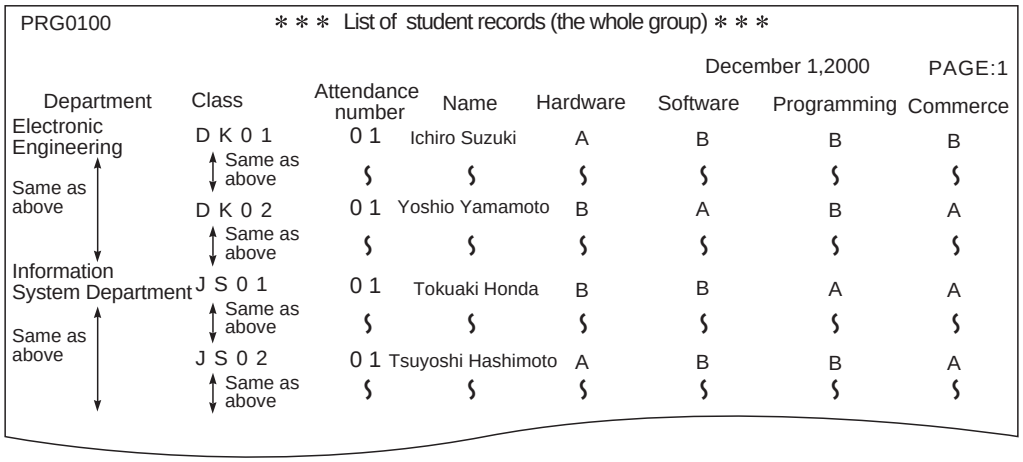


(2) Points to consider when designing the report

The report should be designed with full consideration to usability and "easy to see." Also, the following points should be considered when designing the report:

- Title positions and output items should be standardized as much as possible.
  If title positions and output items are internally standardized, reports are uniform, easy to see and handle.
- An easy-to-see formats should be used.
  For those lines which will contain detailed data, line spaces, character positions, data item editing (zero suppression, etc.), font sizes and so forth should be determined to make the entire report an easy-to-see construction.
- Data should be arranged according to the level of importance from left to right and from top to bottom.
  Data items at the leftmost and topmost positions are emphasized.
- Character strings should be left justified, and numbers should be right justified.
  Character strings should be left justified for output. Unused positions should be left as a blank space. Numbers should be right justified and edited as necessary.
- Formats are set by group.
  Data of the same type should be put together as a group, and arranged in an easy-to-see format. (See Figure 3-4-21.)

Figure 3-4-21 Designing the format of each group



3.5 Creation and reuse of parts

3.5.1 Concept of creation and reuse of parts

General speaking, the challenges of software reuse are ease of 'search', ' registration' and 'modification'. If software parts are considered as means for reuse, these challenges are replaced by 'consistency of parts as units', 'systematization of parts', and 'independence of parts'.
In the object-oriented concept, the unit of a software part is an object. The objects are systematized in a class library, and the independence of object is realized through encapsulation. Therefore, the object-oriented concept can be said to be suitable for reuse of work products.

3.5.2 Use of software packages

(1) Subprogram library

A subprogram library has long been known as the scientific subroutine package (SSP), in the field of scientific and engineering computation. Complex subroutines and functions were not individually developed by the users, but systematized as libraries by manufacturers, universities and research institutes and made available to users.
This concept has been applied to the development of business application programs, and useful subprograms are offered as packages to unspecified users with or without charge. By using the packages, substantial reductions in the man-hours needed to develop software can be achieved. A business application system, however, contains classified matter or proprietary know-how. Therefore, a company organizes the subprograms developed based on internal standards, into a library and allows them to be reused only for internal use.

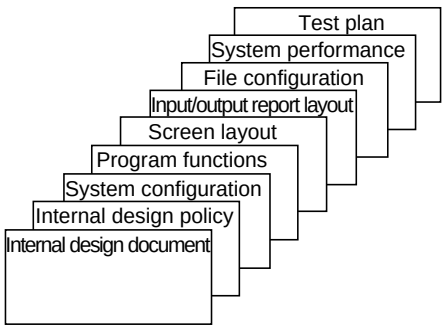
(2) Class library

In the object-oriented development, objects are encapsulated so that changes to an object does not affect surrounding objects. The independence of each object can be maintained. Therefore, objects can be reused as parts more easily than parts in a program developed using procedural language. In the case of the Java language, for example, frequently used objects are collected into a class library. By utilizing a class library, the time and labor needed to develop a program can be saved.

3.6 Creating internal design documents

An internal design document is created based on data provided from internal design. Internal design is prepared based on an external design document from the standpoint of developers. Because an internal design document is reviewed by developers, the use of technical terms in this document is permitted. Also, all details, including subsystems, programs and exceptions, must be described in an internal design document.
Results of specific internal design work are compiled into an internal design document.
Internal design is an important design process since it is responsible for satisfying all user requirements defined by external design. Any design mistake may have a serious impact on subsequent processes. It must be kept in mind that minute portions of a system are designed in the internal design stage, and therefore one should keep in mind that any change in completed design cannot be easily done.
An internal design document consists of the elements shown in Figure 3-6-1.

Figure 3-6-1 Elements in an internal design document



3.6.1 Organization of internal design documents

(1) Internal design policy

In the internal design policy, an overview of the system development policy determined before the start of design work is described.

① Design method

A design method selected to do the work of internal design is described. A structured design method is usually used. How the use of a structured design method was determined is to be described with a mention of other reviewed design methods.

② Documentation techniques

HIPO, DFD, flowchart, bubble chart, etc., are referred to as documentation techniques. Adopted techniques, and how they were adopted is described.

③ Design tasks

④ Others

The procedure of recording changes, details of a design review, and other matters that must be determined before the start of design work are described.

(2) System configuration

In the system configuration, details of a system and subsystems shown below are described.

① System overview (system flow)

In the system overview, the new system flow (new physical DFD) created by external design is described. If a change is made in the new system flow due to the functional partitioning or structuring during internal design, the changed new system flow is described.

② System structure chart (diagram of contents)

A diagram of contents is shown. (See Figure 3-2-13.)

③ Interface between programs

A chart showing input to and output from a program is shown. A data flow diagram (DFD), a flowchart, etc., are used. (See Figure 3-2-14.)

④ Program relationship diagram

The process flow showing the execution sequence of program is usually shown.

⑤ List of programs

A list of programs showing the names of programs used in a system, operation overview, etc., is prepared.

(3) Program functions

In the program functions, details of divided programs are described using the input process output (IPO) diagram.

① General diagram

An overview of subsystems and programs is described.

② Detailed diagram

Details of each individual functional unit (program) are described.

(4) Screen layout

In the screen layout, the following documents are shown:
- General view of the screen
- Screen transition diagram

(5) Input/output report layout

A list of reports used in a system, detailed explanations about each of them and the layouts of both kinds of input source documents and output reports are attached. (See Figures 3-4-2 and 3-4-20.)

(6) File configuration

A list of files used in a system, detailed explanations about each file and the file layout are attached to the file configuration. (See Figure 3-3-4.)
File names, file capacity, library names and positions (disk numbers) are described in a list of files. Also, specifications of tables used in a system are attached.

(7) System performance

The system performance is calculated and evaluated based on the throughput of a computer, the storage capacity and performance of storage devices, the system software itself and processing, system environment, etc. The system performance thus evaluated is described.

<System performance that must be calculated>

- Average amounts of generated data and amounts of data during a peak time
- Time of data processing (for one item or during a specified period)

(8) Test plan

An integration test plan is described.

- Amounts of data

3.6.2 Points to note when creating internal design documents

The following points must be noted when creating internal design documents:

- An internal document must include all functions described in an external design document.
  Although internal design is made from the standpoint of developers, it is based on the external design created for the viewpoint of users. Because the leading actor in the system development is a user, it is necessary to confirm again that all user requirements are included in an internal design document.
- All programs must be clearly described.
  Not only main parts of programs but also exceptional parts, troubleshooting, backup and all other program requirements must be described.
- All files and all inputs and outputs must be clearly described.
  All files, screens and input/output reports used in a system must be clearly described.
- All error handling procedures must be clearly described.
  How errors are handled affects the system reliability. Also, the timing of error handling must be clearly described.
- A documentation standard must be observed, and misleading expressions must be avoided.
  Text in documents must be written in a clear, precise manner.

3.6.3 Design review

A review conducted in the design stage of a software development process is called a design review. It involves design specifications. Because product quality is determined by the design specifications, a design review is very important for the improvement of software quality.

(1) Review method

A design review is mainly conducted on a system design and a program design process. Documents subject to a design review include the following:

- Internal design document
- Program relationship diagram
- Instructions on program functions
- Detailed screen design document
- Detailed report design document
- Detailed file design document
- Test data design document

The following preparations must be made before starting a design review:

- Making a design review plan
  A schedule is adjusted and submitted to an organization in charge of performing design reviews.
- Distributing related materials before starting a design review
  Prepared documents and related materials are distributed to personnel participating in design reviews, so

that they can check them prior to a design review.

- Preparing a review check list
  By referring to a review checklist, the appropriateness of the content of entries, consistency, omissions, conformance with an internal standard, clarity, understandability and other points are checked. A review checklist must include all these points so that an object to be reviewed can be evaluated correctly.

In an actual design review, documents created in the internal design stage are matched with design specifications created in the external design stage. The quality of each specific document is then evaluated, and the level of conformance to the required quality characteristics is reviewed.
The time spent to perform a design review is approximately two hours per review, which may differ, depending on the size of a program. This design review should normally be performed once or twice.

(2) Review system

Personnel shown below play a central role in performing a design review:

- Designers who have the same level of technical skills as those who are directly responsible for creating a program in the internal design stage
- Personnel who are concerned with in a design process

Superiors of above designers and personnel do not need to participate in a design review. This is because a design review aims to evaluate documents, not the capability of people who perform a design review.
In general, a department specializing in design review is established independent of the organization to which the above designers and personnel belong. Therefore, a design review can be conducted from the viewpoint of a third party.

(3) User participation

Detailed screen design or detailed report design, is the work of creating an interface between a system and a user. Therefore, the user organization may participate in a design review as the need arises.

Exercises

Q1 **Which is the most appropriate work to be done during internal design, as part of system development activity?**

a. Code design b. Physical data design c. Structured design of programs
d. Definition of requirements e. Logical data design

Q2 **<u>Choose two</u> tasks to be performed during physical data design in the internal design stage.**

a. Estimating access time and capacity b. Identifying data items c. Analyzing data relationships
d. Creating file specifications e. Determining the layout of records

Q3 **Which is the technique for representing functions and the flow of data by means of symbols showing data flow, processing (functions), data store and external sources (source of generated data and destination)? (This technique is one of structured analysis methods.)**

a. DFD b. ERD c. NS chart d. State transition diagram e. Warnier diagram

Q4 **Which is the appropriate description of the HIPO, one of the structured design methods?**

a. A diagram of contents and a data flow diagram are used.
b. Control information to be passed between processing blocks is described along with an arrow in a diagram of contents.
c. A diagram of contents shows the overall function of a program, and numbers entered in processing blocks show the order of processing.
d. Symbols in a flowchart are used to show what is chosen and what is repeated.
e. Relationships between input/output and process steps can be represented clearly.

Q5 **In the consideration shown below about the screen design in external and internal design stages, which is an <u>inappropriate</u> consideration?**

a. Screen transition should be designed with consideration of not only step-by-step selection using a menu, but also with direct access to a desired screen for users having a high level of proficiency.
b. Each item to which data is inputted on the screen must be enclosed in a rectangle or square brackets, to emphasize that it is a data input field.
c. The screen layout must be designed in such a manner that items to be referred to are arranged from left to right, and from top to bottom.
d. To complete ongoing processing, the screen must be designed to prevent a user from aborting data input, or returning to the previous screen.
e. The screen layout must be standardized; rules on the positions where titles and messages are shown must be established

4 Program Design

Chapter Objectives

Program design is an important process as it is required to do the tasks of programming smoothly.
If a program is developed with clear-cut interfaces and modules designed as black boxes, the parts in the program can be reused or they can be used to create a new program. It may become possible to build a high-quality system by simply combining high-quality modules.
This chapter describes each program design task, as well as the work products in the program design stage.

① Understanding the purpose, important points and tasks regarding program design
② Understanding the contents and meaning of each process in the program design stage
③ Understanding module partitioning and how to evaluate modules

Introduction

Program design is the final process in the whole design stage.
Simultaneously or in parallel with programming, a program design document was prepared in the past. However, with the increased scale of system development, the need for improved productivity and the use of parts, program design is no longer the work supplementary to programming; it is now considered one of the system development processes.
In the internal design stage, a subsystem was divided into functional program units. In the program design stage, the main work is module partitioning. Each functional program unit defined by internal design, is partitioned using the structured design method into modules, the smallest units that can be compiled. By partitioning a program into modules, a program can be made easier to understand and maintain. Furthermore, to make such modules as parts of a newly developed program in the future, logical module partitioning is required.
In the program design stage, the contents of an internal design document must first be understood thoroughly, and then the purpose and procedure must be established to do the tasks of program design efficiently.

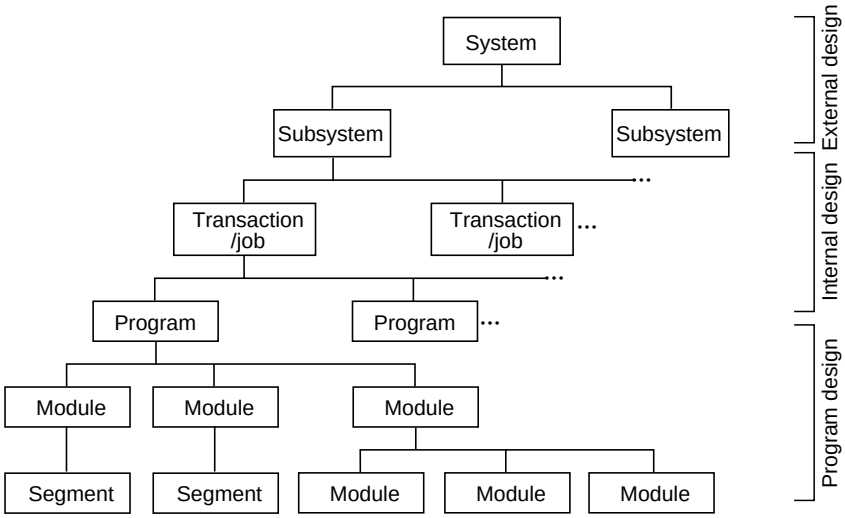
4.1 Purpose and tasks of program design

Program design is the fourth phase in the waterfall model. It is the phase in which programs are designed based on internal design documentation.

4.1.1 Purpose of program design

The purpose of program design is to design the internal structure of a program. In program design, the structured design method is used to partition a program into modules. By partitioning a program into modules and clarifying the relationships among modules, the structure of a program can be made clearer and maintenance can be made easier.
Figure 4-1-1 shows the objects dealt with in system development, including those which are included in the preceding phases.
Objects dealt with in the system development are subdivided depending on the scales of development as follows:
System-subsystems-transactions/jobs-programs-modules-segments (functions)-commands

Figure 4-1-1 Work units in the system development



Modules include the following:

- Compilation units in the high-level languages
- Functional items of a program
- Menu units item
- Source program that has 10 to 300 statements
- Tasks (processes) handled under task (process) management
- Load module units
- Objects used in the object-oriented development
- Graphical user interface (GUI) event units

A module can be defined as one single, quantitatively or logically coherent unit. This definition should be understood only as a criterion.

4.1.2 Program design tasks

The work of program design is done according to the steps shown below:

1. Confirming the contents of an internal design document
2. Partitioning a module
3. Preparing a module specification
4. Preparing a program design document
5. Preparing a test specification
6. Performing a design review

(1) Confirming the contents of an internal design document

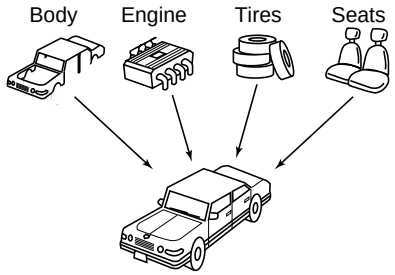
In doing the work of system design, the consistency of all design work, including basic design, external design, internal design and program design, must be maintained. To reflect the contents of internal design as a whole in program design, the points shown below must be examined before a program defined in internal design is partitioned into modules.

- Definition of functions (what to do)
- Input information (what to input)
- Processing (what type of processing to be executed)
- Output (what to output)

(2) Module partitioning

Module partitioning is the work of partitioning program functions into compilation units, using the structured design method. It is the nucleus program design.
Take a car as an example to explain module partitioning.

Figure 4-1-2 Car partitioning



A car is assembled with a body, an engine, tires, seats, etc. If well thought-out design documents and specifications of a car and parts are available, its production is made easy by producing each part before assembling the parts. If a brake fails, its part can be repaired or replaced.
This applies to system development. Unlike the case of car assembly, parts dealt with in system development are intangible, therefore partitioning is done from the physical or logical aspect. Using only a size as a measure for partitioning should be avoided since resulting modules lack uniformity, making it difficult to do the tasks of programming or maintenance. Therefore, as a principle a module must be partitioned into a logical unit, and a physical measure must be used as an auxiliary means.
Module partitioning has the following advantages:

- Module independence can be assured.
  By partitioning the contents of processing defined in the internal design into logically coherent units instead of into physical units, the module independence can be assured.
- Improved processing efficiency
  By minimizing the relationships with other modules, processing efficiency can improve.
- Creation and reuse of parts
  Modules that can be commonly used by programs, or modules that can be reused can be extracted to create new parts, or as parts in a different program.

- Improved maintenance efficiency and reliability
  If it becomes necessary to change system functions, only the modules associated with the system change can be changed or replaced, so that the maintenance efficiency and reliability can improve.

The procedure of module partitioning and details of the structure design will be described later.

(3) Preparing a module specification

The contents of processing to be executed by each module are defined. A module specification must be prepared with attention to details, not to cause missing functions.

(4) Preparing a program design document

Results of the work done according to above steps (1), (2) and (3) are compiled into a program design document, which serves as a guideline for coding work. In the program design document, the following information must be described:

<Contents of program design specifications>

- Program design policy
- Program overview
- Program structure diagram
- Processing details
- Test case
- Item descriptions

(5) Preparing a test specification

A test specification is prepared according to the purpose of each test. Program tests are classified into unit tests and integration tests.

(6) Design review

The purposes of a design review are:

- Validating satisfaction of user requirements
- Verifying consistency to internal design, and readiness for moving on to the work of programming

With these two purposes in mind, a design review must be conducted. Reviews are important at all stages. The results of a design review conducted on the contents of program design have a great influence on the work of programming.

<Important points to consider when conducting a design review>

- The contents of a program design document must be checked.
- The contents of a program design document must be compared with those of an internal design document. Missing functions and defects are pointed out.
- Confirm that there are no missing module-related functions.
- Confirm that module partitioning has been done properly
- The consistency of interfaces between modules must be verified. Also, check to confirm that there is no missing interface.

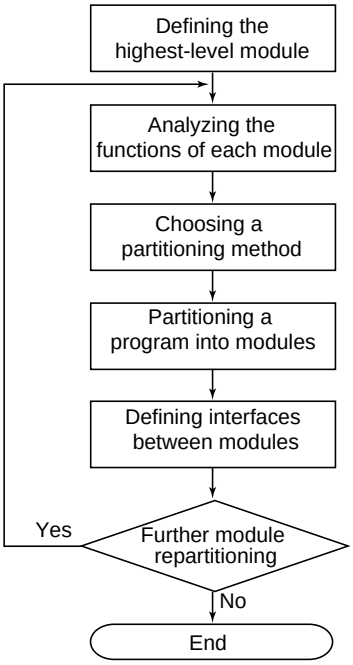
4.2 Structured design of programs

As the scale of systems is growing larger and the functions required by users are becoming increasingly complex, programs used to operate such systems are also becoming complex. As a result, the number of latent bugs increases, and the time and cost spent to correct them also increases.
In the past when hardware was expensive, programs were designed with a complicated logical structure to materialize advanced functions; and to write such programs was a chance for a programmer to show his skills. As system resources are now abundantly available, it is important to create a high-quality, easy-to-understand program.

4.2.1 Structured design procedure

This section describes the structured design tasks of partitioning a program into modules. A key point in the structured design is to partition the program in such a way that it increases the module independence.

Figure 4-2-1 Structured design procedure



Details of each step are as follows:

(1) Definition of the highest-level module

The highest-level module is the one that is first invoked when a program starts. This module performs the following functions:

- Controlling the whole program (each module)
- Setting initial values of data items (such as counters)
- Opening and closing files

There is the case where the highest-level module only controls the whole program, and other modules (low-level modules) set initial values to data items and open/close files.

(2) Analyzing the functions of each module

All essential functions required to run a program are identified. They are further partitioned or combined as necessary so that they can be partitioned to an optimum set of modules.

<Functions of a module>

- Reading files
- Checking errors in inputted data
- Processing (calculating) data
- Outputting data
- Handling errors

(3) Choosing a partitioning method

With the viewpoints mentioned below, the most appropriate module partitioning method must be chosen. Each partitioning method is described in detail in Section 4.2.2.

- Partitioning method designed with a focus on the flow of data
  This method is suitable for an online transaction system.
  <Any of the following methods can be used:>
  - STS partitioning method
  - TR partitioning method
  - Common function partitioning method
- Partitioning method designed with a focus on the data structure
  This method is suitable for the type of batch processing in which files are mainly handled.
  <Either of the following methods can be used:>
  - Jackson method
  - Warnier method

(4) Partitioning a program into modules

Using any of the partitioning methods shown under (3) above, a program is partitioned into modules based on the standards for module partitioning. A module to be invoked is called the subordinate module.

<Partitioning procedure>

1. Defining the highest-level module
2. Defining modules that are invoked by the highest-level module
3. Defining modules that are invoked by the above step 2
4. Modules are broken down step-by-step to low-level modules.

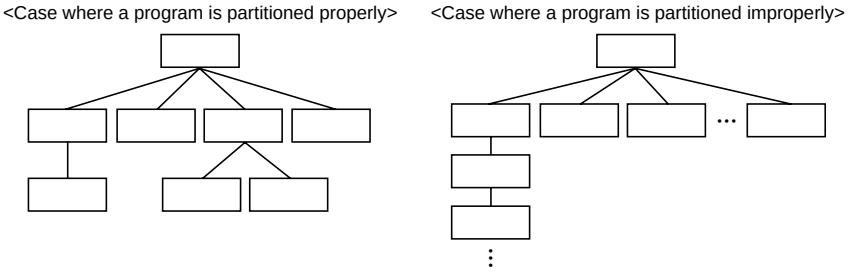
In partitioning, the following guideline should be followed:

<Guideline for module partitioning:>

- If a program consists of 10 to 300 modules, the work of partitioning can be done easily.
- The number of modules to be contained in one level (hierarchy) should be ten or less.
- The depth should be four or fewer levels.

Figure 4-2-2 shows one case in which a program is partitioned properly, and another case in which it is partitioned improperly.

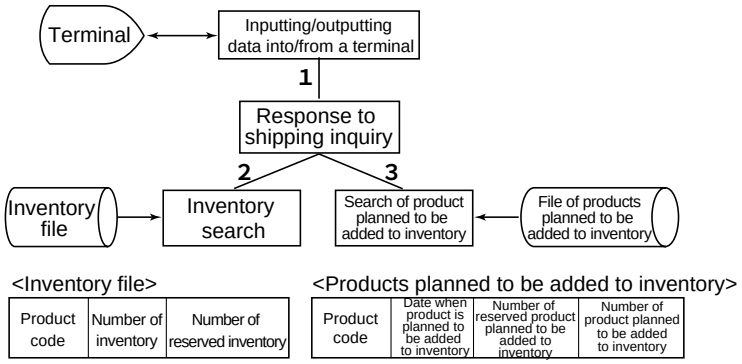
Figure 4-2-2 Two cases of module partitioning



(5) Defining interfaces between modules

We now describe the data, and the conditions required to pass the data between modules. (See Figure 4-2-3.) Defining interfaces between modules is as important as module partitioning. Even after a program is partitioned into modules, the reliability of a program itself can be assured only if each module interface can perform its given function; if A is inputted, B is returned without fail, for example. It is recommended that the number of data items that can be passed via an interface be kept to seven or less.

Figure 4-2-3 Interfaces between modules

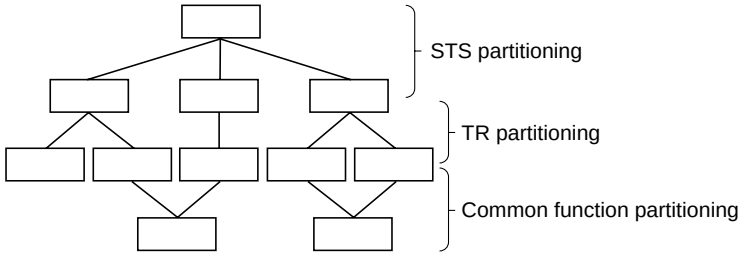


Connection line number	Input from high-level modules	Output to high-level modules
1	Product code, the desired number of products, the desired date of shipment	Status, message A, message B
2	Product code	The number of unreserved inventory
3	Product code	Date when products are planned to be added to inventory, the number of unreserved products planned to be added to inventory

(6) Considering module repartitioning

If a program partitioned into modules, based on the criterion for module partitioning is given an unfavorable evaluation, one of the partitioning methods shown under (3) must be chosen to redo it in a top-down manner. In so doing, multiple methods may be used in combination if necessary.

Figure 4-2-4 Using one partitioning method in combination with others



<How each partitioning method must be used>

1. The STS partitioning method is used for high-level modules.
2. The TR partitioning method is used for medium-level modules.
3. The common function partitioning method is used to integrate detailed, low-level modules.

4.2.2 Typical module partitioning techniques

(1) Partitioning techniques designed with a focus on the flow of data

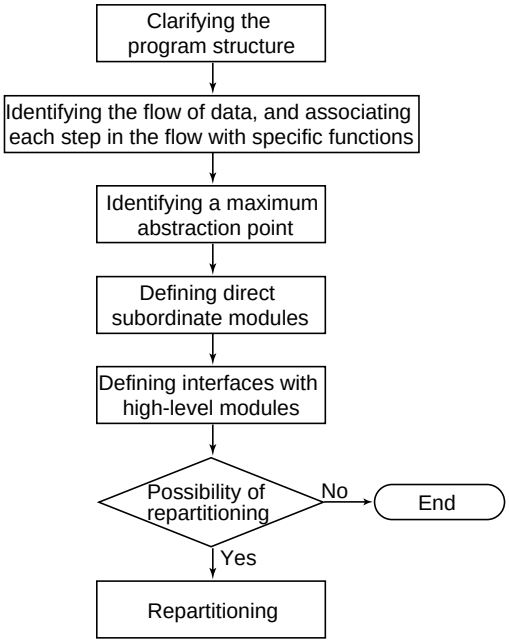
① STA partitioning method

In principle, data is processed through three steps: input, processing and output. Using the STS partitioning method designed with a focus on this flow of data, a program is divided into three parts as follows:

- Source (input processing function)
- Transform (data processing function)
- Sink (output processing function)

This method is suitable for dealing with high-level modules which have all the functions of input, processing and output.

Figure 4-2-5 STS partitioning procedure

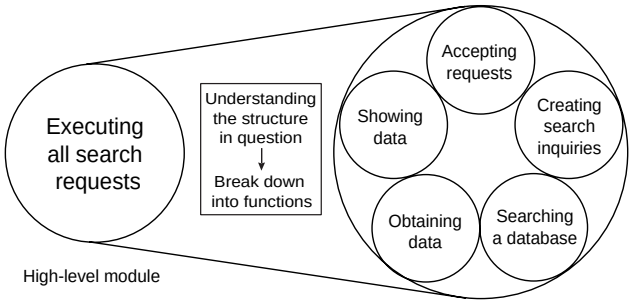


<STS partitioning procedure>

1. The program structure is first clarified.
   The program structure is clarified with main attention given to the functions, as shown in Figure 4-2-6.

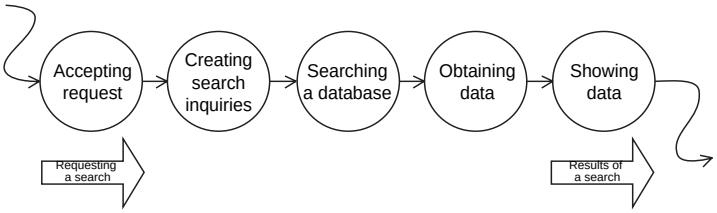
Functions should be grouped so that the number of functions is three to ten.

Figure 4-2-6 Clarifying the program structure



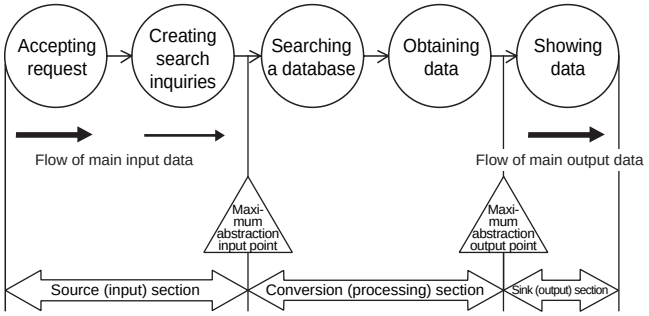
2. The flow of input and output data is identified and each step in the flow is associated with specific functions.
   Each step in the flow of data in a program is associated and linked, using arrows (bubble chart) to form the main flow of input and output data, as shown in Figure 4-2-7.

Figure 4-2-7 Associating each step in the flow of data with specific functions



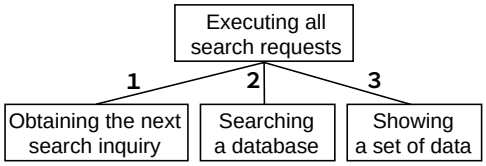
3. Identifying a maximum abstraction point
   Identify an abstraction point where data can no longer be considered input data (maximum abstraction input point), and a point where data begins to take shape as output data (maximum abstraction output point), as shown in Figure 4-2-8.

Figure 4-2-8 Maximum abstraction point



4. Defining direct subordinate modules
   Structure, and associate the modules (including high-level modules) partitioned into input, processing and output, as shown in Figure 4-2-9.

Figure 4-2-9 Structuring modules



5. Defining interfaces with high-level modules
   Define interfaces with high-level modules (interfaces between modules). (See Figure 4-2-10.)

Figure 4-2-10 Defining interfaces between modules

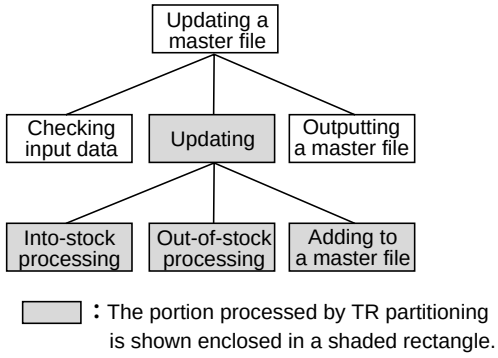
	Input	Output
1	(None)	• Search inquiry • Request statement • Terminal address • Error code
2	• Search inquiry	• Information statement • Error code
3	• Information statement • Request statement • Terminal address	• Error code

6. Checking the possibility of repartitioning
   Check to see if there are modules that must be repartitioned. If any, continue to partition modules.

② TR partitioning method (transaction partitioning method)

If the type of transaction can be determined by the type of data, the TR partitioning method of grouping (modularizing) modules by the type of transaction is used. This method is data-dependent.

Figure 4-2-11 Modules partitioned using the TR partitioning method

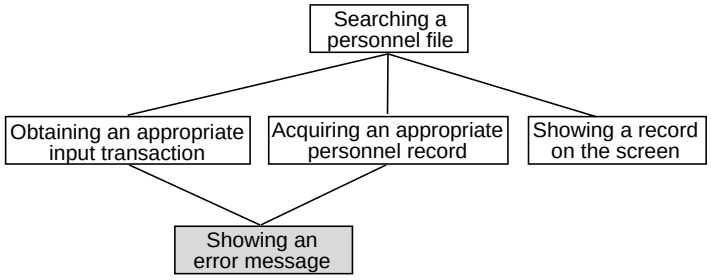


In the example shown in Figure 4-2-11, the types of transaction are partitioned according to the types of input data (into-stock processing, out-of-stock processing, adding to a master file). When it becomes necessary to process a certain type of data, one corresponding transaction of these partitioned transactions is selected.

③ Common function partitioning method

If there are modules that have common functions, this method is used. (See Figure 4-2-12.)

Figure 4-2-12 Common function partitioning method



(2) Data structure-oriented partitioning techniques

Modules are designed by relating the structure of modules to the structures of input and output data.

① Jackson method

With the Jackson method, modules are partitioned by relating the structure of a program to the structures of input and output data.

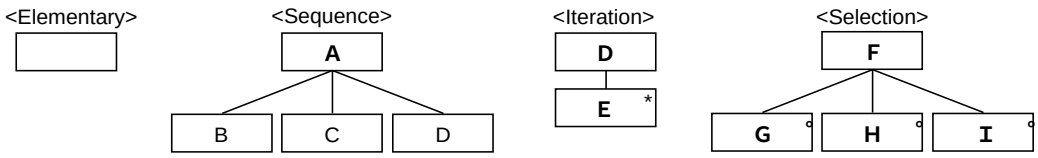
<Partitioning tasks of the Jackson method>

1. Define the structures of input and output data.
2. Find one-to-one relations between constructs in the structures of input and output data. If it cannot be found, an intermediate data structure must be established.
3. Create a program structure based on the structure of output data.
4. Verify the program structure by referring to the structure of input data.

<Constructs of the Jackson method>

In the case of the Jackson method, a data structure as well as a program structure are constructed using three constructs, "sequence," "iteration" and "selection" which are defined according to base elements.

Figure 4-2-13 Constructs of the Jackson method

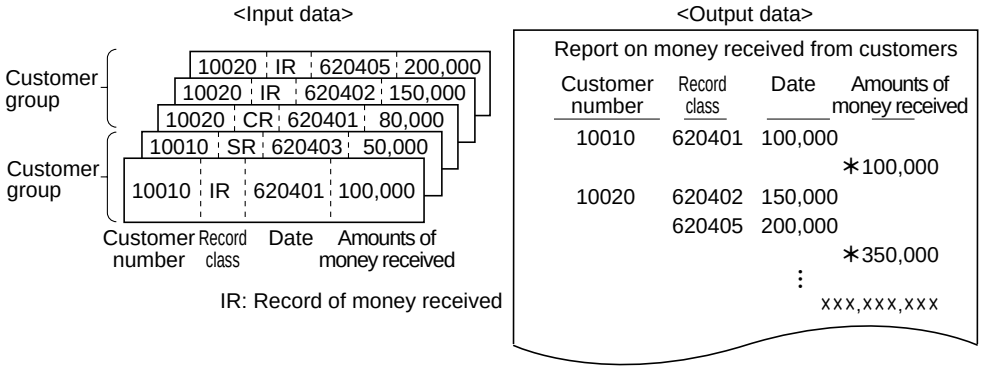


- Elementary: Constructs that cannot be divided any further
  Example: Data items, statements, etc.
- Sequence: Constructs comprising subconstructs; each subconstructs sequentially appears only once.
  Example: Records (multiple data items), sequential processing routines, etc.
- Iteration (*): One construct appears repeatedly.
  Example: Sequential files, PERFORM statement, etc.
- Selection (°): Comprising multiple subconstructs; one of them is selected.
  Example: Debit-credit, EVALUATE statement, etc.

<How to partition modules>

Using the Jackson method, modules are partitioned as shown in Figure 4-2-14.

Figure 4-2-14 Input and output data

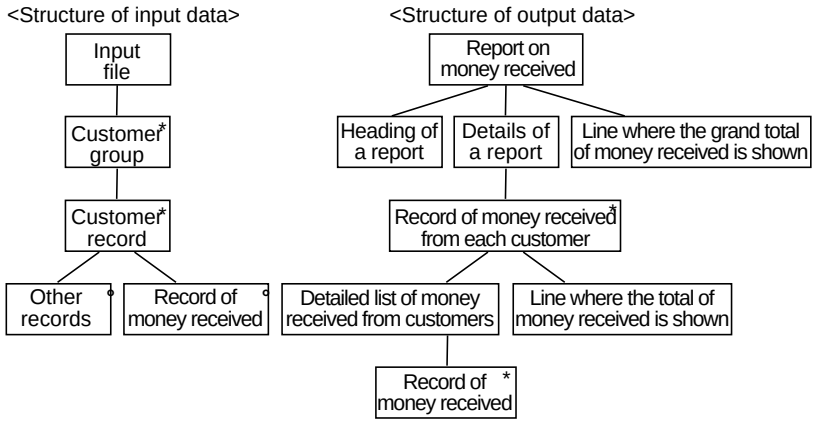


- Define the structures of input and output data.
  Input data consists of the following, as shown in Figure 4-2-14:
  - Input file: The same customer group is repeated.
  - Customer group: The same customer group record is repeated.
  - Type of customer record: Record of money received and other records

Output data consists of the following:
- Output data: A record of money received from each customer is repeated.
- Record of money received: A detailed list of money received from each customer, and a total of money received from customers are repeated.
- A detailed list of money received from customers: A record of money received is repeated.

Therefore, the structures of input and output data can be defined as follows, as shown in Figure 4-2-15.

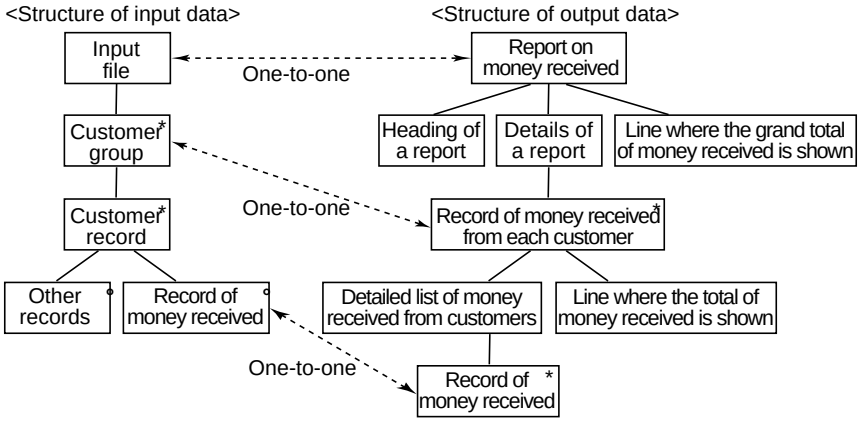
Figure 4-2-15 Structures of input and output data



- Find one-to-one relations between constructs in the structures of input and output data (see Figure 4-2-16). If there are no such relations, an intermediate data structure must be established.

  In the case of Figure 4-2-15, there are three one-to-one relations:
  - Input file and a report on money received
  - Customer group and a record of money received from each customer
  - One record of money received and another record of money received

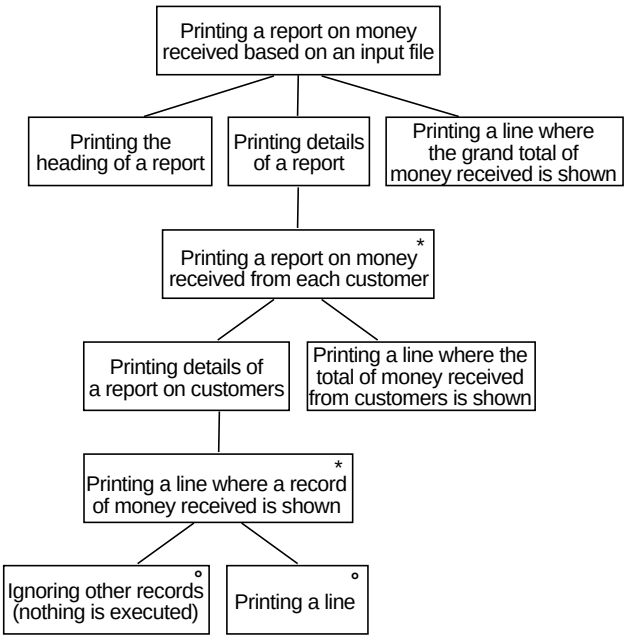
Figure 4-2-16 One-to-one relations between integral elements



- Create a program structure based on the structure of output data.

  Create a program structure based on one-to-one relations between constructs. (See Figure 4-2-17.) In principle, the structure of a program must be related to output data, and input data is used to both verify and correct the structure of a program. (The indication "Ignoring other records" is the result obtained after part of the structure of a program was corrected.)

Figure 4-2-17 Program structure



② Warnier method

The Warnier method is structured module design technique based on the set theory. It is widely used to partition modules for file processing and other business applications.

<Characteristics of the Warnier method>

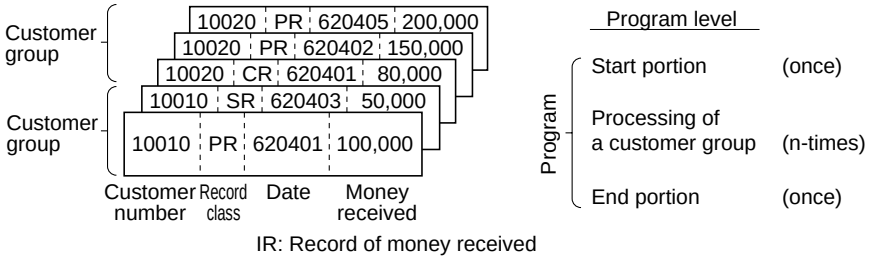
- Data analysis is the basis of this method.
- This method is based on "when, where and how many times." Analysis is made in a top-down manner.
- While the Jackson method is designed to do structuring based mainly on output data, the Warnier method is designed to do structuring based on input data.

The partitioning tasks of the Warnier method is shown below. As an example, that shown in Figure 4-2-14 is used.

<Partitioning tasks of the Warnier method>

1. Find one-to-one relations between the structure of input data and the logical structure of a program. First compare the structure of input data with the logical structure of a program.

Figure 4-2-18 Comparing the structure of input data with the logical structure of a program



IR: Record of money received

The same structure as shown in Figure 4-2-14 is referred to here. The logical structure of a corresponding program consists of the following subsets:

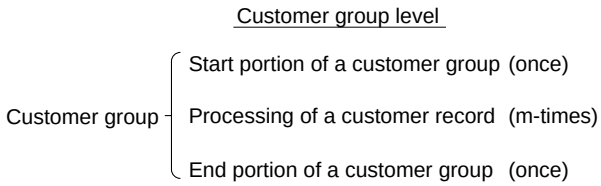
- Start portion
- Customer group portion
- End portion

(N: the number of times of repetition, 0 and 1: either to be selected)

The example shown in Figure 4-2-18 shows that processing of a customer group continues until processing of all files is completed.

2. Break down subsets in a top-down manner.
   A customer group that has been further broken down is shown in Figure 4-2-19.

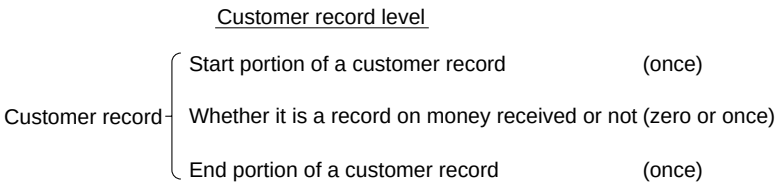
Figure 4-2-19 Breaking down subsets (step 1)



The logical structure of a customer group consists of the following subsets:
- Start portion of a customer group
- Customer record portion
- End portion of a customer group

A customer record portion that has been further broken down is shown in Figure 4-2-20.

Figure 4-2-20 Breaking down subsets (step 2)

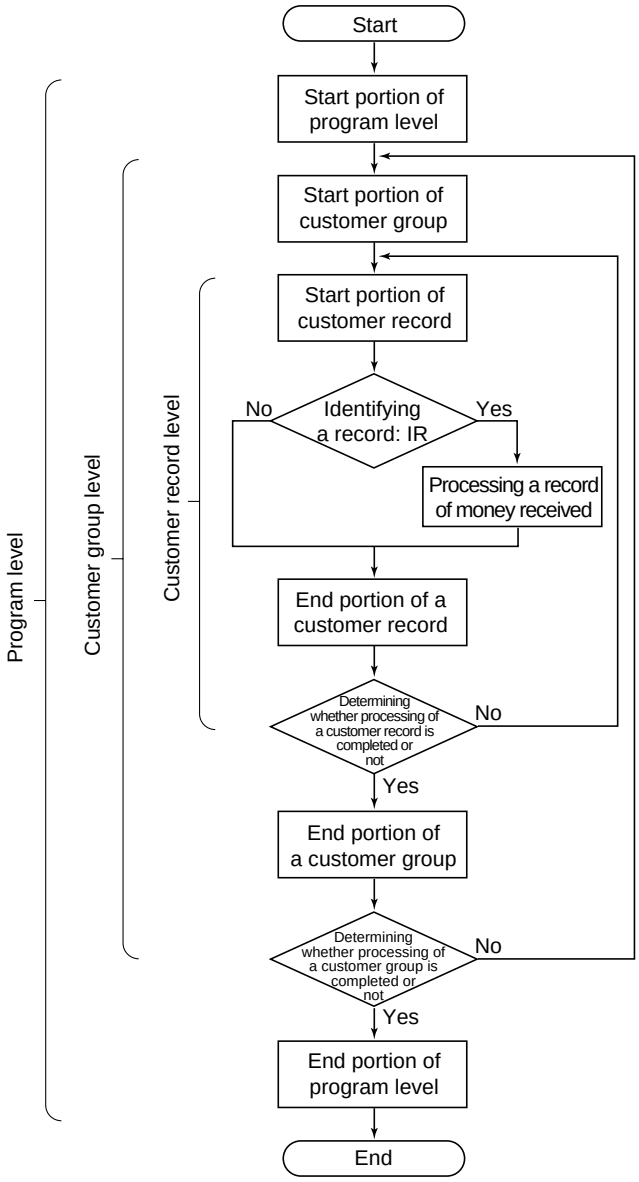


The logical structure of a customer record portion consists of the following subsets:
- Start portion of a customer record
- Whether it is a record of money received or not
- End portion of a customer record

Because the subset "whether it is a record of money received or not" can no longer be broken down, partitioning stops at this level.

3. Drawing a flowchart.
   A flowchart drawn based on Figures 4-2-18, 4-2-19 and 4-2-20 is shown in Figure 4-2-21.

Figure 4-2-21 Flowchart



(3) Using partitioning techniques in combination

The basic partitioning technique used for structured design is the STS method. In the first step of program partitioning, this method is always used, because the STS partitioning allows a programmer to identify the unique structure of a program to work on. It also enables the programmer to understand how data flows at each step of data processing, and how it is transformed while being processed by each function. In the STS partitioning method, functions are divided into subordinate functions.

If the STS partitioning method can't be applied, a different method must be used. STS partitioning does not work well with data-dependent modules - modules for which it is difficult to draw a straight-line program structure. In this case, TR partitioning must be used. In the TR partitioning method, functions are divided into equal subordinate functions.

As modules are being partitioned, the number of modules continues to increase. However, it does not increase infinitely; as modules are partitioned to a certain limit those at a farthest end contain only statements. Although this is an extreme case, partitioning should stop at a certain point. The criteria shown below are used to determine where to stop partitioning:

① Before you start to partition a module, envisage the logical structure of the module. If you can envisage it easily, the module is estimated to contain up to fifty statements. Therefore, you can determine that any further partitioning is unnecessary.
② If you can no longer partition a module into directly subordinate modules that have functional strength (to be explained later), you can determine that any further partitioning is unnecessary.
③ If you hesitate between stopping and continuing partitioning, you should continue partitioning. If you partition too much, you can reorganize modules, however it is much easier to fix an excessively partitioned program than an insufficiently partitioned program.

The module structure changes from a pyramid shape to a mosque shape, as shown in Figure 4-2-22. This is due to the common function partitioning performed in the last step of module partitioning.

Figure 4-2-22 Shapes of the module structure

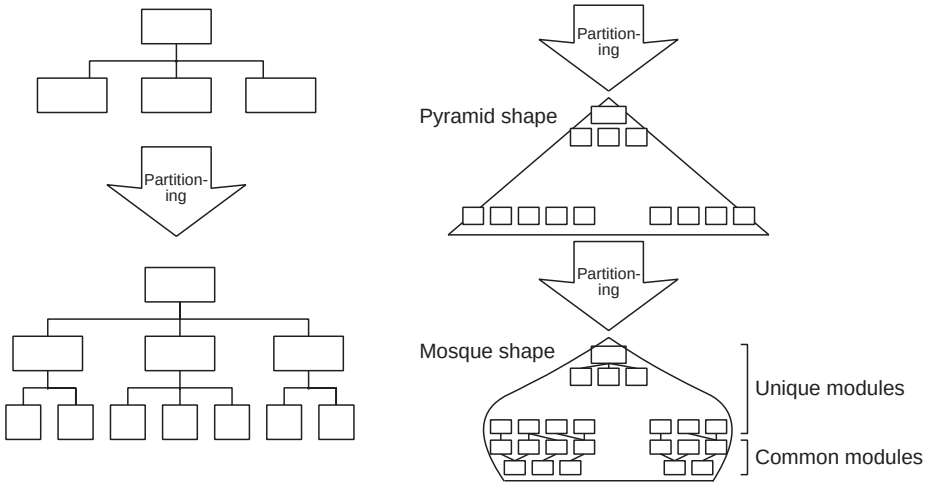
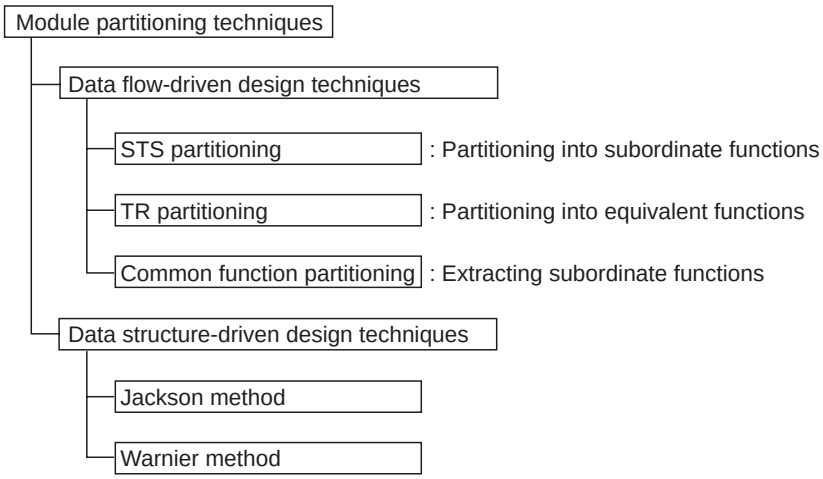


Figure 4-2-23 provides an overview of module partitioning techniques, or program design techniques that have so far been described.

Figure 4-2-23 Module partitioning techniques



4.2.3 Criteria for module partitioning

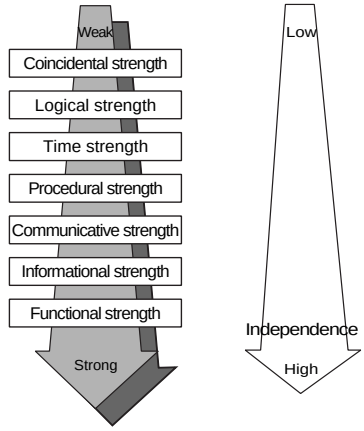
The work of complex program design can be simplified by partitioning a program into modules, as explained in the preceding section. In designing and modifying each part of a program the module independence must be ensured, so that all modules can perform given functions without affecting each other. These two points, simplification of program design and the module independence, are primary goals that must be achieved in the structured design.

There are two measures used to evaluate the module independence: the module strength that shows the strength of relationships between modules, and the module coupling that shows relationships between modules

(1) Module strength

The module strength is one of the guides that we can refer to when defining modules in a design process. Different types of module strength are examined from the aspect of module reusability, error tendency, independence, maintenability, expandability, etc. Specifically, they are examined in the order of unwanted to wanted effects, produced by each type of module strength.

Figure 4-2-24 Module strength



① Coincidental strength

If any of the following conditions is applicable to a module that you are examining, the strength of such a module is called coincidental strength:

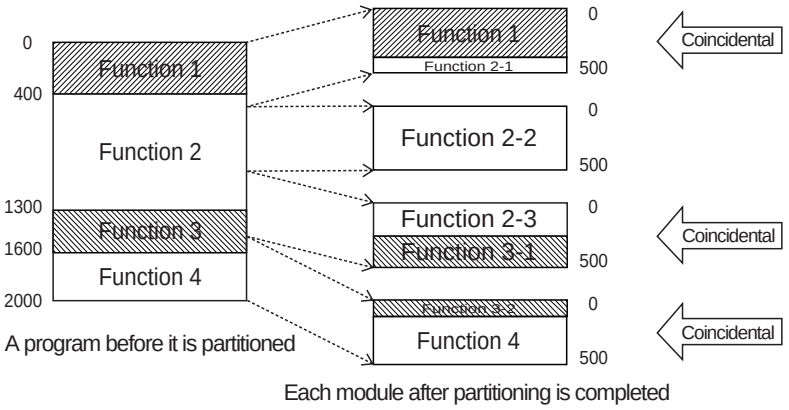
- Functions of a module cannot be defined.
- A module has multiple functions that are irrelevant to each other.

Although a programmer will not create such a module intentionally, it may be created unintentionally under conditions shown below:

- If a module is partitioned into arbitrary portions for overlaying
- If a programmer must meet a very stringent requirement, "the number of statements per module must be 50 to 60," for example.

A module created under such conditions cannot be reused at all; it may cause the maintainability as well as expandability of a program to diminish. It is said that a program partitioned into modules of coincidental strength is worse than a program that is not partitioned at all.
Figure 4-2-25 shows an example of coincidental strength.

Figure 4-2-25 Example of coincidental strength



② Logical strength

A module that has two or more related functions, and executes one of these functions selected by a calling module, is termed a module of logical strength. This type of module performs functions via a single interface. Figure 4-2-26 shows this type of module.

Figure 4-2-26 Module of logical strength (example)

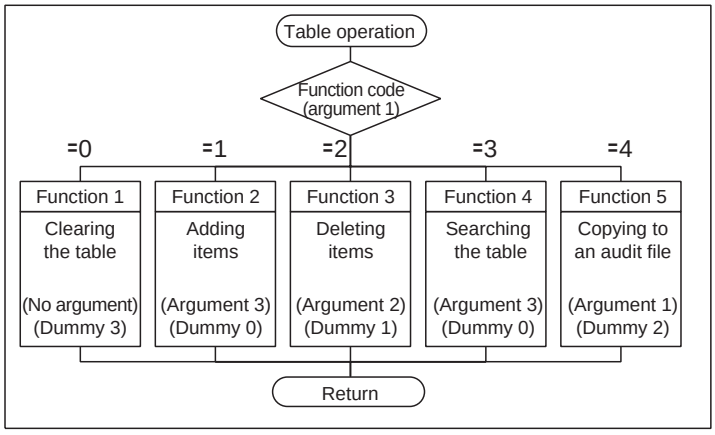
Module name: Prefecture table operation	
Passed arguments: Four arguments	
Argument 1: Function code (one digit)	If the argument is 0: Clearing the prefecture table If it is 1: Adding items to the prefecture table If it is 2: Deleting items from the prefecture table If it is 3: Searching the prefecture table If it is 4: Copying the prefecture table to an audit file
Argument 2: Prefecture name (four digits)	The argument functions as an input argument if functions 1, 2 and 3 are used. The argument functions as a dummy argument if functions 0 and 4 are used.
Argument 3: Prefecture population (eight digits)	The argument functions as an input argument if the function 1 is used. The argument functions as an output argument if the function 3 is used. Others are dummies.
Argument 4: Error flag (one digit)	Output argument (excluding the function 0) 0: No error (normal end) Function 1: 1 - the table is full, 2 - there are overlapping items Functions 2 and 3: 1 - there is no item Function 4: 1 - I/O error, 2 - EOF, 3 - There is no data

A module of logical strength is a little cumbersome to handle for reasons shown below:

- An argument is interpreted differently, depending on the invoked function.
- Although a certain argument is ignored by some functions, it cannot be omitted.
- Even if only one function is used, it is necessary to recognize the other four functions.

Also, if one function used by one module must be modified, all other modules that use this particular module must be modified though they are irrelevant to the function to be modified.
The concept that all processing tasks performed regarding the prefecture table can be incorporated into one module is right and reasonable. To put this concept into practice, it is desirable to use a module of informational strength.
Figure 4-2-27 shows an example of logical strength.

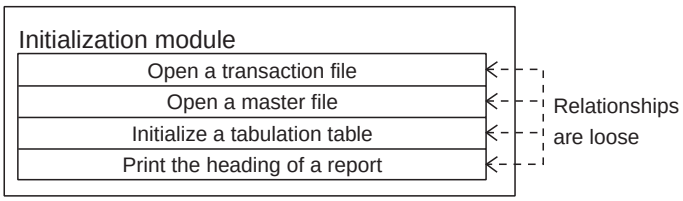
Figure 4-2-27 Logical strength (example)



③ Time strength

A module that executes multiple sequential functions is called a module of time strength. These functions, however, have weak relationships among them. An initialization module or a termination module is this type of module.
Figure 4-2-28 shows a module of time strength.

Figure 4-2-28 Time strength (example)

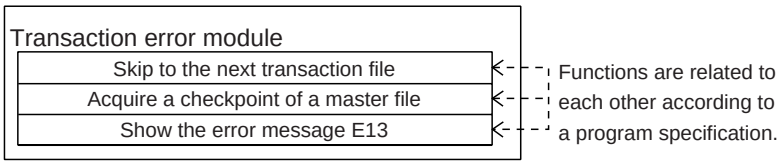


Functions in this type of module have loose relationships among them. However, there is a module performing two or more functions that have strong relationships. If one of the functions is to initialize a tabulation table, as shown in Figure 4-2-28, an operation routine for the tabulation table should exist at a point remote from an initialization module in a program. In its relationship with other modules (implicit modules), it is difficult to distinguish a module of time strength from others.
Therefore, a module of time strength hardly contributes to the independence of a program.

④ Procedural strength

A module of procedural strength is analogous to a module of time length since it performs multiple sequential functions. All sequential relationships among functions, however, are organized into a problem solving procedure. This type of module is called a module of procedural strength.
Figure 4-2-29 shows an example of the module of procedural strength.

Figure 4-2-29 Procedural strength

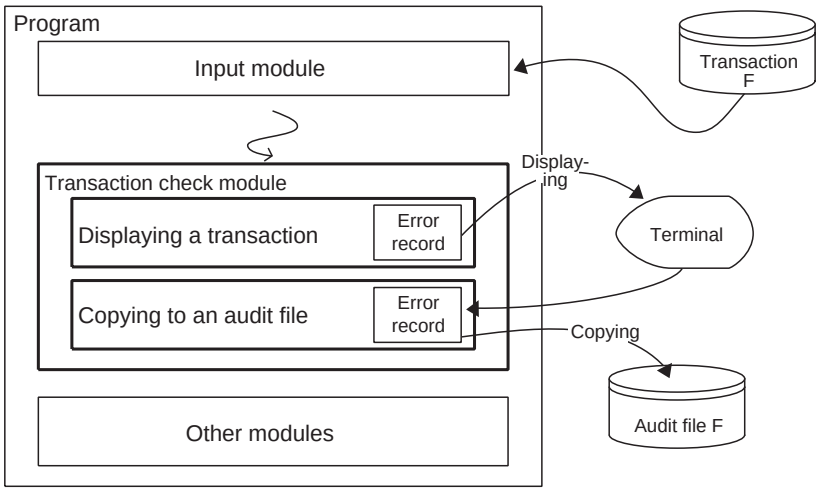


Relationships among functions of a module of procedural strength are slightly stronger than those among functions of a module of time length. If they were not specified, they are hardly discernible. Because the level of procedural strength is around the middle in a 7-level strength hierarchy, there is nothing in particular that we should underrate, or that deserves special mention about this type of module.

⑤ Communicative strength

In a module of communicative strength, sequential relationships among all functions are interpreted as a problem solving procedure. This is the same with a module of procedural strength. A module designed with this characteristic, plus the characteristic that there are data relationships among all the functions, is called a module of communicative strength. That is, a distinctive difference between a module of communicative strength and a module of procedural strength, is that all functions refer to the same data.
If a module specification provides that an inappropriate transaction be displayed on a terminal and copied to an audit file, for example, as shown in Figure 4-2-30, a module that contains a function of displaying on a terminal and a module that contains a function of copying to an audit file are the modules of communicative strength.

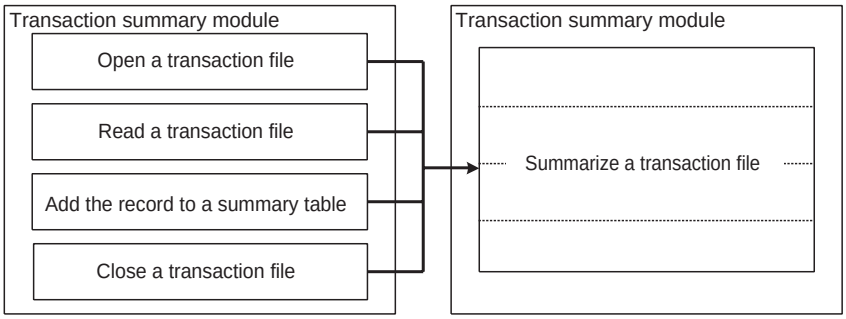
Figure 4-2-30 Communicative strength (example)



⑥ Functional strength

Although informational strength should be taken up if we are to follow the order of strength, we now describe a module of functional strength. Functional strength is the strongest of all types of module strength. A module that executes one unique function is called a module of functional strength. A module that cannot be classified into any of coincidental, logical, time, procedural and communicative strength types can be considered to be a module of functional strength. Even if a module performs two or more functions, it is a module of functional strength if they can be described as a single function.
Figure 4-2-31 shows an example of functional strength.

Figure 4-2-31 Functional strength (example)



⑦ Informational strength

A program designed using only modules of functional strength does not necessarily have the highest level of independence. For example, three modules of functional strength are a module for inserting items into a symbol table, a module for deleting items from a symbol table and a module for searching a symbol table. (See Figure 4-2-32.)
Although these three modules are highly independent of other modules in a program, they are closely related to each other since their functions are dependent on the data structure of a symbol table. Therefore, it can be foreseen that if it becomes necessary to modify one module, the other two modules must be modified as well.

Figure 4-2-32 Module of informational strength (example)

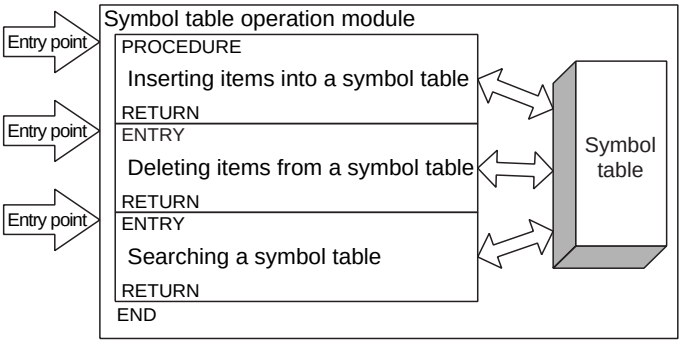
Module name: Prefecture table operation
Entry point: Clearing the prefecture table No argument
Entry point: Adding items to the prefecture table Argument 1: Prefecture name (four digits) (input) Argument 2: Prefecture population (eight digits) (input) Argument 3: Error flag (one digit) (output) 0: No error, 1: The table if full, 2: There are overlapping items.
Entry point: Deleting items from the prefecture table Argument 1: Prefecture name (four digits) (input) Argument 2: Error flag (one digit) (output) 0: No error, 1: There is no matching item in the table
Entry point: Searching the prefecture table Argument 1: Prefecture name (four digits) (input) Argument 2: Prefecture population (eight digits) (input) Argument 3: Error flag (one digit) (output) 0: No error, 1: There is no matching item in the table
Entry point: Copying the prefecture table to an audit file Argument 1: Error flag (one digit) (output) 0: No error, 1: I/O error, 2: EOF, 3: There is no matching data

In the above example, the independence of a program can be increased by replacing three correlated modules of functional strength, with one module. This is called informational strength.
The characteristics of informational strength are:

- There is more than one entry point.
- Each entry point has one unique function.
- All functions are related to one concept, data structure and resource contained in a module.

The purpose of informational strength is to contain a concept, data structure, resource, etc., in one module and achieve information hiding.
Figure 4-2-33 shows an example of informational strength.

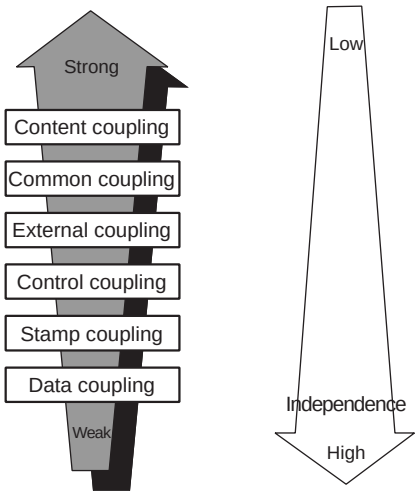
Figure 4-2-33 Informational strength (example)



(2) Module coupling

Module strength concerns the relationships inside a module. On the other hand, module coupling concerns the relationships between modules. (See Figure 4-2-34.) The relationships between modules affect the module independence. It is considered that the looser the relationships between modules, the higher the level of module independence becomes.

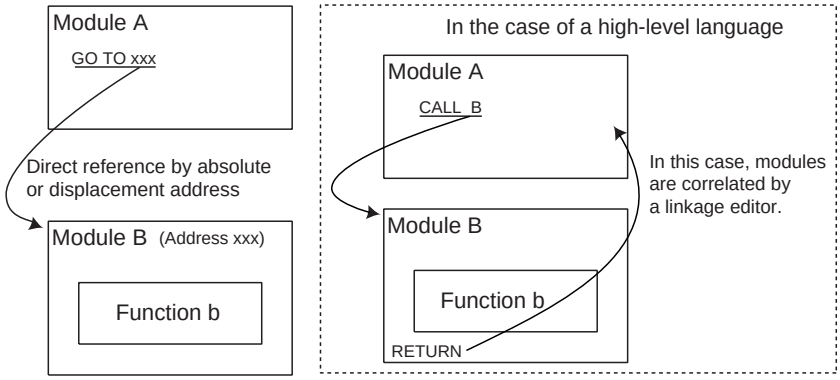
Figure 4-2-34 Module coupling



① Content coupling

If one module directly refers to the content of the other module, or directly branches off to the other module, the relationship between these two modules is called content coupling. (See Figure 4-2-35.) If one module has been modified, it is necessary to modify the other module. Modules of content coupling can be created using assembler languages. In most of high-level languages, however, they can not be created.

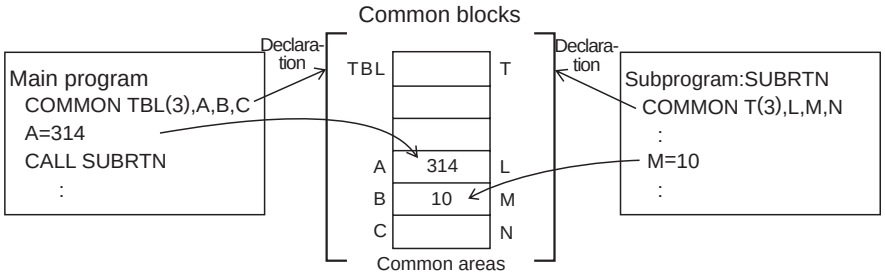
Figure 4-2-35 Content coupling (example)



② Common coupling

The relationships between modules that refer to a global data structure (global variable) are called common coupling. For example, there is a concept called a common block in FORTRAN. Variables are declared by COMMON statements in order to couple memory areas used by programs. (See Figure 4-2-36.) Common coupling derives its name from the fact that memory areas are made common to allow modules to share them.

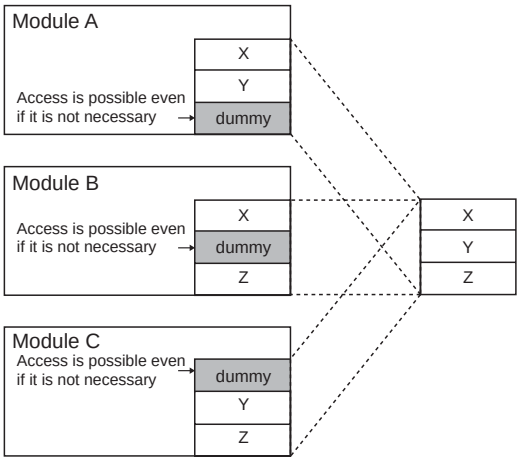
Figure 4-2-36 Common coupling (example 1)



Problems of common coupling are summarized as follows:

- Global data impairs the readability and understandability of a program.
- A global data structure may cause vaguely related modules to become dependent on each other.
- Modules that refer to global data are difficult to use for diverse purposes. (In performing parallel processing or multitask processing, the consistency of a common data area must be taken into consideration.)
- Because modules are coupled via variable names, it is difficult to reuse them for a new program.
- If global data is structured-type data, it is even more difficult to reuse them. (Creating a dummy structure)
- If a global data structure is used, more data than necessary is disclosed to other modules. (See Figure 4-2-37.)

Figure 4-2-37 Common coupling (example 2)



- Data access within a program cannot be managed properly.

If one part in a global data structure is modified, all modules that refer to that modified part of data must be recompiled.

③ External coupling

If there is a group of modules that are of neither content nor common coupling and refer to a global data structure, the relationships between such modules are called external coupling. Although common coupling concerns a set of global data that are different in format and meaning, external coupling concerns a set of data that are of the same type.

Data of the same type mean:

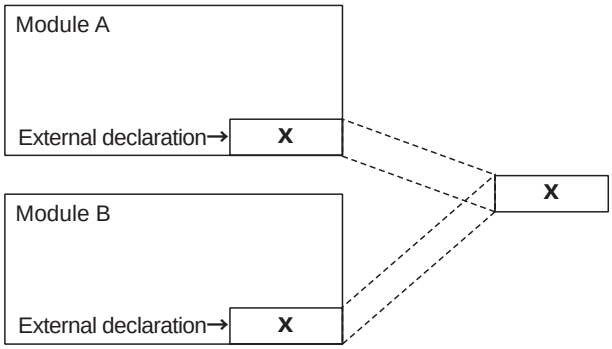
- One type of data that does not contain other types of data; numerical data, character data, etc., for example.
- A table or a list that has only one item
- An array that has elements of the same meaning

In our attempt to find solutions to the problems associated with common coupling, external coupling enables us to improve the conditions concerning only the following problems:

- Undesirable dependence between modules
- Creation of a dummy structure
- Over-disclosure of data

Even if some improvements can be made as to the above problems, the conditions are still unsatisfactory.

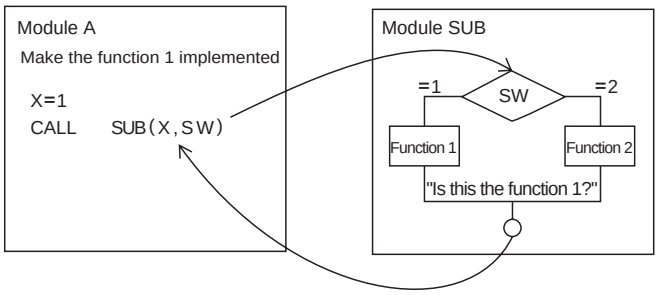
Figure 4-2-38 External coupling



④ Control coupling

If there are two modules that are neither content nor common nor external coupling and if one module passes control elements to the other module, the relationships between these two modules are called control coupling. When one module passes a function code or a control switch to the other module of logical strength, the relationships between them are control coupling. Because a module passing control data must know the logical structure of the other module, the level of independence is not very high. The relationships of control coupling are restricted to the case where one module as a sender of data, passes arguments as a means for controlling the functions and logic of the other module as a receiver. If one module passes an argument for which no specific purpose is designated, to the other module the relationships between these two modules cannot be termed control coupling, even if the other module as a receiver uses the argument (control data).

Figure 4-2-39 Control coupling (example)

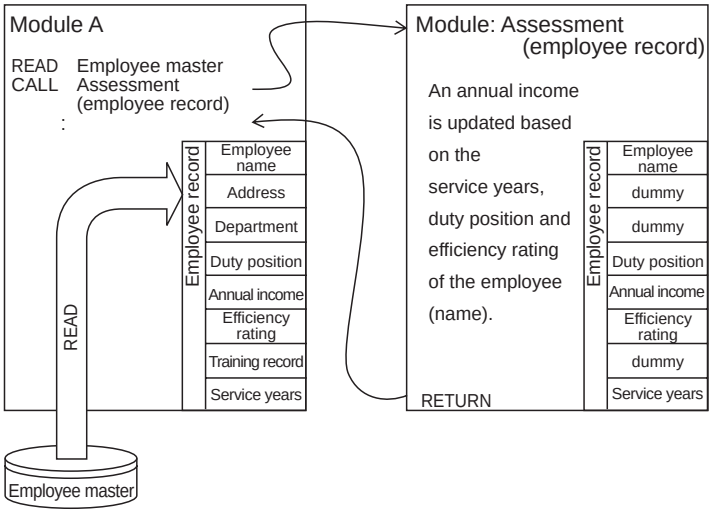


⑤ Stamp coupling

If there are two modules that are of neither content nor common, external nor control coupling and if they refer to the same non-global data structure, the relationships between these two modules are called stamp coupling. Although stamp coupling is similar to common coupling, the difference is that modules of stamp coupling refer to a non-global data structure, not a global data structure.

The module "assessment" shown in Figure 4-2-40 is used to update "annual income," using an employee record imported into module A as an argument, as well as data on "duty position," "efficiency rating" and "service years" of one employee.

Figure 4-2-40 Stamp coupling (example)



For a module to perform the given "assessment" function, at least four different data items are needed. Because the whole of an employee record functions as an argument, other unnecessary items in addition to four items are passed. Therefore, if there is any change made in items other than these four, the module responsible for performing the "assessment" function must be recompiled. Furthermore, as unnecessary items can be referred to, there is the possibility that they are mistakenly updated. The relationships between the module "assessment" and all other modules that refer to specific items in an employee record, are called stamp coupling.
Stamp coupling enables us to solve the following problems of common coupling:

- Lack of readability and understandability
- Not suited to be used for diverse purposes
- Name dependence

Although the workload needed to control data access can be reduced, it cannot be completely eliminated. Also, the following problems still exist:

- Undesirable dependence between modules
- Creation of a dummy structure
- Over-disclosure of data

⑥ Data coupling

In order to improve the conditions concerning the module "assessment" problem, the module responsible for "assessment" must be designed as a module that does not know anything about an employee record. The module required three items, "duty position," "efficiency rating" and "service years." It outputs one item "annual income" based on data contained in these three items. If we can define these four items as arguments, we can create a module that knows nothing about an employee record.

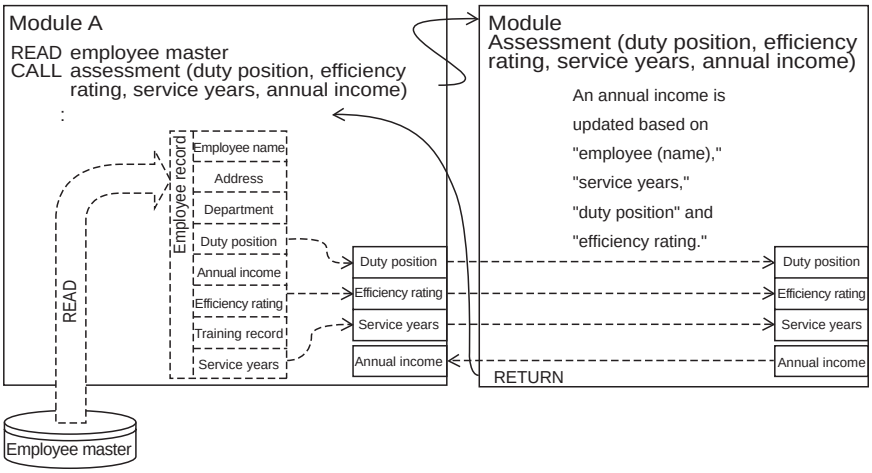
CALL assessment (employee record)
↓
CALL assessment (duty position, efficiency rating, service years, annual income)

Assuming that the above module is newly created and the following requirements are met, the relationships between modules are called data coupling:

- The status of module coupling is none of content, common, external, control or stamp coupling.
- Two modules are directly correlated.
- All interfaces between modules are of the same data type.

Data coupling is at the weakest level of coupling. (See Figure 4-2-41.)

Figure 4-2-41 Data coupling



(3) Criteria for partition sizes

A proper size for a module is generally 40 to 50 statements in terms of the number of executable instructions in a high-level language. If the size is much smaller than this, the person who reads a program must frequently jump from one module to another and his flow of thought is interrupted.
If the size is much larger, that is, if it is over 100 statements, clearly defined interfaces are not available and there are too many things that have to be considered.
The criteria for partition sizes are as follows:

- For a small-size, well-structured program (the number of executable instructions is 200 to 300 statements): An average of 30 statements
- For a medium-size program (the number of executable instructions is 2,000 to 3,000): An average of 40 to 50 statements
- For a large-size program (the number of executable instructions is 10,000 or more): An average of 100 to 150 statements
- For a module of informational strength (this is an exceptional case): An average of 40 to 50 statements for one entry point

Above figures are average and should be used as a mere guideline. If modules are partitioned or combined in too strict obedience to the figures, the level of module independence will decrease. Caution must be exercised in this regard.

(4) Creation and reuse of parts

In order to regard modules as parts and reuse them, we should be more conscious of the module independence. A module with vaguely defined functions, one that is too much dependent on other module, one that is too much influenced by other modules or one that is dependent on a particular data structure cannot be reused.
The goal of structured design is to create only modules of functional and informational strength. If only such modules can be created, the module independence can be assured, the error rate can be decreased, the expandability can be enhanced, and the probability of reuse can be increased.

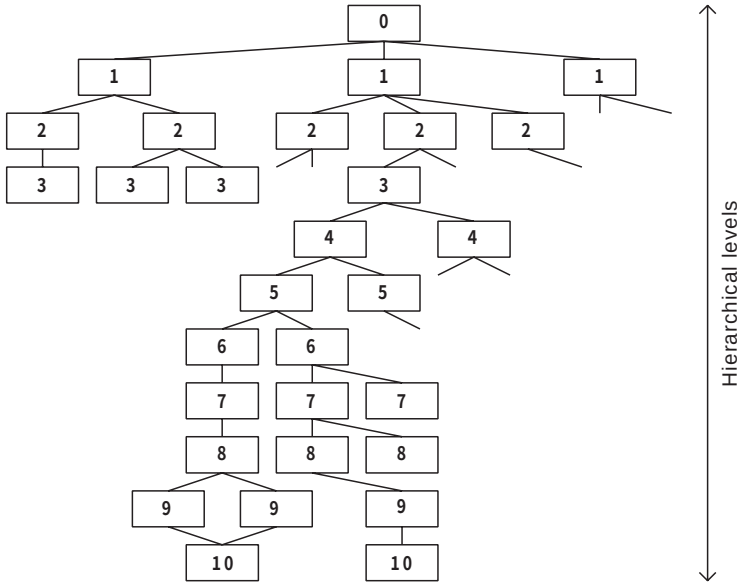
4.2.4 Program partitioning

(1) Important points to consider concerning the number of partitions and the hierarchy depth

① Hierarchy depth

If a hierarchy is too deep, the logical structure becomes difficult to understand. Although the size of a program is a factor to consider, the hierarchy depth must be limited to ten or less levels.

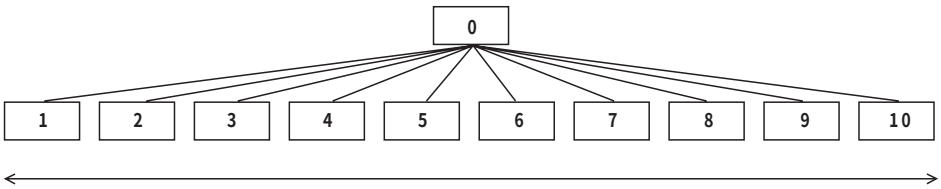
Figure 4-2-42 Hierarchy depth



② Fan out of a hierarchical structure

If a hierarchy of modules expands too much, the program flow becomes difficult to understand. The number of subordinate modules that a certain module can directly invoke should be limited to ten or less.

Figure 4-2-43 Fan out of a hierarchical structure



4.3 Creating module and test specifications

4.3.1 Creating module specifications

After a program is partitioned into modules, details of the processing that is executed by each module are determined (module logical design).
The coding work must be done with attention to minute details so that there is no missing function.

(1) Important points to consider when creating module specifications

- Creating a simple, easy-to-understand structure
- Taking into account the ease of debugging and maintenance
- Observing the standard for creating a program (coding standard)

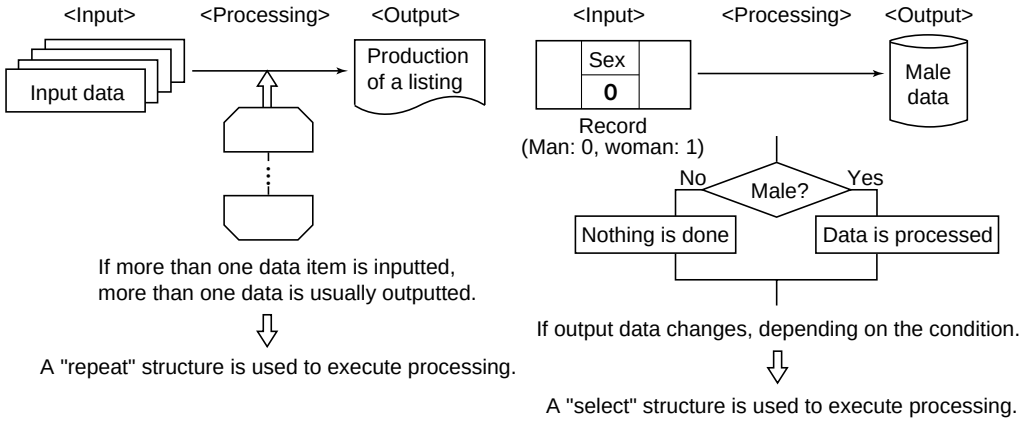
(2) Procedure for creating module specifications

Module specifications are created according to the procedure shown below.

① Analyzing a data structure

Because a data structure has a close relationship with the logical structure of a module, structures of input and output data must be analyzed, and relations between input and output data as well as the items generated through processing must be identified and listed.

Figure 4-3-1 Input, processing and output



If there is more than one data item, data processing is executed using a repeat structure. If output data changes, depending on the condition, data processing is executed using a select structure.

② Layering a data structure

With the meanings of each data item in mind, data is organized and combined, and a data structure is created. Relations between input and output data are clarified, and input data are associated with output data.

③ Building logic (algorithm)

Converting input data into output data is a task to perform. Output conditions are defined in consideration of the structure of output data. Defined output conditions are used to determine processing details and to build an algorithm.
Logic is usually documented using pseudocodes.

Figure 4-3-2 Pseudo-codes

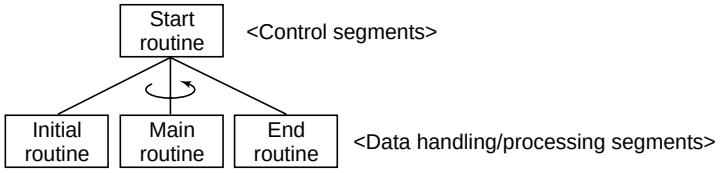
```
Reading the first record
DO   WHILE   a record resides
     IF   class = into-stock   THEN
          Into-stock processing
     ELSE
       IF   class = out-of-stock   THEN
            Out-of-stock processing
       ELSE
            Error processing
       ENDIF
     ENDIF
     Reading a record
ENDDO
```

(3) Segments and statements

A module consists of logically coherent segments. The size of segments is 10 to 50 statements (sentences, instructions).
Segments in a module are classified into two types shown below:

- Control segment (high-level segment) that controls all segments
- Data handling/processing segment (low-level segment) that executes a data processing task

Figure 4-3-3 Control segments and data handling/processing segments



If the processing conditions and processing details are the same, they should be combined into a segment.

4.3.2 Creating test specifications

A unit test is the test conducted in the program design stage. The purpose of this test is to verify the logical structures of coded modules and the interfaces between modules.

(1) Types of tests

The first unit test is conducted to verify that the logical structures of modules are created in accordance with a module specification. A technique used during this test is a white box test for removing bugs and preventing them from being passed to subsequent tests.
The next module integration test is conducted to check interfaces between modules. Specifically, modules are coupled to confirm that input and output parameters defined in a module specification, can be passed properly. A technique used during this test is a black box test. The module coupling test is divided into a top-down test conducted by successively coupling high- to low-level modules, and a bottom-up test conducted by successively coupling low- to high-level modules. Each method has advantages and disadvantages. Either should be selected in consideration of the size of a program and available resources.

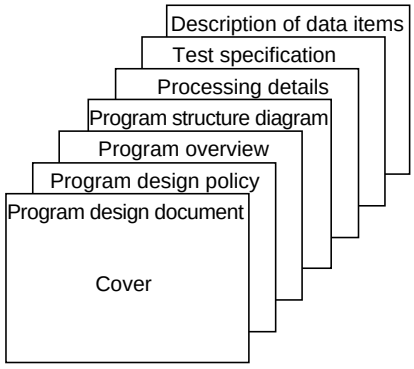
(2) Important points to consider when designing a test case

In general, a test case means not only test data but also a test plan and other documents. Before conducting tests at each level, test data should be prepared with consideration of test techniques to be adopted. In addition to normal data, error data must also be prepared. Output results that can be foreseen based on test data must also be prepared. If tests are conducted with attention to details, the quality of a program can improve, but more work-load will be needed. Because productivity is inversely proportional to coverage, it is necessary to design test data with consideration given to keeping them balanced.

4.4 Creating program design documents

Program design documents are created based on various data and materials prepared in the program design stage. In the program design stage, a program built by internal design is partitioned into modules, and the logical structures of modules are determined. In doing the work of program design therefore, modules must not be partitioned or their logical structures must not be designed in such a manner that may impair the functions of a program, which is designed in the internal stage to meet specified requirements.
In the next programming stage, the coding work is done in accordance with a program design document. There is the case where a contractor is commissioned to do the work of program design and subsequent processes. In this sense, a program design document is very important, in that it greatly affects program design and subsequent processes. Therefore, a design review must be conducted in a thoroughgoing manner, based on a program design document in order to extract and correct all the defects.
Figure 4-4-1 shows the items that comprise a program design document.

Figure 4-4-1 Program design document



4.4.1 Creating program design documents and the contents

(1) Program design policy

A program is designed based on the following program design policies:

- Design policy
  Prior to the start of program design, the adopted design technique is described. Structured design is usually used. If a different design technique is adopted, the reasons for adopting it must be described.
- Documentation techniques
  HIPO, DFD, flowchart, bubble chart, etc., are referred to as documentation techniques. Adopted techniques and how they were adopted must be described.
- Design tasks
  The program design tasks are described.
- Others
  Change records, details of design reviews and other matters that need to be determined prior to the start of program design must be described.

(2) Program overview

A diagram that shows the structure of a program (including types of input and output data) is attached to the program overview. (See Figure 4-4-2.)

Figure 4-4-2 Program overview

Program overview	Date	／ ／	Originator		Approved	
System name	Pay calculation system		Subsystem	Monthly processing		
Program name	Monthly processing		Program ID	GET20030		

Language		Main or sub		Estimated lines of code	

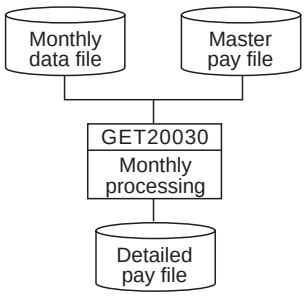
Function

Sorted work data (monthly data file) is checked against a master pay file.

After the total pay, total deduction and a balance to be paid are calculated,

a detailed pay file is created.

Input and output summary



Content of a design document

- ☐ Change record
- ☐ Supplementary explanations about functions and how to use them
- ☐ Explanations about input and output parameters
- ☐ Program structure diagram
- ☐ Description of process IPO
- ☐ Supplementary explanations about process IPO
- ☐ A listing of messages
- ■ Program overview
- ☐
- ☐

(3) Program configuration diagram

A module configuration diagram and a list of interfaces between modules are attached. (See Figure 4-2-3.) A diagram of contents made using the HIPO technique can be used as a module configuration diagram.

(4) Processing details

A document describing details of processing to be executed by each module is attached to the processing details. IPO diagrams, etc. are helpful.

(5) Test specifications

A test plan describing test personnel, test items and verification methods is prepared to conduct program tests efficiently.

(6) Description of data items

The screen used to run a program and input/output data are attached to the description of data items. The screen and input/output data provided by external and internal design can be used as is.
The following documents are attached to the description of data items:

- Input forms and output reports
- Screen design document
- File design document
- Table specification
- Other documents

4.4.2 Points to note when creating program design documents

The following points should be considered when creating program design documents:

<Important points to consider>

- Functions described in an internal design document, must be described in a program design document without omission.
- All input and output data must be clearly described.
- All rules on error handling must be clearly described.
- The criteria for preparing documents must be observed and misleading expressions must be avoided.

4.4.3 Design review

A design review is performed when the work of program design is completed, as in the case of the internal design stage. (Refer to Chapter 3 for more detail.)

(1) Documents to be reviewed

Documents to be reviewed in the program design stage are:

- Program design document
- Program structure diagram
- Program specification
- Module specification
- Test case design document
- Other documents

(2) Review personnel

Personnel shown below play a central role in performing a design review:

- Designers who have the same level of technical skills as those who are directly responsible for designing a program
- Personnel who concern a design process

Superiors of above designers and personnel do not need to participate in a design review.

Exercises

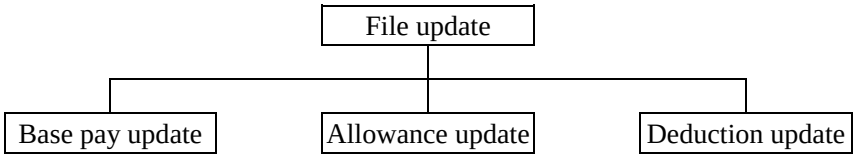
Q1 **Which is <u>inappropriate</u> as a remark made concerning the work of module partitioning in the program design stage?**

a. The number of subordinate modules that one module can invoke must be limited.
b. One module must be designed so that it contains a proper number of steps.
c. In designing a hierarchical structure in which one module invokes the other, care should be taken to keep the depth within specified limits.
d. Interfaces between modules must be simplified.
e. Proper comments should be included to make the logic inside a module easy to understand.

Q2 **When a program for reading data, choosing numeric data alone and showing the average value is subjected to STS partitioning, functions are sorted into sink, source and transform categories. Choose a right combination from combinations shown below.**

	Functions			
	Inputting data	Choosing numbers	Calculating the average value	Showing the result
a	Sink	Sink	Source	Transform
b	Sink	Source	Source	Transform
c	Source	Source	Transform	Sink
d	Source	Transform	Transform	Sink
e	Transform	Sink	Sink	Source

Q3 **There is a program for accepting base pay update, allowance update and deduction update slips and updating a payroll calculation file. This program was partitioned into modules, as shown below. Which module partitioning method was used?**



a. STS partitioning method
b. Jackson method
c. Transaction partitioning method
d. Warnier method

Q4 **Which is the method that you should use to convert a data flow diagram created by structured analysis into a structure chart to be used for structured design?**

a. KJ method b. OMT method c. Jackson method d. Transaction partitioning method

Q5 **Which one is a data structure-oriented module partitioning technique?**

a. Common function partitioning method
b. Source/transform/sink partitioning method (STS partitioning method)
c. Jackson method
d. Transaction partitioning method (TR partitioning method)

Q6 **Which of the remarks shown below best describes the Warnier method used to create the structured design of a program?**

a. Structure diagrams of input and output data are drawn with the main attention given to the structure of data. A program structure diagram is then prepared based on an input/output data structure diagram.
b. Functions in the flow of data are grouped into source, transform and sink categories with the main attention given to the flow of data to deal with.
c. Software is considered a collection of data and processes. The module independence is increased by encapsulating these data and processes.
d. With the main attention given to the control structure of a program, program logic is designed based on the control flow that shows invocation relationships.

Q7 **Module coupling is a measure of module independence. The weaker the module coupling, the higher the level of module independence becomes. Which of module coupling types shown below has the strongest coupling?**

a. Common coupling b. Stamp coupling c. Data coupling d. Content coupling

Q8 **When the following programs are executed, which of the results shown below do you get? x (formal argument) is a call by value and y is a call by reference.**

```
Main program              Subprogram  sub (x, y)
    a=3 ;                   x=x+y ;
    b=2 ;                   y=x+y ;
    sub (a, b) ;            return ;
```

a. a=3, b=2 b. a=3, b=7 c. a=5, b=2 d. a=5, b=7

5 Program Implementation

Chapter Objectives

In the program implementation stage, an information system designed according to procedures so far described, is constructed.
This chapter describes important points to note when doing the work of programming (coding), testing methods and various development tools that we can use.

① Understanding programming paradigms and programming styles
② Understanding various types of testing, testing methods and testing procedures
③ Understanding types and characteristics of various development tools that you can use for programming and testing

Introduction

Program implementation is to give a concrete form to a logically designed program. Specifically, it includes the process of creating a program based on the contents of program design, and the process of conducting various tests before running the program as a system.

5.1 Programming

Programming is to describe (code) in a programming language a procedure (algorithm) defined by program design.
Each programming language has its own meanings of instructions and the syntax of uniting them, and so describes the algorithms in different ways. Therefore, a general standard (programming paradigm) is required with consideration given to the characteristics of each programming language.
Because the work of programming is done by a group of people, a clear-cut programming style is necessary to ensure consistency within a system.

5.1.1 Programming paradigm

Each programming language used has its own paradigm. In doing the work of programming, it is necessary to understand each individual programming paradigm.
Because a paradigm is dependent on each programming language, it must be studied for each specific language. We classify it broadly into four types according to the classification of the programming languages:

- Procedural programming paradigm
- Logic programming paradigm
- Functional programming paradigm
- Object-oriented programming paradigm

(1) Procedural programming paradigm

Procedural programming is the paradigm of procedural programming languages which describes problem solutions in terms of a series of procedures. C and COBOL are representative procedural programming languages.
One characteristic of this programming language is structured programming. The concept of structured programming is to express all algorithms using three basic control structures (sequence, selection and repetition). By using this programming scheme, we can minimize the use of the go to statements that cause deterioration in the maintenance work.

(2) Logic programming paradigm

Logic programming is the paradigm of a logic programming language that describes a problem solution in terms of logical declarations. Prolog is a representative logic programming language.
The characteristic of this language is the resolution rule based on syllogism. In the case of Prolog, three syntaxes are used: rule, fact and query. Pattern matching (unification), automatic searching (back tracking), etc. are used as basic controls.

(3) Functional programming paradigm

Functional programming is the paradigm of a functional programming language that describes a problem solution in terms of function declarations. LISP is a representative functional programming language.
The characteristic of this language is the use of a list structure. Because objects (data) to be processed are dealt with as list structures, many functions are provided to process lists. To describe and define functions, a highly abstract language system called the lambda calculus is used.

(4) Object-oriented programming paradigm

Object-oriented programming is the paradigm of an object-oriented programming language that uses an object having data and procedures (behaviors) encapsulated. Smalltalk and Java are representative object-oriented programming languages.
The characteristics of this programming language are:

- Encapsulation function
  A function of dealing with data (attributes) and procedures (methods) as one entity
- Instance construction function
  A function of instantiating an abstract class
- Inheritance function
  A function of inheriting the properties of a superclass to a subclass
- Message passing function
  A function of passing messages between objects

5.1.2 Programming style

A programming style is the base on which a program is created. As a system becomes large in scale, many developers take part in the work of programming and, therefore, a clear-cut programming style is required.
A programming style is determined from the following viewpoints:

- Clarity
- Efficiency
- Maintainability

(1) Clarity

Clarity means the understandability of a program. To increase the level of clarity, coding rules (standards) are generally established.
Coding rules specify the rules to follow when doing the coding work: indenting, naming of variables and modules, comments, etc. By following the rules, one can create a program that is understandable to other people. The coding work done in obedience to the rules, seems to be more time-consuming than the coding work done without rules. The coding work done according to the rules eventually leads to a reduction in the time spent on a peer-review, however, allowing the overall time for program development to be curtailed.

(2) Efficiency

Efficiency means the ease of creating a program. To increase efficiency, redundant portions of a program must be removed as much as possible. However, we should note here that the clarity of a program may be improved by supplemental portions of a program. For example, although a comment is a supplemental portion that does not concern program execution, it contributes to increasing the level of clarity of a program. Therefore, it is necessary to keep clarity and efficiency well balanced.

(3) Maintainability

Maintainability means the ease of modifying a program. A person who does the work of programming usually does not modify a program. This is done by another person. To make the work of maintenance easy, it is important to create an easy-to-understand program.

To improve maintainability, a program itself should be highly structured to prevent a modification made in one part, from affecting other parts in a program.

5.1.3 Use of language processors

A language processor is a generic term for programs that perform translation and editing tasks to make a coded program executable.
Representative language processors are:

- Compiler that translates a program written in a high-level language into a machine language all at once
- Assembler that translates a program written in an assembler language into a machine language all at once
- Interpreter that translates and executes statements in a program written in a high-level language one by one.

In doing the work of programming, characteristics of each of these language processors are used.
Characteristics of an interpreter are:

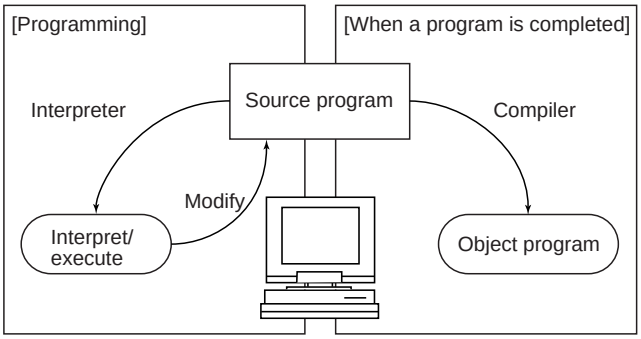
- Statement-by-statement execution is possible.
- A program can be run even if it is not yet completed.

By taking advantage of these characteristics, you can obtain the following benefits:

- The behavior of a program can be verified at a point where it is halfway completed.
- The work of debugging is made easy (debugging instructions can be inserted easily).

The execution speed of a program written in an interpreter, however, is slower than that of a compiled program. In the actual work of programming, therefore, an interpreter is first used to complete a program, then a compiler is used to speed up its execution.

Figure 5-1-1 Using language processors



Another commonly used language processor is a preprocessor. In the narrow sense, a preprocessor is used to perform macro expansion in a program statement or to import files. In the actual work of programming, you should be aware that a preprocessor is running to perform these functions. In the broad sense, a preprocessor is used to convert a program written in one high-level language into a program written in the other high-level language. Using a preprocessor, therefore, you can first build a program using a programming language x, then convert it to the format of a programming language y and run this converted program. This method of using one language processor in combination with the other, is useful if the compiler y is more widely used than the compiler x, or if using two different language processors can increase the optimization efficiency. Because the prerequisite for this programming technique is smooth conversion from x to y, x is usually an extended version of y.

5.1.4 Programming environment

Language processors and other various development tools are included in the program development environment. A programming system that has had increasing use in recent years is the IDE that has these language processors and development tools integrated into one system.
Also, development tools designed to support new programming techniques such as web programming are being developed; web programming tools, for example.

(1) IDE (integrated development environment)

The IDE allows a series of programming tasks from source program editing to compiling, and from linking to debugging to be carried out in one seamless environment. The work of programming was done using different tools to perform each programming task. Therefore, the smooth flow of programming was interrupted, causing efficiency and productivity to suffer. The IDE provides a solution to these problems, allowing all programming tasks to be performed in one integrated environment.
Characteristics of the IDE are as follows:

- Compatible with conventionally used programming languages (C, COBOL, etc.)
- Designed to allow a series of programming tasks to be visually performed
- Linkage with a database management system

Representative IDE products are:

- Visual Studio .NET (Microsoft): The Windows development environment using VB, VC# and others as one package
- Delphi 7 Studio (Borland): The visual development system using high-speed compilers
- Oracle9i Developer Suite (Oracle): Integrated suite of application development and business intelligence tools, including such tools as JDeveloper and Forms Developer and others.

(2) Web programming

There are two approaches taken to create web programming using the web environment:

- Making an existing program executable in the web environment
- Developing a new program with the intent of running it in the web environment

A development tool suitable for the above first approach is VB-web and one suitable for the second approach is FrontPage. Both are provided by Microsoft.
Functions required on a web development tool are:

- Simple GUI construction functions
- Program skeleton design functions using a wizard
- Use of the ASP (Active Server Page)
- Compatibility with conventionally used languages

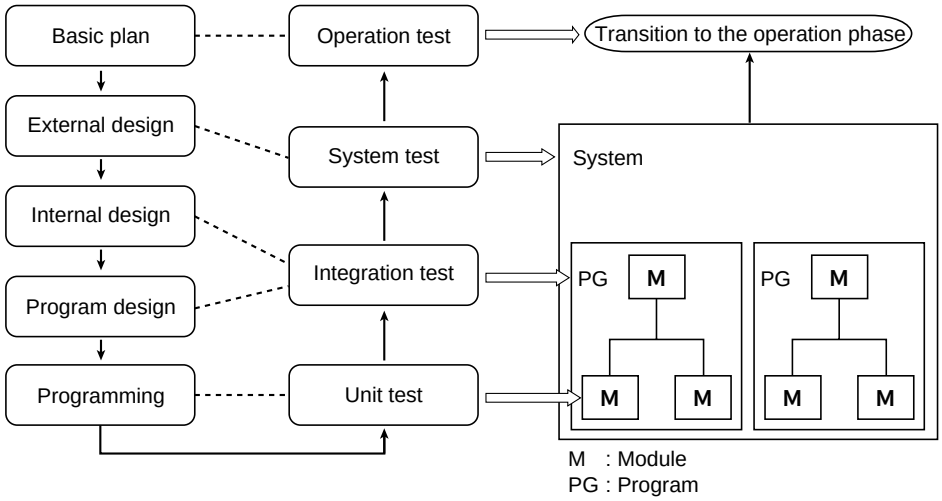
5.2 Test

A newly created program is said to contain several bugs in 100 lines. In these lines, bugs from the programming process and also those from the design process are included. If a program containing bugs is used to operate an online system, serious damage will be caused not only to the company that operates the system, but to the public at large.
Therefore, product tests must be conducted prior to shipment of industrial products. Program tests must also be conducted in a specified order of testing to verify that a program and a system controlled by it, can function according to specifications. Although we cannot guarantee the complete removal of bugs in a program, we can reduce the number of them to the lowest possible level if we test a program in an efficient, accurate way.

5.2.1 Overview of tests

In the waterfall model, unit test, integration test, system test and operation test are conducted in that order. The system development side takes the initiative in conducting unit, integration and system tests, while the user department is responsible for conducting the operation test.

Figure 5-2-1 Test overview



When the user department completes an operation test, a program is officially passed from the system organization to the user. After this, the user organization takes the responsibility for program management.

5.2.2 Unit tests

Unit tests are conducted at the earliest stages in the test phase. The target to be tested is each module.

(1) Outline and purpose of a unit test

A unit test is conducted for each module, the smallest unit dealt with in system development. The purpose is to verify the work which has been done in the programming phase. In an actual unit test, whether a module functions according to a module specification or not is verified.
After modules have been linked and integrated to the program system level, a large amount of workload is necessary to remove bugs. To avoid this, bugs must first be removed from each module in the unit testing

stage, before modules are integrated.

(2) Test methods and designing test cases

① Test methods

In principle, a black box test is usually conducted. A white box test is conducted if necessary.
- White box test: Main attention is given to the internal structure.
- Black box test: Main attention is given to interfaces (input and output) between modules.

② Designing test cases

Before testing a program, test cases (test data) must be prepared. It is an important factor to consider since it affects the results of testing, or even determines the quality of a system.
To design optimal test cases, a certain guideline (test case design manual) will be very useful if available. By accumulating data and revising the test case design manual, the development organization can store the know-how to improve software quality, and use it as a means to pass data to the next stage of programming.

5.2.3 Integration tests

Integration tests are conducted after unit tests have been completed. They are intended to verify the works which were done in the program design and internal design phases.

(1) Outline and the purpose of an integration test

An integration test is conducted to verify that multiple modules can function properly when linked with other related modules. In conducting this test, the main attention is given to the interfaces between modules (interfaces between programs). Even if no problem is found during a unit test, errors often occur when modules are linked. If errors occur during an integration test, you must go back to the previous process and correct the problem. Keep in mind that you must also correct a design document.
Before conducting an integration test, in what order and when the modules are linked must be clearly defined.

(2) Stubs and drivers

In the programming stage of system development, a program has a hierarchical structure comprised of several modules. Therefore, coding is performed for each individual module, and high- or low-level modules are required to verify the normal functioning of developed modules. In an actual test, dummy modules called a stub or a driver respectively are used.
- Stub: A program for testing to provide functions of low-level modules
- Driver: A program for testing to provide functions of high-level modules

Figure 5-2-2 Stubs and drivers

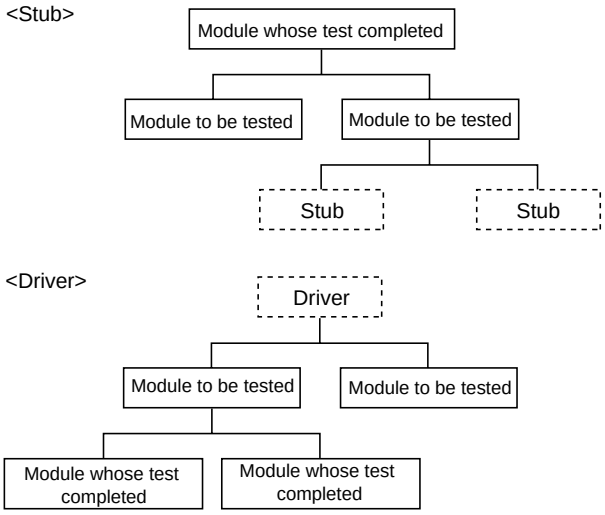
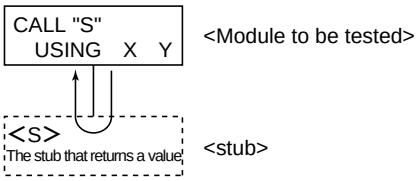


Figure 5-2-3 shows examples of stubs.

Figure 5-2-3 Example of a stub



(3) Incremental test

In an incremental test, modules whose testing has been completed are successively linked with other modules. An incremental test is classified broadly into three tests shown below:

- Top-down test
- Bottom-up test
- Combination test (sandwich test)

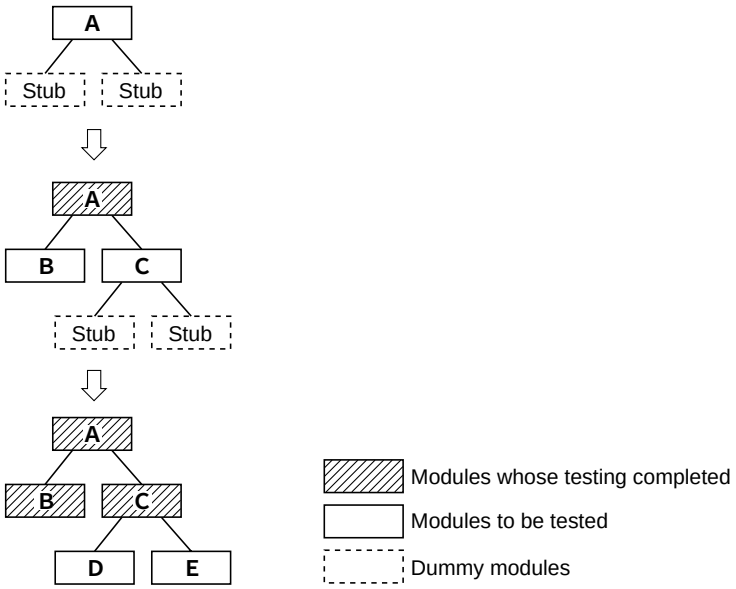
<Characteristics>

- Suitable for testing a large-size program
- It is necessary to use such test modules (dummy modules) as stubs and drivers instead of unfinished modules.
- Results of testing may vary, depending on in what order modules are linked.
- It is easy to trace errors back to causes.

① Top-down test

A top-down test is used to develop a system in the order of high- to low-level modules (called top-down programming).

Figure 5-2-4 Top-down test



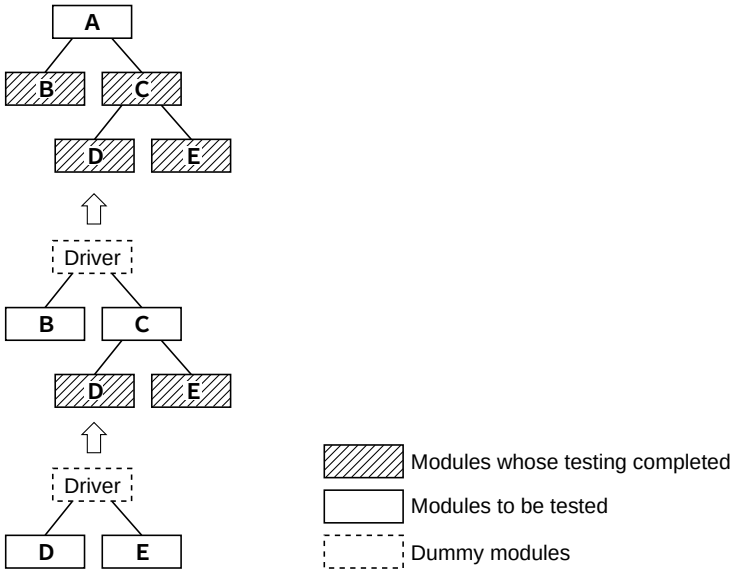
<Characteristics>

- The highest-level module (a kernel module or a module that has logic) is first linked with the next highest module, and all other modules are likewise linked in the order of high- to low-level modules.
- Important modules are tested more frequently than less important modules, thus increasing the reliability of interfaces between high-level modules.
- A precondition for using this test is that a program itself must be produced using the structured design.
- Because a high-level module with a small number of programs is first developed, it is difficult to do the work of programming and conduct tests in parallel at the initial stage.
- Suitable for developing a new system
- As a test system (program), stubs are required.

② Bottom-up test

A bottom-up test is used to develop a system in the order of low- to high-level modules (called bottom-up programming).

Figure 5-2-5 Bottom-up test



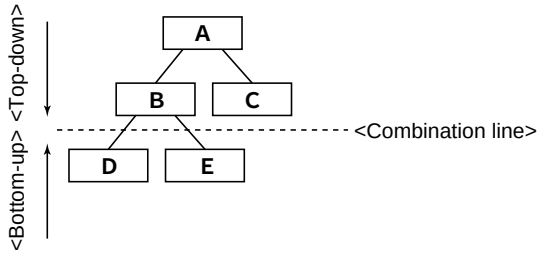
<Characteristics>

- A test is conducted by linking modules in the order of low- to high-level modules.
- When an integration test has been completed, a program can be tested under the actual operation condition.
- Because low-level modules with a large number of programs are first developed, it is possible to do the work of programming, and conduct a test in parallel at an initial stage.
- As a test system (program), a driver is required.
- Suitable for developing a modified version of an existing system

③ Combination test (sandwich test)

A combination test (sandwich test) is a combination of the top-down and bottom-up tests. Top-down and bottom-up tests are conducted at the same time until a predefined compromise line is reached.

Figure 5-2-6 Combination test



<Characteristics>

- Modules above a combination line are subjected to a top-down test while those under a combination line are subjected to a bottom-up test.
- Because top-down and bottom-up tests can be conducted simultaneously, tests can be completed in a much shorter time.
- The skeleton part of a program can be tested early.
- As a test system (program), both stubs and a driver are required.

(4) Nonincremental test

In this test, all modules whose unit testing have been completed are linked and run. A representative nonincremental test is the big bang test.

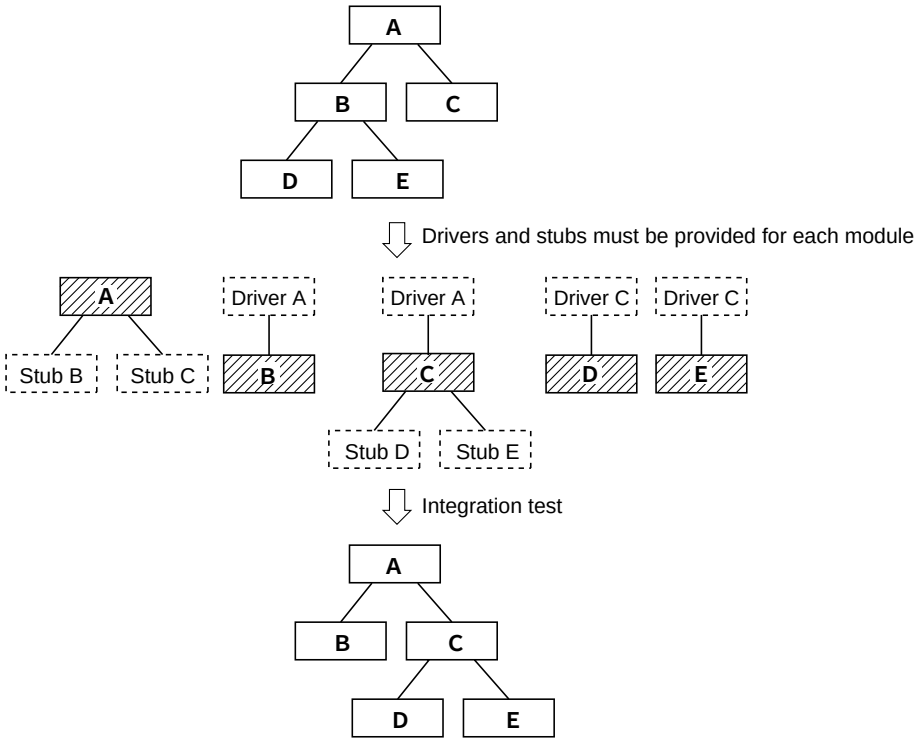
<Characteristics>

- Suitable for testing a small-size program
- A stub or a driver is not required.
- If errors occur, it is difficult to trace them back to causes.

① Big-bang test

A big-bang test uses the technique of first performing a unit test on all modules, then linking them all at one time and performing an integration test entirely.

Figure 5-2-7 Big bang test



<Characteristics>

- Because this test is conducted after a unit test, the results are highly reliable.
- To conduct a linkage test, a stub or a driver is not required.
- Modules can be tested all at once.
- If a unit test is not completed, a big bang test cannot be conducted.
- It is difficult to find errors in the interfaces between modules early.
- After errors are found, the debugging is cumbersome.

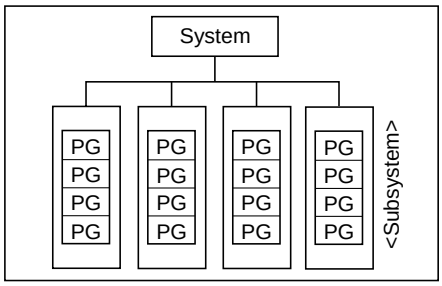
5.2.4 System tests

After modules linked with each other are tested, tests are conducted in the order of each individual program, then testing of each individual subsystem, and finally testing of the entire system.
Tests to be conducted after an integration test are called system tests.

(1) System test overview

A system test is conducted to verify the appropriateness of external design. In conducting this test, the most attention is given to the interfaces between subsystems. A system test is called a total test, and is conducted by a group specialized in testing. It is the last test conducted with the initiative taken by the system development organization.

Figure 5-2-8 Scope of a system test



<<Programs are integrated and the entire system is tested>>

(2) Types of system test

A system test is conducted to check functions and performance from various different angles shown below:

① Program/subsystem integration test

Programs are linked to each other, and subsystems are linked with each other, and subsystems and interfaces between programs are tested.

② Functional test

This test is conducted to verify whether users' functional requirements are met or not.

③ Performance test

This test is conducted to verify a response time and other performance items.

④ Operability test

A human interface (GUI) and other points regarding operability are checked and verified.

⑤ Failure recovery test

How a system can recover from failure and resume its functions is tested.

⑥ Load test

This test is conducted to check performance and functions of a system when large amounts of data are inputted at one time, or when a large load is applied to a system.

⑦ Exception test

This test is conducted to verify that a system can adequately handle illegal data when it is inputted.

⑧ Endurance test

This test is conducted to verify that a system can withstand many hours of continuous operation.

5.2.5 Other tests

There are also an operation test (to be conducted after a system test is completed) and a regression test.

(1) Operation test

The user organization is responsible for conducting an operation test. An operation test is the last test to be conducted on a system. The user organization must prepare test cases, conduct a test under the actual operation conditions, and verify that a system satisfies the required specifications. Because this test is intended to have a developed system accepted by the user organization, it is called an approval test or an acceptance test.
Because an operation test is conducted by running a program on a machine being used for actual business operations, care should be exercised not to interfere with the business operations.

(2) Regression test

A regression test is closely related to maintenance activities.
The purpose of a regression test is to verify that system modifications made during maintenance work, do not affect other normally functioning parts of a system.

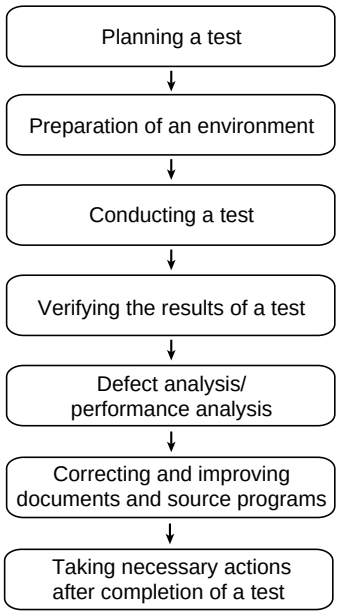
5.2.6 Testing plan and tasks

Various tests conducted during system development have so far been described. This section explains a general test tasks and the contents of work done to conduct a test.

(1) Testing tasks (overview)

Figure 5-2-9 shows an overview of testing tasks.

Figure 5-2-9 Testing tasks

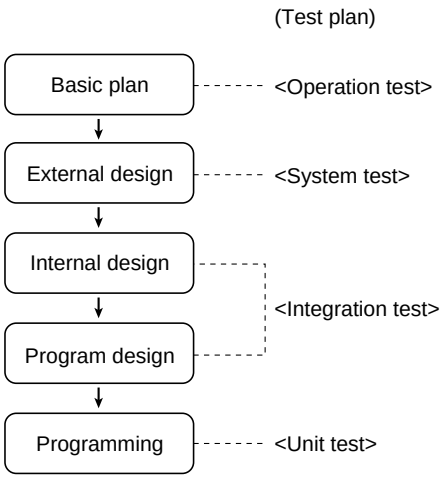


(2) Test flow and contents of tasks

① Test Plan

A test plan is prepared. A test plan has a close relationship with the work of system development, as shown in Figure 5-2-10.

Figure 5-2-10 Test plan



A general plan called a master schedule is prepared. The schedules in the basic plan phase of conducting tests are determined at this point. Each test in Figure 5-2-10 should be scheduled with the timing shown below:

<When to determine a schedule for each test>

- Operation test: To be determined when a basic plan is made
- System test: To be determined during external design
- Integration test: To be determined during internal design or program design
- Unit (module) test: To be determined during programming (module design)

② Preparation of an environment for testing

The following must be prepared before conducting each test:

- Test system
- Test data
- Test programs (modules)
- Test tool (software package)

a. Designing a test system

There is such a case where a machine being used for actual business operations must be used to run a program for testing. However, if the machine stores files or a database being accessed in daily operations, or if a large load is applied to the machine during testing, production work will be disturbed. Therefore, it is necessary to develop or design the same operating system, files and database as those which will be used in an actual production runs.

b. Designing a test program and test data

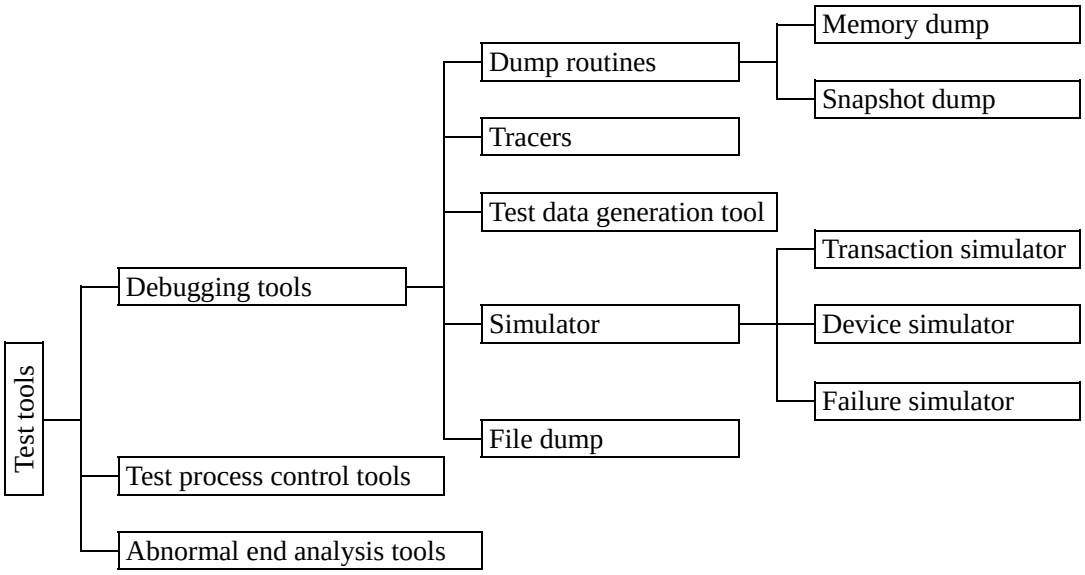
A test program (stubs and drivers) must be designed. Also, test data must be prepared and test cases must be designed using various methods.
A recently developed tool that can easily generate test data based on simple parameters can be used. Such a tool is very convenient since erroneous data can be intentionally generated.

c. Installing test tools

Test tools must be installed. Utility programs of an operating system are generally used as test tools. Software packages designed as test tools are now abundantly available. The operating environment of such a package should be checked and introduced to reduce development costs, and to increase the

efficiency and quality of the testing.
Figure 5-2-11 shows some test tools.

Figure 5-2-11 Test tools



1) Debugging tools
   - Dump routines
     A dump routine outputs the content of the memory or a register.
     - Memory dump: Writing the content memory or a register when an abnormal end occurs
     - Snapshot dump: With a debugging instruction embedded in a program, and the contents of the memory or a register are written out each time the instruction is executed
   - Tracer
     A tracer is also called a trace program. Each time an instruction is executed and certain conditions are met, a tracer is activated and records the content of a register and the address of a referred memory area.
   - Test data generation tool
     A test data generation tool automatically generates test data based on parameters provided by the user.
   - Simulator
     - A transaction simulator simulates the online transaction processing system.
     - A device simulator simulates a terminal device.
     - A failure simulator simulates failure conditions.
   - File dump
     A file dump program writes out the content of a file stored on such a storage medium as magnetic disks or magnetic tape.

2) Test process control tool
   A test process control tool can support all test processes from test planning and design to debugging.

3) Abnormal end (abend) analysis tool
   An abnormal end analysis tool can find the cause of an abnormal end that occurs to a program, and present corrective actions.

The above test tools should be utilized as much as possible with consideration to available budgets. They are effective tools that can contribute to the improvement of productivity. As a result, the quality of a system will improve, the maintenance cost after a system becomes operational can be reduced, and better returns on investments can be expected.

Also, the following points should be considered before conducting a test:

- Creating a function for processing test data in a lump by means of JCL, shell script or batch file
- Improving failure, network and other simulation environments (by using test tools)
- Automating interactive processing

③ Conducting a test

Various tests must be conducted in well prepared execution environments conditions.

④ Checking the results of testing

Tests are conducted in accordance with test plans and test specifications, and the results of testing must be examined.

⑤ Failure analysis, performance analysis

Detected bugs and failures must be closely analyzed using tools and various quality control (QC) techniques.

⑥ Correcting and improving documents and source programs

If bugs or design mistakes are found and if they can be corrected immediately, a source program must be corrected or improved. The portion of a design document that concerns such bugs or design mistakes must also corrected. If only the source program is corrected, the consistency between the source program and a design document is lost, and trouble may occur when the work of verification of other portions or maintenance is done.

⑦ Taking necessary actions after a test is completed

After a test has been completed and a program has been corrected, the following actions must be taken:

a. Managing the progress of testing and reporting

A person in charge must report the progress of testing using a weekly work report or a test report. After a test is completed, he must report the results of testing using a test completion report.

b. Controlling failure-related data

Data on bugs or defects found during testing must be accumulated. Also, in what situation bugs occurred, what corrective action was taken and other detailed information must be kept.

c. Revising upstream operation manuals

A mistake found in a human interface or in interfaces between subsystems is attributable to external design, as described in item ⑥ above. Not only the portion of a program that led to such mistakes must be corrected, but also the external design document itself must be corrected to prevent the same mistake from happening again.
In order to prevent the recurrence of bugs caused by a mistake made in an upstream process, a program, or that portion of a design document that concerns such a mistake as well as a corresponding operation manual, must be corrected or revised to prevent another occurrence of the same mistake.

Exercises

Q1 **In conducting tests in the system development stage, tests are conducted in the order of small to large units, and results of testing are accumulated as data. Which is the most appropriate order of testing?**

a. System test → integration test → unit test
b. System test → unit test → integration test
c. Unit test → integration test → system test
d. Unit test → system test → integration test

Q2 **What is the most appropriate term for a test that is conducted with the most attention given to the internal structure of a program and algorithms?**

a. System test
b. Top-down test
c. Black box test
d. White box test
e. Bottom-up test

Q3 **What is the technique for preparing test data, and testing the functions of a program with the most attention given to the relationships between input data and output results?**

a. Top-down test
b. Black box test
c. Bottom-up test
d. White box test

Q4 **Which one is an appropriate description of the integration test that is conducted in the system development test process, immediately after a unit test (module test) is completed?**

a. Verifying that a system can flawlessly perform all functions specified in an external design document
b. Verifying that the goals set for processing time and response time goals have been achieved
c. Verifying that there is no problem in the types and the number of input and output devices and communications equipment to be connected
d. Verifying that there is no problem with interfaces between modules which are the components of a program
e. Verifying that the running of multiple jobs and the simultaneous connection of terminals can be realized as specified

Q5 **Which one is the proper explanation about the bottom-up test, one of the testing techniques?**

a. A test is conducted by linking modules in the order of low- to high-level ones. Drivers are required as substitutes for uncompleted high-level modules.
b. Each individual module is tested. When all modules have been tested, they are then linked and tested.
c. A test is conducted by linking modules in the order of high- to low-level ones. Stubs are required as substitutes for uncompleted low-level modules.
d. Tests are conducted in the order of unit, integration, system and operation tests.

Q6 **Which one is the proper description of test data that is used to inspect a program?**

a. Test cases are prepared beforehand, and test data that can meet the requirements specified in the test cases are prepared.
b. Only test data that can be processed correctly are prepared as testing progresses.
c. As data to be used for testing, about 20% of data volumes to be processed during actual operations are prepared.
d. Test data that is rejected as errors during the input stage does not need to be provided.

Q7 **Which is a debugging technique for writing out the contents of variables or registers each time a specified statement is executed?**

a. Walk-through
b. Snapshot
c. Test data generator
d. Driver

Index