CS2403 Programming Languages

Data Type

Chung-Ta King
Department of Computer Science
National Tsing Hua University
Modified by apirak

(Slides are adopted from *Concepts of Programming Languages*, R.W. Sebesta)

Overview

- Various Data Types (Sec. 6.2~6.9)
  - Primitive Data Types, Character String Types, User-Defined Ordinal Types, Array Types, Associative Arrays, Record Types, Union Types, Pointer and Reference Types
- Type Binding (Sec. 5.4.2)
- Type Checking (Sec. 6.10)
- Strong Typing (Sec. 6.11)
- Type Equivalence (Sec. 6.12)
- Type Conversions (Sec. 7.4)

Variable type : Why you should define variable type

- Specify memory size for storing data.
  (Compiler/OS considers types of variable.)
  - character 1 byte
  - integer 2/4 bytes
  - float/real 2/4 bytes

  Compiler/OS considers types of variable.

- Specify encoding data format in memory

- Define its belonging operators

```
#include <stdio.h>


int main( )
{
   int  x1 , x2 , x3 ;


    printf ("Pls, enter 3 values\n" );
     scanf("%d", &x1) ;
     scanf("%d", &x2) ;
     scanf("%d", &x3) ;


  if ( x1 > x2 && x1>x3 )
       printf ("max = %d",x1 );
  else  if ( x2 > x1 && x2 > x3 )
       printf( "max = %d", x2 );
  else  if ( x3 > x1 && x3 > x2 )
       printf( "max = %d", x3 );
  return 0;
}
```

Variable type : Why you should define variable type

Specify memory size for storing data.

(Compiler/OS considers types of variable.)

- character 1 byte
- integer 2/4 bytes
- float/real 2/4 bytes

Compiler/OS considers types of variable.

```
#include <iostream.h> //Turboc
//#include <iostream>  //Gnu C
//using namespace std; //Gnu C
int main( )
{
 int i; //declare i as integer
 char c; //declare c as char
 float f; //declare f as float
 int ai[10]; //declare ai as array of integer

 cout  << " sizeof( i ) = " << sizeof ( i ) << '\n';
 cout  << " sizeof( c ) = " << sizeof ( c ) << '\n';
 cout  << " sizeof( f ) = " << sizeof ( f ) << '\n';
 cout  << " sizeof( ai ) = " << sizeof ( ai ) << '\n';
 return 0 ;
}
```

Data Types

Type	Length	Range
unsigned char	8 bits	0 to 255
char	8 bits	-128 to 127
enum	16 bits	-32,768 to 32,767
unsigned int	16 bits	0 to 65,535
short int	16 bits	-32,768 to 32,767
int	16 bits	-32,768 to 32,767
unsigned long	32 bits	0 to 4,294,967,295
long	32 bits	-2,147,483,648 to 2,147,483,647
float	32 bits	3.4 × (10××-38) to 3.4 × (10××+38)
double	64 bits	1.7 × (10××-308) to 1.7 × (10××+308)
long double	80 bits	3.4 × (10××-4932) to 1.1 × (10××+4932)

รูปที่ 3.4.1 ตารางข้อกำหนดขนาดชนิดข้อมูลของบริษัท Borland

Type	Storage
char, unsigned char, signed char	1 byte
short, unsigned short	2 bytes
int , unsigned int	4 bytes
long, unsigned long	8 bytes
float	4 bytes
double	8 bytes
long double	8 bytes

รูปที่ 3.4.2. ตารางข้อกำหนดขนาดชนิดข้อมูลของบริษัท Microsoft

Data type specify encoding data format in memory

value -> Type

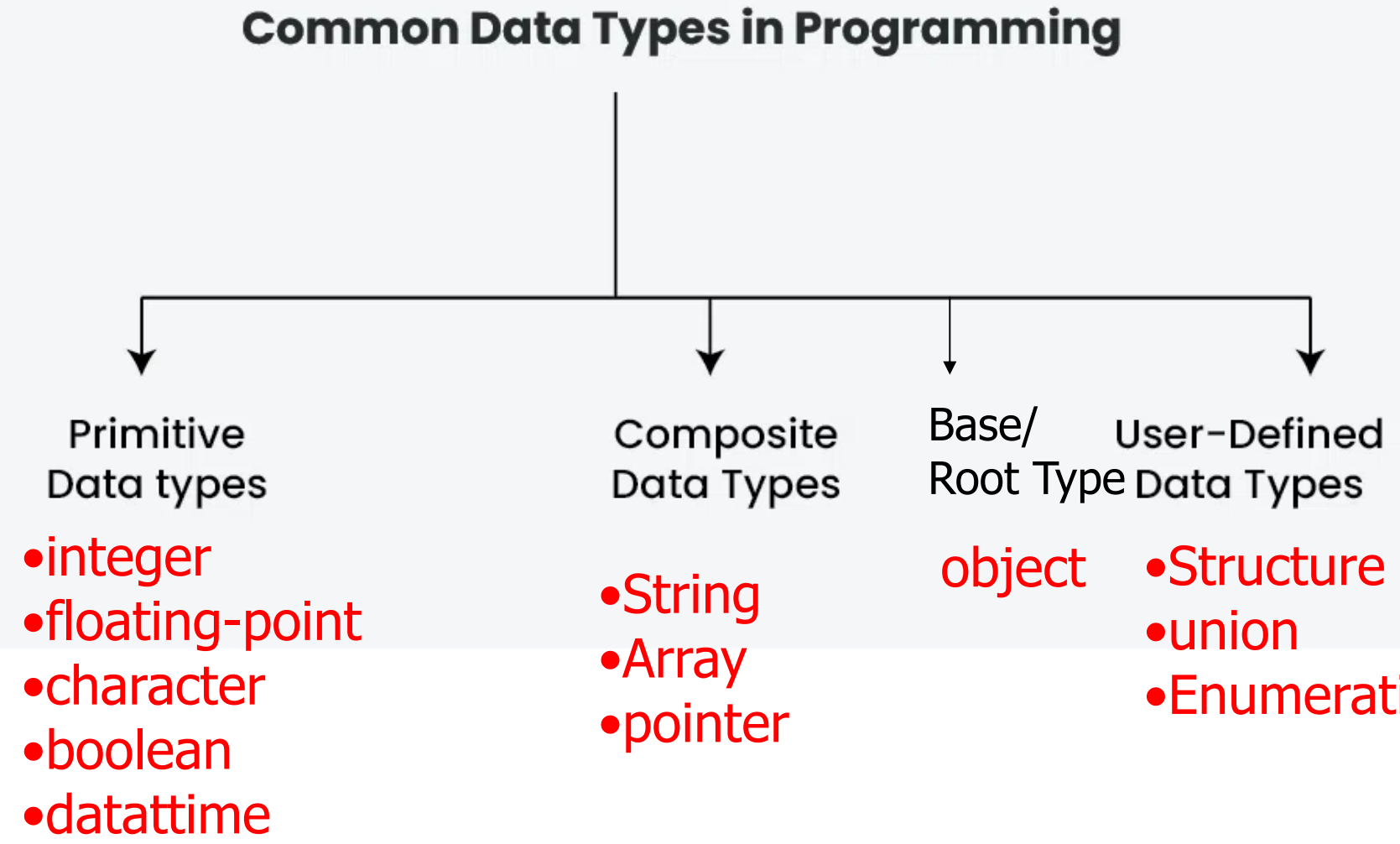
addr	8 bits	value
10	0000-0000	10.0
11	0000-0000	
12	0010-0000	
13	1000-0001	
14	0000-1010	10
15	0000-0000	
16	0010-0000	
17	0000-0000	
18	1100-0000	"0"
19	1100-0001	"1"
20	1100-0000	"0"
21	0110-1110	"."
22	1100-0000	"0"
23	1100-0001	"1"

float f =10.0; -> floating point constant (4-bytes)

int i = 10; -> integer constant (4-bytes)

Decimal	Hexadecimal	Binary	Octal	Char
48	30	110000	60	0
49	31	110001	61	1
50	32	110010	62	2
46	2E	101110	56	.

Common Data Types in Programming Lang.



ประเภทของตัวแปร : Variable Categories

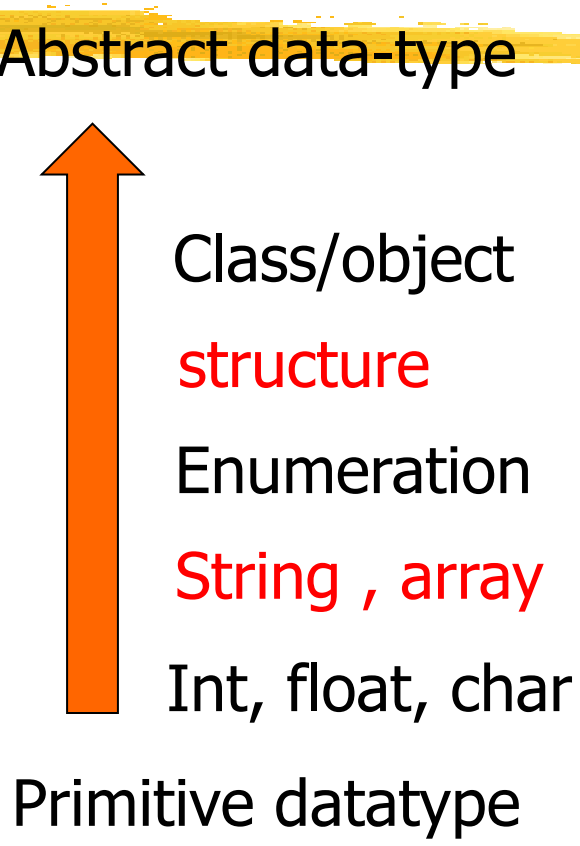
- Mutable variable
  - Variables can be updated their value.
    - int x
    - string y;
- Immutable variable
  - Variables can be not updated their value.
    - static binding time :
      - Const x = 10; //Pascal
      - final int x = 10; //Java
    - Dynamic binding time:
      - const int = f1();

```
1  program ConstExample;
2  const
3     c = 10;
4
5  function f1: Integer;
6  begin
7      f1 := 10;
8  end;
9
10 var
11     x: Integer;
12
13 begin
14     x := f1;
15    // c := f1;
16     x := c + 1;
17     { c := x + 1; }  { This is
18
19     Writeln;
20     Writeln('x = ', x);
21     Writeln('c = ', c);
22 end.
```

```
#include <iostream>

int f1()
{
    return 10;
}

int main()
{
    int x = f1();
    const int c = f1();
    x = c+1;
    //c = x+1;
    std::cout << "\nx = " << x;
    std::cout << "\nc = " << c;
    return 0;
}
```

Evolution of Data Types

- Earlier PL's tried to include many data types to support a wide variety of applications, e.g. PL/I
- Wisdom from ALGOL 68:
  - A few basic types with a few defining operators for users to define their own according to their needs
- From user-defined type to *abstract data type*
  - The interface of a type (visible to user) is separated from the representation and set of operations on values of that type (invisible to user)

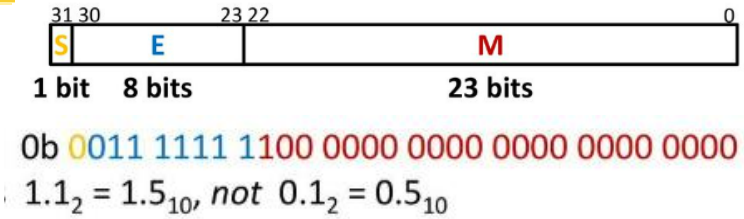


Data type design in programming Lang. issue:

- Declaration statement

```
float f =  1.5;
```

- Data Representation



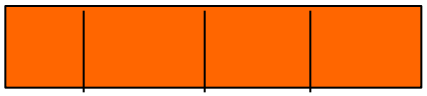
- Values Domain

3.4×10^{-38} to $3.4 \times 10^{+38}$.

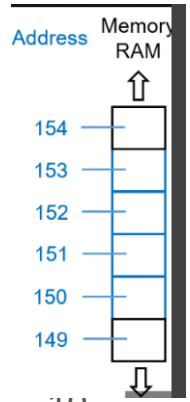
- Operators

+, - , *, /

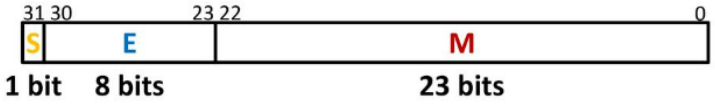
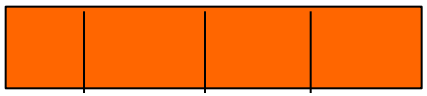
- Physical Implementation

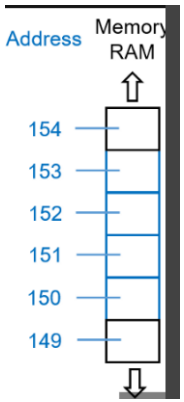


Memory 4 cell



0b 0011 1111 1100 0000 0000 0000 0000 0000

datatype	Primitive (float 32 bits)	String
Declaration stm	`float f = 1.5;`	`String str = "Geeks";`
(Data Representation)	 0b 0011 1111 1100 0000 0000 0000 0000 0000 $1.1_2 = 1.5_{10}$, *not* $0.1_2 = 0.5_{10}$	
Domain	3.4×10^{-38} to $3.4 \times 10^{+38}$.	ascii code
Operator	+, - , *, /	+ , < , >
3.Physical (implementation)	 Memory 4 cell : 0b 0011 1111 1100 0000 0000 0000 0000 0000	



Data Type	Value Domain	Data Representation	Operators
int (C, Java, C#)	จำนวนเต็ม เช่น -2,147,483,648 ถึง 2,147,483,647 (32-bit signed)	ใช้ 4 ไบต์ (32 bits), แบบ two's complement	+, -, *, /, %, ==, <, >
float / double	จำนวนจริง เช่น 3.14, -0.001	IEEE 754 format (32/64 bits)	+, -, *, /, ==, <, > (เปรียบเทียบควรใช้ด้วยความระวัง)
char (C, Java)	ตัวอักษรเดี่ยว เช่น 'A', '7'	1 byte (ASCII) หรือ 2 bytes (UTF-16 ใน Java)	==, <, ++, --, ใช้ร่วมกับ string
bool / boolean	{true, false}	1 byte (แต่จริงๆ ใช้เพียง 1 bit)	!, &&, `
String	ลำดับของตัวอักษร เช่น "hello"	array of characters (C), หรือ immutable object (Python, Java, C#)	+ (concat), ==, [], len(), substring()

Data Type	Value Domain	Data Representation	Operators
array	ลำดับของค่าที่มีชนิดเดียวกัน เช่น [1, 2, 3]	contiguous block of memory	[] indexing, length, loop, assignment
enum (C, Java)	ชุดของค่าที่กำหนดไว้ เช่น RED, GREEN, BLUE	mapped to int (โดยปกติ)	==, switch-case, compare ordinal
struct / class	ประกอบด้วยหลาย field (heterogeneous)	memory layout ตามลำดับ field	.field, method call, operator overloading (ถ้าภาษา support)

Type declaration stm design

- An explicit declaration is a program statement that lists variable names and specifies their types

  var x: int

- An implicit declaration is a means of associating variables with types through default conventions, rather than declaration statements
  - First use of variable: X := 1.2;
    - X is a float and will not change afterwards
    - Var x ; x = 1.2;
  - Default rules • In Fortran, if an undeclared identifier begins with one of the letters I, J, K, L, M, or N, or their lower case versions, it is implicitly declared to be Integer type

ตัวอย่างการออกแบบคำสั่งประกาศตัวแปร
กรณี static data type binding

```c
int main() {
    // ตัวแปรชนิด int (จำนวนเต็ม)
    int age = 25;

    // ตัวแปรชนิด float (จำนวนจริง)
    float height = 175.5;

    // แสดงผล
    printf("Age: %d\n", age);
    printf("Height: %.1f\n", height);

    return 0;
}
```

C/C++

อนุญาติให้มีการกำหนดค่าเริ่มต้น

```go
package main

import "fmt"

func main() {
        // ตัวแปรชนิด int (จ านวนเต็ม)
        age := 25

        // ตัวแปรชนิด float (จ านวนจริง)
        height := 175.5

        // แสดงผล
        fmt.Println("Age:", age)
        fmt.Println("Height:", height)
}
```

Go

การกำนดค่าเริ่มต้นและdatatype
โดยใช้เครื่องหมาย assign

```pascal
program PersonInfo;
var
  age: integer;
  height: real;
begin
  age := 25;          { กำหนดค่าอายุ }
  height := 175.5;    { กำหนดค่าส่วนสูง }

  writeln('Age = ', age, ' years');
  writeln('Height = ', height:0:1, ' cm');
end.
```

pascal

ไม่มีการอนุญาติให้กำหนดค่าเริ่มต้น

C# support both implicit & explict

```csharp
using System;

class Program
{
    static void Main()
    {
        // ตัวแปรชนิด var (compiler จะ infer type ให้อัตโนมัติ)
        var age = 25;           // int
        var height = 175.5;     // double

        // แสดงผล
        Console.WriteLine("Age: " + age);
        Console.WriteLine("Height: " + height);
    }
}
```

```
class Program
{
    static void Main()
    {
        // ตัวแปรชนิด int (จำนวนเต็ม)
        int age = 25;

        // ตัวแปรชนิด float (จำนวนจริง)
        double height = 175.5;

        // แสดงผล
        Console.WriteLine("Age: " + age);
        Console.WriteLine("Height: " + height);
    }
}
```

Type Inference technique:

Improve clean code Java 10

```java
import java.util.ArrayList;

class User {
    private int id;
    private String name;
    // Constructor
    public User(int id, String name) {
        this.id = id;
        this.name = name;
    }
}

public class Main {
    public static void main(String[] args)
    {
        ArrayList<User> users =
                new ArrayList<User>();
    }
}
```

Main.java

```java
import java.util.ArrayList;

class User {
    private int id;
    private String name;
    // Constructor
    public User(int id, String name) {
        this.id = id;
        this.name = name;
    }
}

public class Main {
    public static void main(String[] args)
    {   //java
        var users = new ArrayList<User>();
    }
}
```

```java
// แบบเดิม (verbose)
ArrayList<String> names = new ArrayList<String>();
HashMap<Integer, String> map = new HashMap<Integer, String>();
List<Map<String, List<Integer>>> complexData = new ArrayList<Map<String, List<Integer>>>();

// ใช้ var (concise)
var names = new ArrayList<String>();
var map = new HashMap<Integer, String>();
var complexData = new ArrayList<Map<String, List<Integer>>>();
```

Var in c#

Type Inference

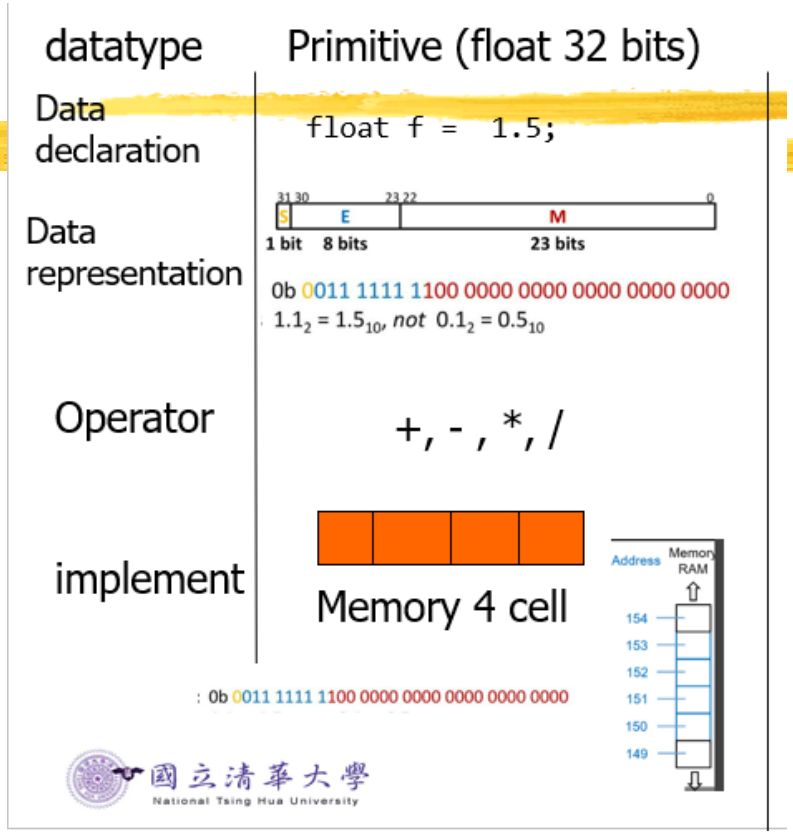
```
namespace ConsoleApplication3
{
    class Program
    {
        public class User
        {
            public string FirstName { get; set; }
            public string LastName { get; set; }
            public int BirthYear { get; set; }
            public double salary { get;set; }
        }

        static void Main(string[] args)
        {
            var users = new List<User>()
                {
                    new User()
                    {
                        FirstName = "Terrance",
                        LastName = "Johnson",
                        BirthYear = 2005,
                        salary= 5000.0
                    },
                    new User()
                    {
                        FirstName = "John",
                        LastName = "Smith",
                        BirthYear = 1966,
                        salary= 15000.0

                    },
                    new User()
                    {
                        FirstName = "Eva",
                        LastName = "Birch",
                        BirthYear = 2002,
                        salary= 20000.0
                    }
                };
```

Var (implicit) in c#

```
45          int currentYear = 2028;
46          //Get the full combined name for people born in 1990 or later
47          var fullNames = from x in users
48                          where x.BirthYear >= 1990
49                         select new { fullname=x.FirstName,
50                                      x.LastName ,x.salary,
51                                      age = currentYear-x.BirthYear  };
52          foreach (var item in fullNames)
53          {
54              System.Console.WriteLine("fullname={0} ,salary={1},age={2}",
55                      item.fullname,item.salary, item.age);
56          }
57
58          var age1990 = from x in users
59                          where x.BirthYear >= 1990
60                          select new { x.FirstName, x.BirthYear };
61          foreach (var item in age1990)
62          {
63              System.Console.WriteLine("FirstName : {0} , BirthYear: {1}", item.FirstName, item.BirthYear);
64          }
65
66          return;
67      }
68  }
69  }
```

Component of Data type

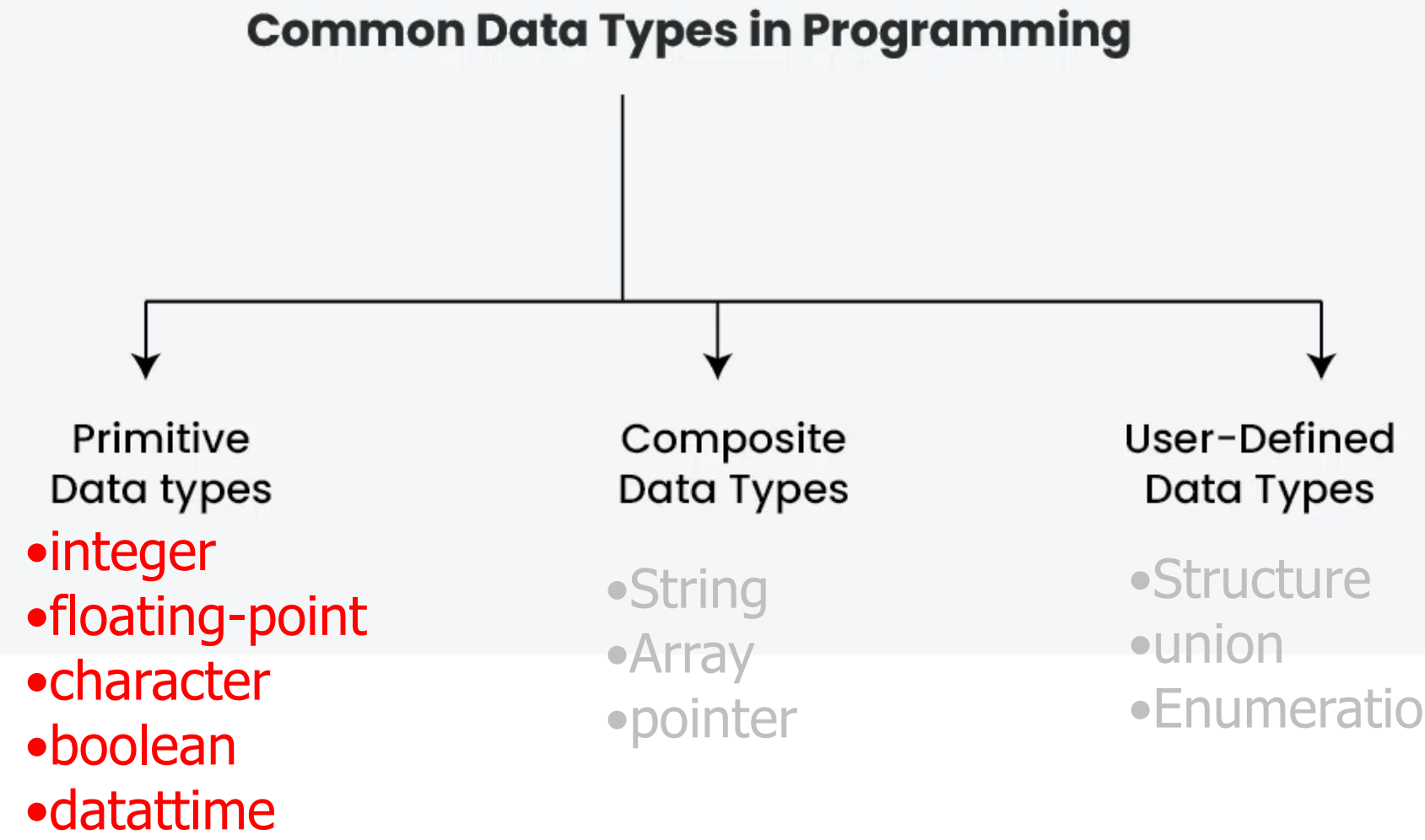
- Value Domain
- Operator
- Data presentation
- Implementation



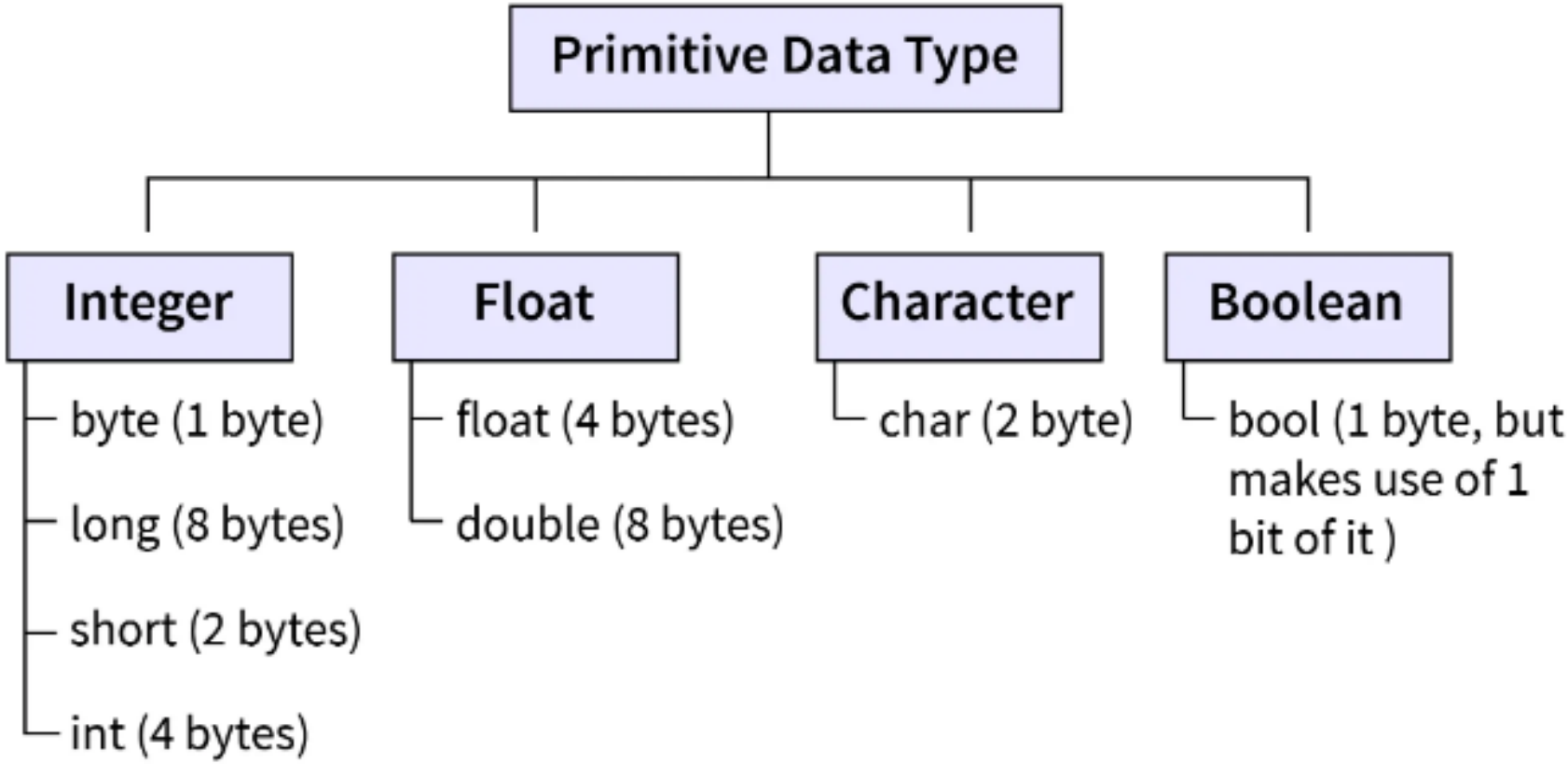
Overview

- Various Data Types (Sec. 6.2~6.9)
  - Primitive Data Types(Integer, float/real,Character), Character String Types, User-Defined Ordinal Types, Array Types, Associative Arrays, Record Types, Union Types, Pointer and Reference Types
- Type Binding (Sec. 5.4.2)
- Type Checking (Sec. 6.10)
- Strong Typing (Sec. 6.11)
- Type Equivalence (Sec. 6.12)
- Theory and Data Types (Sec. 6.13)

Common Data Types in Programming Lang.



การออกแบบ Primitive Data Type



A **primitive data type** is a basic, built-in data type of a programming language that represents a single, simple value and is not composed of other data types.

Key Characteristics

- Built directly into the language
- Stores **one simple value**
- Has a **fixed size** in memory
- Example
  - Integer
  - Character
  - Floating point /Real
  - Complex
  - Decimal

Integer Primitive type

- These sizes of integers, and often a few others, are supported by some programming languages. For example, Java includes four signed integer sizes: byte, short, int, and long.

$$v = \sum_{i=0}^{N-1} 2^i b_i$$

2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	1	1	1	1	1	0	1	0	0	0	0

$V = 0*2^{11} + 1*2^{10} + 1*2^9 + \ldots$
$= 1024 + 512 + 256 + 128 + 64 + 16$
$= 2000$

Integer Primitive type

$$v = \sum_{i=0}^{N-1} 2^i b_i$$

2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	1	1	1	1	1	0	1	0	0	0	0

V = 0*2^{11} + 1*2^{10} + 1*2^9 + …
= 1024 + 512 + 256 +128 + 64 + 16
= 2000

63=0011 1111
1's=1100 0000
2's=1100 0001

```
  0 = 00000000   (1 byte)
  1 = 00000001   (1 byte)
127 = 01111111   (1 byte)
128 = 00000001 00000000   (2 bytes)
```

```
 63 = 00001111   (1 byte)
-64 = 11000000   (1 byte)
-63 = 11000001   (1 byte)
```

Unsigned Integers

2-complement quantities.

signed Integers

ตย. Operatorของ integer

ประเภท Operator	C / C++ / Java / C#	Python	JavaScript	Pascal
Addition (บวก)	`+`	`+`	`+`	`+`
Subtraction (ลบ)	`-`	`-`	`-`	`-`
Multiplication (คูณ)	`*`	`*`	`*`	`*`
Division (หาร)	`/` (ใน C/C++/Java int/int หารแล้วเป็น int)	`/` (ผลลัพธ์ float) หรือ `//` (หารปัดเศษ)	`/` (ผลลัพธ์ float)	`div` (หารปัดเศษ) หรือ `/` (ผลลัพธ์ float)
Modulus (หารเอาเศษ)	`%`	`%`	`%`	`mod`
Increment (เพิ่ม 1)	`++` (prefix/postfix)	ไม่มี (ใช้ `+= 1`)	`++` (prefix/postfix)	ไม่มี (ใช้ `:= var + 1`)
Decrement (ลด 1)	`--` (prefix/postfix)	ไม่มี (ใช้ `-= 1`)	`--` (prefix/postfix)	ไม่มี (ใช้ `:= var - 1`)
Bitwise AND	`&`	`&`	`&`	`and`
Bitwise OR	`	`	`	`
Bitwise XOR	`^`	`^`	`^`	`xor`
Bitwise NOT	`~`	`~`	`~`	`not`
Left Shift	`<<`	`<<`	`<<`	ไม่มี (ต้องใช้ฟังก์ชัน)
Right Shift	`>>`	`>>`	`>>`	ไม่มี (ต้องใช้ฟังก์ชัน)
Comparison (เปรียบเทียบ)	`==`, `!=`, `<`, `>`, `<=`, `>=`	เหมือนกัน	เหมือนกัน	`=`, `<>`, `<`, `>`, `<=`, `>=`

main.cpp

```c
#include <stdio.h>

int main() {
    int x = 11;

    // คูณด้วย 2 โดยการ shift ซ้าย 1 บิต
    int multiplied = x << 1;  // เท่ากับ x * 2

    // หารด้วย 2 โดยการ shift ขวา 1 บิต
    int divided = x >> 1;     // เท่ากับ x / 2 (หารปัดเศษลงสำหรับ int)

    printf("Original x = %d\n", x);
    printf("x << 1 (คูณ 2) = %d\n", multiplied);
    printf("x >> 1 (หาร 2) = %d\n", divided);

    return 0;
}
```

```
Original x = 11
x << 1 (คูณ 2) = 22
x >> 1 (หาร 2) = 5
```

Charecter

- Binary Encoding depends on platform

Dec	Hex	Name	Char	Ctrl-char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	0	Null	NUL	CTRL-@	32	20	Space	64	40	@	96	60	`
1	1	Start of heading	SOH	CTRL-A	33	21	!	65	41	A	97	61	a
2	2	Start of text	STX	CTRL-B	34	22	"	66	42	B	98	62	b
3	3	End of text	ETX	CTRL-C	35	23	#	67	43	C	99	63	c
4	4	End of xmit	EOT	CTRL-D	36	24	$	68	44	D	100	64	d
5	5	Enquiry	ENQ	CTRL-E	37	25	%	69	45	E	101	65	e
6	6	Acknowledge	ACK	CTRL-F	38	26	&	70	46	F	102	66	f
7	7	Bell	BEL	CTRL-G	39	27	'	71	47	G	103	67	g
8	8	Backspace	BS	CTRL-H	40	28	(	72	48	H	104	68	h
9	9	Horizontal tab	HT	CTRL-I	41	29	)	73	49	I	105	69	i
10	0A	Line feed	LF	CTRL-J	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	VT	CTRL-K	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	FF	CTRL-L	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage feed	CR	CTRL-M	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	SO	CTRL-N	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	SI	CTRL-O	47	2F	/	79	4F	O	111	6F	o
16	10	Data line escape	DLE	CTRL-P	48	30	0	80	50	P	112	70	p
17	11	Device control 1	DC1	CTRL-Q	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	DC2	CTRL-R	50	32	2	82	52	R	114	72	r
19	13	Device control 3	DC3	CTRL-S	51	33	3	83	53	S	115	73	s
20	14	Device control 4	DC4	CTRL-T	52	34	4	84	54	T	116	74	t
21	15	Neg acknowledge	NAK	CTRL-U	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	SYN	CTRL-V	54	36	6	86	56	V	118	76	v
23	17	End of xmit block	ETB	CTRL-W	55	37	7	87	57	W	119	77	w
24	18	Cancel	CAN	CTRL-X	56	38	8	88	58	X	120	78	x
25	19	End of medium	EM	CTRL-Y	57	39	9	89	59	Y	121	79	y
26	1A	Substitute	SUB	CTRL-Z	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	ESC	CTRL-[	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	FS	CTRL-\	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	GS	CTRL-]	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	RS	CTRL-^	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	US	CTRL-_	63	3F	?	95	5F	_	127	7F	DEL

ASCII : c/c++

[hide]	Code	Decimal	Octal	Description	Abbreviation / Key
	U+0000	0	000	Null character	NUL
	U+0001	1	001	Start of Heading	SOH / Ctrl-A
	U+0002	2	002	Start of Text	STX / Ctrl-B
	U+0003	3	003	End-of-text character	ETX / Ctrl-C[1]
	U+0004	4	004	End-of-transmission character	EOT / Ctrl-D[2]
	U+0005	5	005	Enquiry character	ENQ / Ctrl-E
	U+0006	6	006	Acknowledge character	ACK / Ctrl-F
	U+0007	7	007	Bell character	BEL / Ctrl-G[3]
	U+0008	8	010	Backspace	BS / Ctrl-H
	U+0009	9	011	Horizontal tab	HT / Ctrl-I
	U+000A	10	012	Line feed	LF / Ctrl-J[4]
	U+000B	11	013	Vertical tab	VT / Ctrl-K
	U+000C	12	014	Form feed	FF / Ctrl-L

Java/c# :unicode

TABLE 6

EBCDIC (IBM MAINFRAME) CHARACTER CODES

Each code is shown in decimal, hexadecimal, and character form.

129	81	a	193	C1	A	240	F0	0
130	82	b	194	C2	B	241	F1	1
131	83	c	195	C3	C	242	F2	2
132	84	d	196	C4	D	243	F3	3
133	85	e	197	C5	E	244	F4	4
134	86	f	198	C6	F	245	F5	5
135	87	g	199	C7	G	246	F6	6
136	88	h	200	C8	H	247	F7	7
137	89	i	201	C9	I	248	F8	8
						249	F9	9
145	91	j	209	D1	J			
146	92	k	210	D2	K	64	40	blank
147	93	l	211	D3	L	76	4C	<
148	94	m	212	D4	M	77	4D	(
149	95	n	213	D5	N	78	4E	+
150	96	o	214	D6	O	79	45	|
151	97	p	215	D7	P	80	50	&
152	98	q	216	D8	Q	90	5A	!

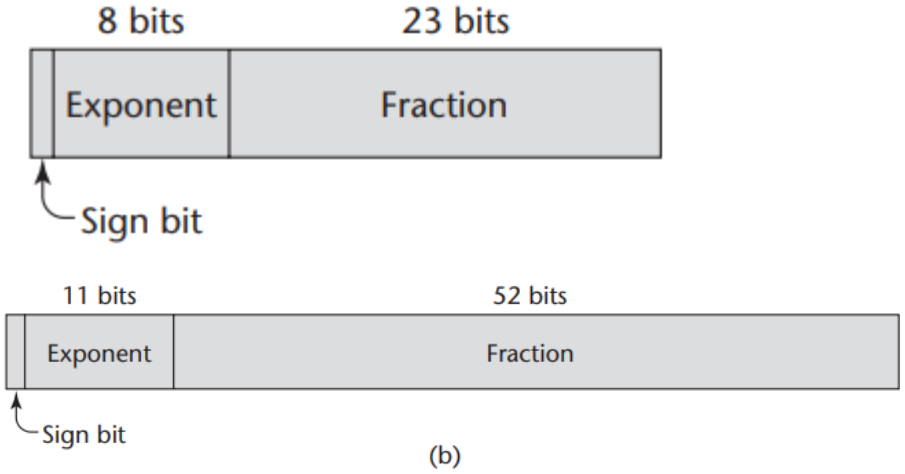
ตัวอย่าง operator ที่ใช้กับ `char` ในภาษาโปรแกรมต่าง ๆ

ประเภท Operator	C / C++ / Java / C#	Python	JavaScript	Pascal
Addition (บวก)	ใช้ได้ (เพิ่มค่ารหัส ASCII)	ใช้ได้ (เพิ่มค่ารหัส Unicode)	ใช้ไม่ได้โดยตรง (แปลงเป็นตัวเลขก่อน)	ใช้ไม่ได้โดยตรง
Subtraction (ลบ)	ใช้ได้ (ลบค่ารหัส ASCII)	ใช้ได้ (ลบค่ารหัส Unicode)	ใช้ไม่ได้โดยตรง	ใช้ไม่ได้โดยตรง
Comparison (เปรียบเทียบ)	`==`, `!=`, `<`, `>` ใช้ได้	ใช้ได้ (`==`, `!=`, `<`, `>`)	ใช้ได้ (`==`, `!=`, `<`, `>`)	ใช้ได้ (`=`, `<>`, `<`, `>`)
Increment/Decrement	`++`, `--` ใช้ได้	ไม่มี (ใช้ `ord()` และ `chr()` แปลง)	ไม่มี	ไม่มี
Concatenation (ต่อ)	ต้องแปลงเป็น string ก่อน	ใช้ `+` ต่อ string ได้	ใช้ `+` ต่อ string ได้	ใช้ `+` ต่อ string ได้ (ไม่ใช่กับ char โดยตรง ต้องแปลง)
Bitwise Operator	ใช้ได้ (เพราะ char เก็บเลข)	ใช้ได้ (char เป็น string 1 ตัว แต่ถ้าเป็น int ก็ได้)	ไม่ได้โดยตรง	ใช้ไม่ได้โดยตรง

Floating Point

- Model real numbers, but only as approximations
- Languages for scientific use support at least two floating-point types; sometimes more
- Usually exactly like the hardware, but not always; some languages allow accuracy specs in code e.g. (Ada)

  type SPEED is digits 7 range 0.0..1000.0;
  type VOLTAGE is delta 0.1 range -12.0..24.0;
- IEEE Floating Point Standard 754
  - Single precision: 32 bit representation with 1 bit sign, 8 bit exponent, 23 bit mantissa
  - Double precision: 64 bit representation with 1 bit sign, 11 bit exponent, 52 bit mantissa




```
1-bit sign | 8-bit exponent | 23-bit mantissa
(sign bit = 1 for negative values)

For example:
   43 4D 40 00 (hex)
re-written in binary:
   0100 0011  0100 1101  0100 0000 0000 0000
re-grouped:
   sign:   exponent(+127):   mantissa:
     0       10000110          (1.)100110101000...
 (positive)    134(dec)
             exponent:
                134-127 = 7
  moving the decimal point 7 positions to the right:
       1   1  0  0  1  1  0  1 .  0   1
     128 64 32 16  8  4  2  1   1/2 1/4 (weights)
  = +205.25
```

Complex

Some programming languages support a complex data type—for example, Fortran and Python. Complex values are represented as ordered pairs of floating-point values. In Python, the imaginary part of a complex literal is specified by following it with a `j` or `J`—for example, `(7 + 3j)`

```
z1 = 6 + 7j
z2 = 1 + 4j

print("Addition of numbers:", z1 + z2)
print("Subtraction of numbers:", z1 - z2)
print("Multiplication of numbers:", z1 * z2)
print("Division of numbers:", z1 / z2)
```

Shell

```
Addition of numbers: (7+11j)
Subtraction of numbers: (5+3j)
Multiplication of numbers: (-22+31j)
Division of numbers: (2-1j)
```

ตัวอย่าง operator ของ `float` ในภาษาโปรแกรมต่าง ๆ

ประเภท Operator	C / C++ / Java / C#	Python	JavaScript	Pascal
Addition (บวก)	`+`	`+`	`+`	`+`
Subtraction (ลบ)	`-`	`-`	`-`	`-`
Multiplication (คูณ)	`*`	`*`	`*`	`*`
Division (หาร)	`/`	`/`	`/`	`/`
Modulus (หารเอาเศษ)	ไม่มี (ใช้ `fmod` ใน C/C++)	ไม่มี (ใช้ `math.fmod()`)	ไม่มี	ไม่มี
Increment (เพิ่ม 1)	`++` (C/C++/Java/C#)	ไม่มี (ใช้ `+= 1.0`)	`++`	ไม่มี (ใช้ `:= var + 1.0`)
Decrement (ลด 1)	`--`	ไม่มี (ใช้ `-= 1.0`)	`--`	ไม่มี (ใช้ `:= var - 1.0`)
Comparison (เปรียบเทียบ)	`==`, `!=`, `<`, `>`, `<=`, `>=`	เหมือนกัน	เหมือนกัน	`=`, `<>`, `<`, `>`, `<=`, `>=`
Bitwise Operators	ไม่มี	ไม่มี	ไม่มี	ไม่มี

Boolean

Boolean

- **Could be implemented as bits, but often as bytes**
- ***Advantage:* readability**

Boolean

- **Such as C# , Java are support : true and false**
- C/C++ : not support : True when the integer value is not equal to 0, False the integer value equal to 0
- ***Advantage:* readability**

```
#include <iostream>
using namespace std;
int main()
{
    int b1 = -1;
    int b2 = 0;

    if(b1) {
        cout << "\nb1 = " << b1;
    }
    if(!b1) {
        cout << "\nnot b1 = " << b1;
    }
    if(!b2) {
        cout << "\nb2 = " << b2;
    }
    return 0;
}
```

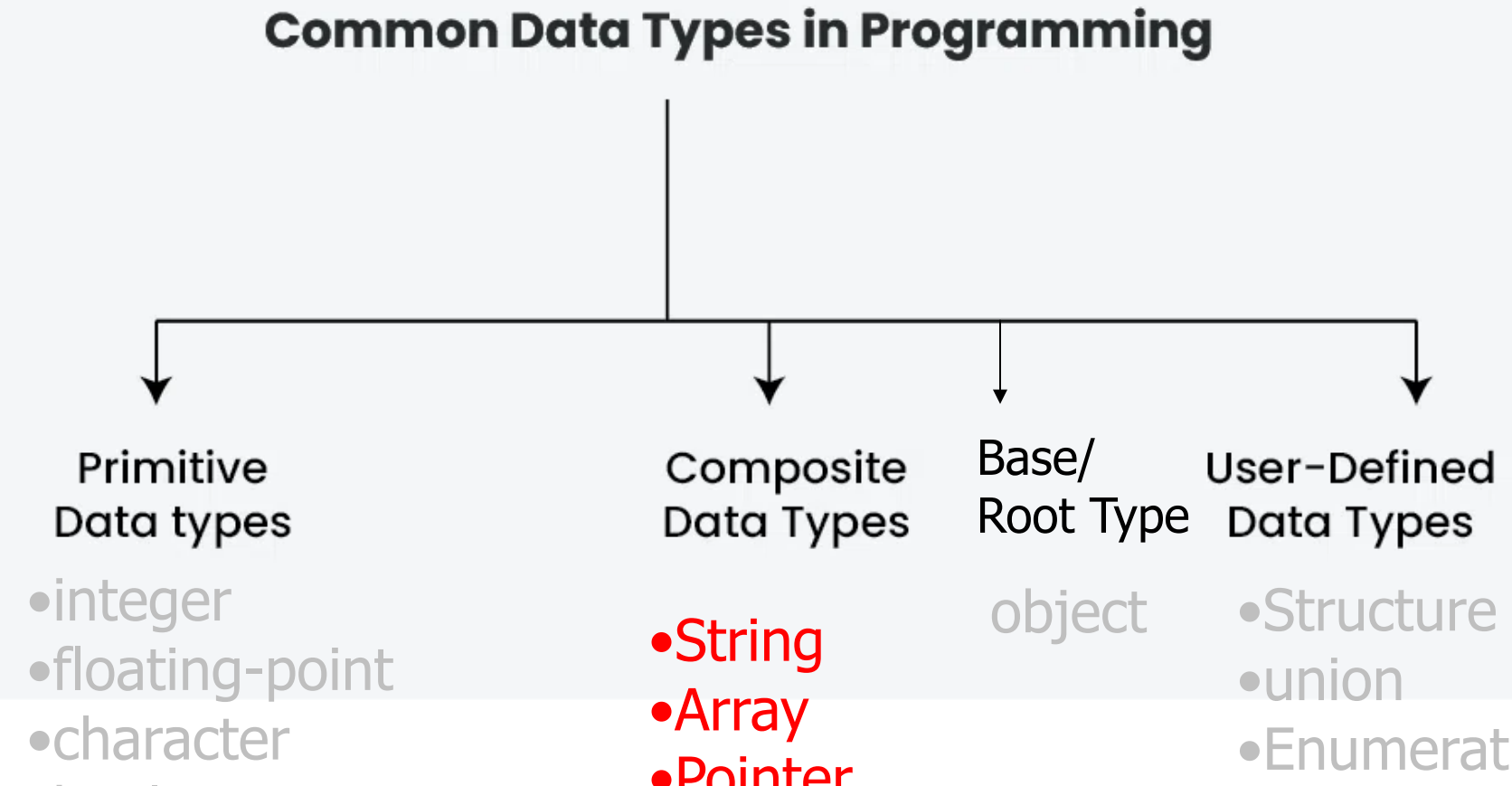
```
b1 = -1
b2 = 0

...Program finished with exit code 0
Press ENTER to exit console.
```

```
using System;

public class HelloWorld
{
    public static void Main(string[] args)
    {
        bool b1 = true;
        bool b2 = false;

        if(b1) {
            Console.WriteLine("b1 = {0}",  b1);
        }
        if(!b1) {
            Console.WriteLine("not b1 = {0}", b1);
        }
        if(!b2) {
            Console.WriteLine( "b2 = {0}", b2);
        }
    }
}
```

```
Output

mono /tmp/niA0PRzA1E.exe
b1 = True
b2 = False
```

Operation	C / C++ / Java / C#	Python	JavaScript	Pascal
Logical AND	&&	and	&&	and
Logical OR	||	||	||	or
Logical NOT	!	not	!	not
Equality	==	==	=== (strict), == (loose)	=
Inequality	!=	!=	!== (strict), != (loose)	<>

Common Data Types in Programming Lang.



Character String Types

- Values are sequences of characters
- Design issues:
  - Is it a primitive type or just a special kind of array?
  - Should the length of strings be static or dynamic?
  - What kinds of string operations are allowed?
    - Assignment and copying: what if have diff. lengths?
    - Comparison (=, >, etc.)
    - Catenation
    - Substring reference: reference to a substring
    - Pattern matching

Character String in C/C++

C and C++

- **Not primitive**; use `char` arrays, terminate with a null character (0)
- A library of functions to provide operations instead of primitive operators
- Problems with C string library:
  - Functions in library do not guard against overflowing the destination, e.g., `strcpy(src, dest);`
    What if `src` is 50 bytes and `dest` is 20 bytes?

```
#include <iostream>
using namespace std;
#include <string.h> //examples of strlen function
int main ( )
{
   char unix1[ ] = { 'u','n','i','1','\0' ,'x','y' }; // 5 elems
   char unix2[5] = { 'u','n','i','x','2' }; // 5 elems
   char unix3[ ] = "uni3"; //5 elems include '\0'
   char unix4[5] = "uni4"; //5 eles include '\0'
   char unix5[ ] = { 'u','n','i','5' }; // 4 elems
   char *unix6   = "uni6";  //5 elems include '\0'

   cout<<"unix1 : " << unix1 << "\t\tstrlen = " << strlen(unix1) << "\tsizeof = "<< sizeof(unix1) << '\n';
   cout<<"unix2 : " << unix2 << "\tstrlen = " << strlen(unix2) << "\tsizeof = "<< sizeof(unix2) << '\n';
   cout<<"unix3 : " << unix3 << "\t\tstrlen = " << strlen(unix3) << "\tsizeof = "<< sizeof(unix3) << '\n';
   cout<<"unix4 : " << unix4 << "\t\tstrlen = " << strlen(unix4) << "\tsizeof = "<< sizeof(unix4) << '\n';
   cout<<"unix5 : " << unix5 << "\tstrlen = " << strlen(unix5) << "\tsizeof = "<< sizeof(unix5) << '\n';
   cout<<"unix6 : " << unix6 << "\t\tstrlen = " << strlen(unix6) << "\tsizeof = "<< sizeof(unix6) << '\n';

}
```

Unix6 ->pointer

Character String in C/C++

- C and C++
  - A library of functions to provide operations instead of primitive operators
  - Problems with C string library:
    - Functions in library do not guard against overflowing the destination, e.g., `strcpy(src, dest);`
      What if `src` is 50 bytes and `dest` is 20 bytes?

```
1
2  #include <iostream>
3  using namespace std;
4  #include <string.h> //examples of strlen function
5  int main ( )
6  {
7     char source[ ] = "source1234567890";
8     char dest[2];
9     char other[ ] = "other";
10   cout << "-- before--" << endl;
11   cout << "source = " << source << endl;
12   cout << "dest = " << dest << endl;
13   cout <<  "other = " <<other << endl;
14   strcpy( dest,source );
15   cout << "--after--" << endl;
16   cout <<  "source = " <<source << endl;
17   cout <<  "dest = " <<dest << endl;
18   cout <<  "other = " <<other << endl;
19
20 }
21
```

```
-- before--
source = source1234567890
dest =
other = other
--after--
source = 34567890
dest = source1234567890
other = urce1234567890
```

Standard string lib in c/c++

Sr.No	#include <string.h>	Function & Purpose
1	**strcpy(s1, s2);** Copies string s2 into string s1.	
2	**strcat(s1, s2);** Concatenates string s2 onto the end of string s1.	
3	**strlen(s1);** Returns the length of string s1.	
4	**strcmp(s1, s2);** Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2.	
5	**strchr(s1, ch);** Returns a pointer to the first occurrence of character ch in string s1.	
6	**strstr(s1, s2);** Returns a pointer to the first occurrence of string s2 in string s1.	

Character String in JAVA/C#

- C#
  - C# string is able access to each string elements with array operator "[]"
  - No have methods to support string operations (e.g. length, <, >) . And dynamic length
  - It is non-primitive type
- Java
  - `String` class Length is set when string is created
  - string is able access to each string elements with method "getCharAt()"
  - No have methods to support string operations (e.g. length, <, >) . And dynamic length
  - It is non-primitive type .

```java

public class Main
{
    public static void main(String[] args) {
        String s1 = "ABC";
        String s2 = "Abc";
        System.out.printf ("\n s1=%s,s2 =%s",s1,s2);
        System.out.printf ("\n s[0]=%c,s[1]=%c",s1.charAt(0),s2.charAt(1));
        System.out.printf ("\n s1==s2 = %s" , s1==s2);
        //System.out.printf ("\n s1 < s2 = %s" ,s1 < s2);
        System.out.printf ("\n s1 + s2 = %s", s1 + s2);
    }
}

```

```
s1=ABC,s2 =Abc
s[0]=A,s[1]=b
s1==s2 = false
s1 + s2 = ABCAbc

...Program finished with exit code 0
Press ENTER to exit console.
```

```c#
using System;
public class HelloWorld
{
    public static void Main(string[] args)
    {
        string s1 = "ABC";
        string s2 = "Abc";
        Console.WriteLine ("s1={0},s2 ={1}",s1,s2);
        Console.WriteLine ("s[0]={0},s[1]={1}",s1[0],s2[1]);
        Console.WriteLine ("s1==s2 ={0}",s1==s2);
       // Console.WriteLine ("s1 < s2 = {0}",s1 < s2);
        Console.WriteLine ("s1 + s2 = {0}",s1 + s2);
    }
}
```

```
mono /tmp/6pso3mwfvS.exe
s1=ABC,s2 =Abc
s[0]=A,s[1]=b
s1==s2 =False
s1 + s2 = ABCAbc
```

PASCAL vs JAVA

```pascal
{https://www.onlinegdb.com/online_pascal_compiler}
program Hello;
 var
   s:string;
begin
  s := '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ';
  s := s+ '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ';
  s := s+ '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ';
  s := s+ '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ';
  s := s+ '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ';
  s := s+ '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ';
  s := s+ '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ';
  s := s+ '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ';
  s := s+ '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ';
  writeln (s);
end.
```

Main.java

```java
public class Main
{
    public static void main(String[] args) {
      String s = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
      s = s+ "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
      s = s+ "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
      s = s+ "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
      s = s+ "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
      s = s+ "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
      s = s+ "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
      s = s+ "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
      s = s+ "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ";
      System.out.printf("%s",s);
    }
}
```

國立清華大學
National Tsing Hua University

String operator

ภาษา	วิธีเปรียบเทียบ string	Operator หรือ Function ใช้	ตัวอย่าง	หมายเหตุ
C	ใช้ฟังก์ชันเปรียบเทียบ string	`strcmp(str1, str2)`	`strcmp(a, b) == 0` คือ เท่ากัน	`==` เปรียบเทียบ pointer (address) ไม่ใช่เนื้อหา
Java	ใช้เมธอด `.equals()`	`str1.equals(str2)`	`if (a.equals(b))`	`==` เปรียบเทียบ reference (ที่อยู่) ไม่ใช่เนื้อหา
Pascal	ใช้ operator เท่ากับ `=`	`a = b`	`if a = b then ...`	ใน Delphi, `=` เปรียบเทียบเนื้อหา string
Python	ใช้ operator เท่ากับ `==`	`a == b`	`if a == b:`	เปรียบเทียบเนื้อหา string

String : index operator

✅ Index Operator สำหรับ `string` (สรุปเปรียบเทียบ)

ภาษา	Index Operator	ตัวอย่าง	Index เริ่มที่	หมายเหตุสำคัญ
C	`[]`	`s[0]`	0	string คือ `char[]`
Java	`charAt(i)`	`s.charAt(0)`	0	ใช้ method ไม่ใช่ `[]`
Pascal	`[]`	`s[1]`	1	Pascal เริ่ม index ที่ 1
Python	`[]`	`s[0]`	0	รองรับ index ติดลบ

Character String Implementation

- Static length: compile-time descriptor(turbol PASCAL)
- Limited dynamic length: may need a run-time descriptor for length (but not in C and C++)
- Dynamic length: need run-time descriptor; allocation/deallocation is the biggest implementation problem

Static string
Length
Address

Limited dynamic string
Maximum length
Current length
Address

Array Types Design

- Array:
  - Homogeneous Array: an aggregate of homogeneous data elements, in which an individual element is identified by its position in the aggregate such as C/C++,C#
    - Int myArray[10] = { 10,11,12,13 };
  - **Heterogeneous arrays** : An aggregate of **heterogeneous** data elements, is identified by its position in the aggregate such as javascript , php

```
let products = [
   ["iPhone 14", 35900, true],        // [ชื่อ, ราคา, มีของไหม]
   ["Samsung A55", 12900, false],
   ["Xiaomi Redmi", 7990, true]
];

// แสดงรายการทั้งหมด
for (let p of products) {
   console.log(`ชื่อ: ${p[0]}, ราคา: ${p[1]} บาท, ${p[2] ? "มีของ" : "หมด"} `);
}
```

ARRAY

- An **array** is a data structure that stores a **fixed-size, ordered collection of elements of the same data type**, where each element is accessed using an index
  - A **heterogeneous array** is a data structure in which **elements may have different data types**, and each element is typically stored as a **reference or object**, rather than in a uniform contiguous memory block.
  - A **homogeneous array** stores elements of a single data type in contiguous memory locations, whereas a heterogeneous array allows elements of different data types and is typically implemented using references.

✅ Example Comparison Table: C vs Python

Comparison Topic	C (Homogeneous Array)	Python (Heterogeneous List/Array)
Structure type	True array	List (used as an array)
Element data type	Same data type for all elements	Different data types allowed
Declaration example	`int a[4] = {10,20,30,40};`	`arr = [10, 3.14, "ABC", True]`
Elements in example	10, 20, 30, 40 → all are `int`	10 (`int`), 3.14 (`float`), `"ABC"` (`str`), `True` (`bool`)
Homogeneous / Heterogeneous	✅ Homogeneous	✅ Heterogeneous
Memory storage	Contiguous memory	Reference-based
Data access	`a[0]` , `a[1]`	`arr[0]` , `arr[1]`
Type checking	Compile-time	Run-time
Performance	Very fast	Slower
Flexibility	Low	High

python

```python
# Each employee is a list with mixed data types: [name, age
employees = [
    ["Alice", 30, 55000.75, True],
    ["Bob", 25, 48000.00, False]
]

# Access and print each employee's data
for i, emp in enumerate(employees):
    print(f"Employee {i+1}:")
    print(f"  Name: {emp[0]} ,Type: {type(emp[0])}")
    print(f"  Age: {emp[1]},Type: {type(emp[0])}")
    print(f"  Salary: {emp[2]},Type: {type(emp[0])}")
    print(f"  Employed: {emp[3]},Type: {type(emp[0])}")
    print()
```

```
Employee 1:
  Name: Alice ,Type: <class 'str'>
  Age: 30,Type: <class 'str'>
  Salary: 55000.75,Type: <class 'str'>
  Employed: True,Type: <class 'str'>

Employee 2:
  Name: Bob ,Type: <class 'str'>
  Age: 25,Type: <class 'str'>
  Salary: 48000.0,Type: <class 'str'>
  Employed: False,Type: <class 'str'>
```

Some languages started simple with **arrays as generic containers** before introducing strict typing or classes.

Array Design

- Homogeneous vs Heterogeneous

- Can arrays be initialized at allocation time?
- What types are legal for subscripts? -> integer, Boolean
- When are subscript ranges bound?
- When does array allocation take place?
- How many subscripts are allowed?
- Operator : Assign , slices allowed

การออกแบบการกำหนดขนาดและกำหนดค่าเริ่มต้น (Declare Data type and initial value)

- C/C++ , Pascal : Static binding
  - Array Size must be predefined (can be changed)

```
1.Int arr[5] = {1,2,3,4,6};     // c/c++
2. var
        arr: array[1..5] of integer;  { Pascal }
```

- Java, C# : Dynamic binding
  - Not nee predefined array size (can be changed

```
int n = 5;                          //java
int[] arr = new int[n];
```

java

Main.java

```java
public class Main {
    public static void main(String[] args) {
        // วิธี 1: ประกาศพร้อมกำหนดค่า
        int[] arr1 = {1, 2, 3, 45};
        // วิธี 2: ประกาศพร้อม new และกำหนดค่า
        int[] arr2 = new int[ ] {1, 2, 3, 45};

        //can change memory
        arr1 = new int[ ] {1, 2, 3, 45};

        // วิธี 3: ประกาศขนาดแล้วกำหนดค่าทีละตัว
        int n = 5;
        int[] arr3 = new int[n];
        arr3[0] = 1;
        arr3[1] = 2;
        arr3[2] = 3;
        arr3[3] = 45;
        arr3[4] = 0;

    }
}
```

C#

```csharp
using System;

public class HelloWorld
{
    public static void Main(string[] args)
    {
        // ประกาศพร้อมกำหนดค่า (ไม่ต้องระบุขนาด)
        int[] arr = {1, 2, 3, 45};


        // ประกาศด้วย new พร้อมกำหนดค่า (ไม่ต้องระบุขนาด)
        int[] arr2 = new int[] {1, 2, 3, 45};

        // ระบุขนาด array พร้อมกำหนดค่าได้เลย (ต่างจาก Java)
        int[] arr3 = new int[5] {1, 2, 3, 45, 0};

        // หรือประกาศขนาดแล้วกำหนดค่าทีละตัวก็ได้
        int n1=5;
        int[] arr4 = new int[n1];
        arr4[0] = 1;
        arr4[1] = 2;
        arr4[2] = 3;
        arr4[3] = 45;
        arr4[4] = 0;

    }
}
```

Array Index : What types are legal for subscripts & specify range checking

- Array index types:
  - FORTRAN (start with 1), C, Java (start with 0): integer only
  - Pascal: any ordinal type (integer, Boolean, char)
  - user defines index rangeArray index types:

- Range checking :
  - C, C++, Perl, and Fortran do not specify range checking,
  - Java, ML, C# ,Java : dynamic checking
  - PASCAL : static checking

หาความถี่ช่วง -5 ถึง 5 โดย pascal

main.pas

```
1  program FrequencyRange;
2
3  var
4    freq: array[-5..5] of integer;
5    data: array[1..15] of integer = (-3, 0, 5, -1, -5, 2, 3, 5, -5, 1, 0, -2, 4, -3, 5);
6    i, val: integer;
7
8  begin
9
10   for i := -5 to 5 do
11     freq[i] := 0;
12
13   for i := 1 to length(data) do
14   begin
15     val := data[i];
16     if (val >= -5) and (val <= 5) then
17       freq[val] := freq[val] + 1;
18   end;
19
20
21   writeln('Value : Frequency');
22   for i := -5 to 5 do
23     writeln(i, ' : ', freq[i]);
24 end.
25
```

จงเขียนโปรแกรมนับความถี่ของข้อมูล 'B' - 'H'

main.pas

```
1  program FreqBH;
2
3  var
4    freq : array['B'..'H'] of integer;  { B..H }
5    c : char;
6    text : string;
7    i : integer;
8  begin
9    text := 'ABBCDEFFGHBBCHHG';
10
11   for c := 'B' to 'H' do
12     freq[c] := 0;
13
14   for i := 1 to length(text) do
15   begin
16     c := text[i];
17     if (c >= 'B') and (c <= 'H') then
18     begin
19       freq[c] := freq[c] + 1;
20     end;
21   end;
22
23  for c := 'B' to 'H' do
24     writeln(  c, ' =', freq[c]);
25 end.
26
```

สมมุติว่า การคำนวณค่าขนส่งสินค้า มีสูตร คือ

ค่าขนส่งสินค้า (cost) = น้ำหนักของส่ง (w) x ค่าส่งต่อปอนด์ (R)

ซึ่งค่าส่งต่อปอนด์ (R) ขึ้นอยู่กับโซน (IZ)

รหัสโซน : IZ	ค่าขนส่ง/ปอนด์ : R
West	.5
North	.75
Central	1.05
East	1.40
South	1.70

จงเขียนโปรแกรมอ่านโซนปลายทาง (IZ) และน้ำหนักของของส่งของ (W) และพิมพ์โซนน้ำหนัก และ ค่าส่ง

Array Index:ใช้ร่วมกับ set

```
program Hello;
 type
        Zone = (West, North,Central, East, South);
 var
         ZI : array [Zone] of real;
         W : real;
         R : real;
         Z   :Zone;
begin
    ZI[West] := 0.5;
    ZI[North] := 0.75;
    ZI[Central] := 1.05;
    ZI[East] := 1.4;
    ZI[South] := 1.7;

    write( 'Zone (West,North,Central,East,South)' ); readln(Z);
    write( 'Weigth  = ' ); readln(W);
    R := ZI[Z] *w ;
    writeln( 'Zi (', Z ,') = ' ,ZI[Z] );
    writeln( 'R  = ' ,R );
end.
```

```
Linking a.out
23 lines compiled, 0.0 sec
Zone (West,North,Central,East,South)Central
Weigth  = 3.9
Zi (Central) =  1.05000000000000000E+000
R  =  4.09499999999999998E+000
```

main.pas

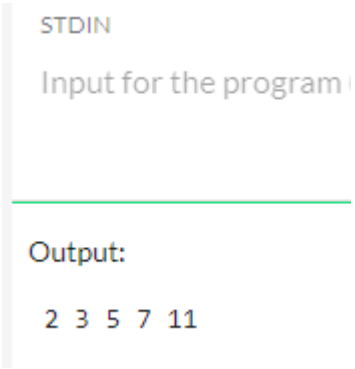
```
program WeekArray;

type
  //set
  TWeekDay = (Mon, Tue, Wed, Thu, Fri);

var
  workHours : array[TWeekDay] of integer;
  dayName   : array[TWeekDay] of string;
  rateOt : array[TWeekDay] of real;

  hours    : array[TWeekDay] of real;
  empName : string;

  d : TWeekDay;
  normalH, otH : real;
  normalPay, otPay, totalPay : real;
  normalRate : real;
begin

  dayName[Mon] := 'Monday';
  dayName[Tue] := 'Tuesday';
  dayName[Wed] := 'Wednesday';
  dayName[Thu] := 'Thursday';
  dayName[Fri] := 'Friday';

  workHours[Mon] := 8;
  workHours[Tue] := 4;
  workHours[Wed] := 5;
  workHours[Thu] := 7;
  workHours[Fri] := 6;

  rateOt[Mon] := 1.8;
  rateOt[Tue] := 1.4;
  rateOt[Wed] := 1.5;
  rateOt[Thu] := 1.7;
  rateOt[Fri] := 1.6;
```

Try to read the code.

```
  { ---------- Input ---------- }
  write('Enter employee name (normal rate=',normalRate:0:2,'): ');
  readln(empName);

  writeln('Enter working hours for each day (Mon-Fri):');
  for d := Mon to Fri do
  begin
    write('Day ', dayName[d] ,' (',workHours[d] ,') hours: ');
    readln(hours[d]);
  end;

  for d := Mon to Fri do
  begin
    if hours[d] > workHours[d] then
    begin
      normalH := normalH + workHours[d];
      otH     := otH+(hours[d] - workHours[d] );
      otPay   := otPay + (hours[d] - workHours[d] )*rateOt[d];
    end
    else
      normalH   := normalH+hours[d];
  end;
  normalPay := normalH * normalRate;
  totalPay := normalPay + otPay;

  { ---------- Output ---------- }
  writeln('----------------------------');
  writeln('Employee: ', empName);
  writeln('Normal Hours = ', normalH:0:2);
  writeln('OT Hours     = ', otH:0:2);
  writeln('Normal Pay   = ', normalPay:0:2);
  writeln('OT Pay       = ', otPay:0:2);
  writeln('Total Pay    = ', totalPay:0:2);

end.
```

https://onecompiler.com/ada

```
with Ada.Text_IO; use Ada.Text_IO;

procedure Greet is
   type My_Int is range 0 .. 1000;
   type Index is range -2 .. 2;

   type My_Int_Array is
     array (Index) of My_Int;
   --                ^ Type of elements
   --       ^ Bounds of the array
   Arr : My_Int_Array := (2, 3, 5, 7, 11);
   --                    ^ Array literal
   --                      (aggregate)

   V : My_Int;
begin
   for I in Index loop
      V := Arr (I);
      --        ^ Take the Ith element
      Put (My_Int'Image (V));
   end loop;
   New_Line;
end Greet;
```



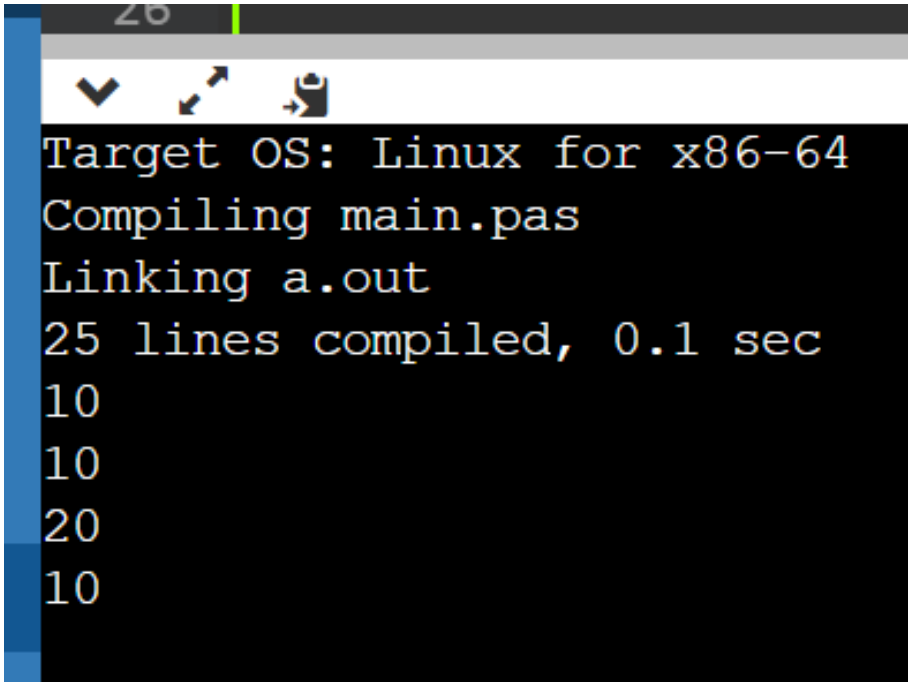
Array Operations

- Assigment (A = B)
  - octave (MATLAB) , PASCAL
- array Concatenation/Shink
  - octave (MATLAB)
  - C/C++ , JAVA not support
- Arithmetic operation
  - octave (MATLAB) : + , * , - , /
  - C/C++ , JAVA not support

https://www.onlinegdb.com/online_pascal_compiler

Assign : copy value right -> left

```
program Hello;
var
   n2: array [1..10] of integer;   (* n is an array of 10 integers *)
   n1: array [1..10] of integer;   (* n is an array of 10 integers *)
begin
    n1[1] := 10;
    n2 := n1;
    Writeln( n1[1]);
    Writeln( n2[1]);

    n1[1] := 20;
    Writeln( n1[1]);
    Writeln( n2[1]);

end.
```



C# & java not support assign '='

Main.cs

```
1  using System;
2
3  class Program
4  {
5      static void Main()
6      {
7          int[] n1 = new int[3];
8          int[] n2 = new int[3];
9
10         n1[0] = 10;
11         n2    = n1;
12
13         Console.WriteLine(n1[0]);  // 10
14         Console.WriteLine(n2[0]);  // 30
15
16         n1[0] = 20;
17
18         Console.WriteLine(n1[0]);  // 20
19         Console.WriteLine(n2[0]);  // 30
20     }
21 }
22
```

d = b (: ,2 : 3)

https://octave-online.net/

Support : +,-,*,% =

```
a =

   2   4   1
   2   3   9
   3   1   8

b =

   10   20   30
   30   60   10
   20   40   70
```

a = [2, 4, 1; 2, 3, 9 ; 3, 1, 8]

b = [10, 20, 30 ; 30, 60, 10 ; 20, 40, 70]

a+b

c = [a ; b]

d = b (: ,2 : 3)

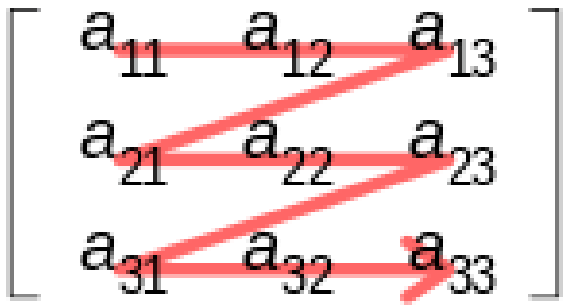
```
octave:6> a+b
ans =

   12   24   31
   32   63   19
   23   41   78
```

```
octave:7> c = [ a ; b ]
c =

    2    4    1
    2    3    9
    3    1    8
   10   20   30
   30   60   10
   20   40   70
```

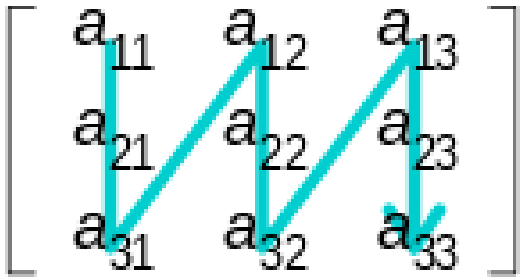
```
octave:10> d = b ( : ,2 : 3 )
d =

   20   30
   60   10
   40   70
```

Memory Layout :Row- and column-major order

Row-major order



Column-major order



For example, the array

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}$$

int A[2][3]

could be stored in two possible ways:

Address	Row-major order	Column-major order
0	a_{11}	a_{11}
1	a_{12}	a_{21}
2	a_{13}	a_{12}
3	a_{21}	a_{22}
4	a_{22}	a_{13}
5	a_{23}	a_{23}

2D array conceptual memory representation

Second subscript →

first subscript ↓

abc[0][0]	abc[0][1]	abc[0][2]	abc[0][3]
abc[1][0]	abc[1][1]	abc[1][2]	abc[1][3]
abc[2][0]	abc[2][1]	abc[2][2]	abc[2][3]
abc[3][0]	abc[3][1]	abc[3][2]	abc[3][3]
abc[4][0]	abc[4][1]	abc[4][2]	abc[4][3]

int abc[5][4]

abc[0][1]	abc[0][2]	abc[0][3]	abc[1][0]	abc[1][1]	....	abc[4][2]	abc[4][3]
82206	82210	82214	82218	82222		82274	82278

memory locations for the array elements →

Array is of integer type so each element would use 4 bytes that's the reason there is a difference of 4 in element's addresses.

The addresses are generally represented in hex. This diagram shows them in integer just to show you that the elements are stored in contiguos locations, so that you can understand that the address difference between each element is equal to the size of one element(int size 4). For better understanding see the program below.

Actual memory representation of a 2D array

Associative Array

- An **associative array** is an abstract data type that stores a collection of (key, value) pairs and supports the operations of insertion, deletion, and lookup based on keys, typically implemented using hash tables or balanced search trees.

Cpp : hash map

main.cpp

```cpp
#include <string>
using namespace std;
int main() {
    // สร้าง Hash Table (Associative Array)
    unordered_map<string, int> scores;

    unordered_map<int , string > std;

    // ✅ Insert (เพิ่มข้อมูล)
    std[6871000] = "Alice";
    std[6871001] = "Bob";
    std[6871002] = "Tom";

    // ✅ Insert (เพิ่มข้อมูล)
    scores["Alice"] = 90;
    scores["Bob"]   = 85;
    scores["Tom"]   = 78;

    // ✅ Access (เข้าถึงข้อมูลด้วย key)
    cout << "Alice score = " << scores["Alice"] << endl;
    cout << "Alice score = " << scores[ std[6871000] ] << endl;

    // ✅ Update (แก้ไขค่า)
    scores["Tom"] = 88;

    // ✅ Iterate (วนลูปดูข้อมูลทั้งหมด)
    for (auto pair : std) {
        cout << "id=" << pair.first  << " ,name = " << pair.second;
        cout << " ,score = " << scores[pair.second] << endl;
    }

    return 0;
}
```

Associative Arrays (user defined index)

- An unordered collection of data elements indexed by an equal number of values (*keys*)
  - User defined keys must be stored
- Perl:

```
%hi_temps = ("Mon" => 77, "Tue" => 79,
  "Wed" => 65, …);
$hi_temps{"Wed"} = 83;
```

  - `$` begins the name of a scalar variable

```
delete $hi_temps{"Tue"};
```

  - Elements can be removed with `delete`

```perl

# https://www.tutorialspoint.com/execute_perl_online.php
%ages = ( 12=>51,'Bill Clinton'=>53, 'Hillary'=>51, 'Socks'=>'27 in cat years');
$ages{'Hillary'} = 52;
print $ages{'Hillary'}; print "\n";
print $ages{'Bill Clinton'};   print "\n";
print $ages{12}; print "\n";

@people=('Bill Clinton','Hillary','Socks');
print $people[1];       print "\n";

```

```
52
53
51
Hillary
```

Date type of Key can be any type

main.php

```php
<?php
/* First method to create an associate array. */
$student_one = ["Maths"=>95, "Physics"=>90,
                "Chemistry"=>96, "English"=>93,
                "Computer"=>98];

/* Second method to create an associate array. */
$student_two["Maths"] = 95;
$student_two["Physics"] = 90;
$student_two["Chemistry"] = 96;
$student_two["English"] = 93;
$student_two["Computer"] = 98;

//unset($student_two["Physics"]);    // ลบ Physics
//unset($student_two["Chemistry"]);  // ลบ Computer

/* Accessing the elements directly */
echo "Marks for student one is:\n";
echo "Maths:" . $student_two["Maths"], "\n";
echo "Physics:" . $student_two["Physics"], "\n";
echo "Chemistry:" . $student_two["Chemistry"], "\n";
echo "English:" . $student_one["English"], "\n";
echo "Computer:" . $student_one["Computer"], "\n";
?>
```

C# using Dictionary build in class

```
1  using System;
2  using System.Collections.Generic;
3
4  class GFG {
5
6          // Main Method
7      static public void Main () {
8
9          Dictionary<int, string> My_dict1 =
10                         new Dictionary<int, string>();
11                     My_dict1.Add(1123, "Welcome");
12           My_dict1.Add(1124, "to");
13           My_dict1.Add(1125, "GeeksforGeeks");
14
15           My_dict1[1124] = "abcd";
16           Console.WriteLine("{0} and {1}", 1124, My_dict1[1124]);
17           Console.WriteLine("{0} and {1}", 1125, My_dict1[1125]);
18           Console.WriteLine();
19
20         Dictionary<string, string> My_dict2 =
21               new Dictionary<string, string>(){
22                                     {"a.1", "Dog"},
23                                     {"a.2", "Cat"},
24                                     {"a.3", "Pig"} };
25
26           Console.WriteLine("{0} and {1}", "a.2", My_dict2["a.2"]);
27
28           foreach(KeyValuePair<string, string> ele2 in My_dict2)
29           {
30               Console.WriteLine("{0} and {1}", ele2.Key, ele2.Value);
31           }
32     }
33 }
```

```
1124 and abcd
1125 and GeeksforGeeks

a.2 and Cat
a.1 and Dog
a.2 and Cat
a.3 and Pig
```

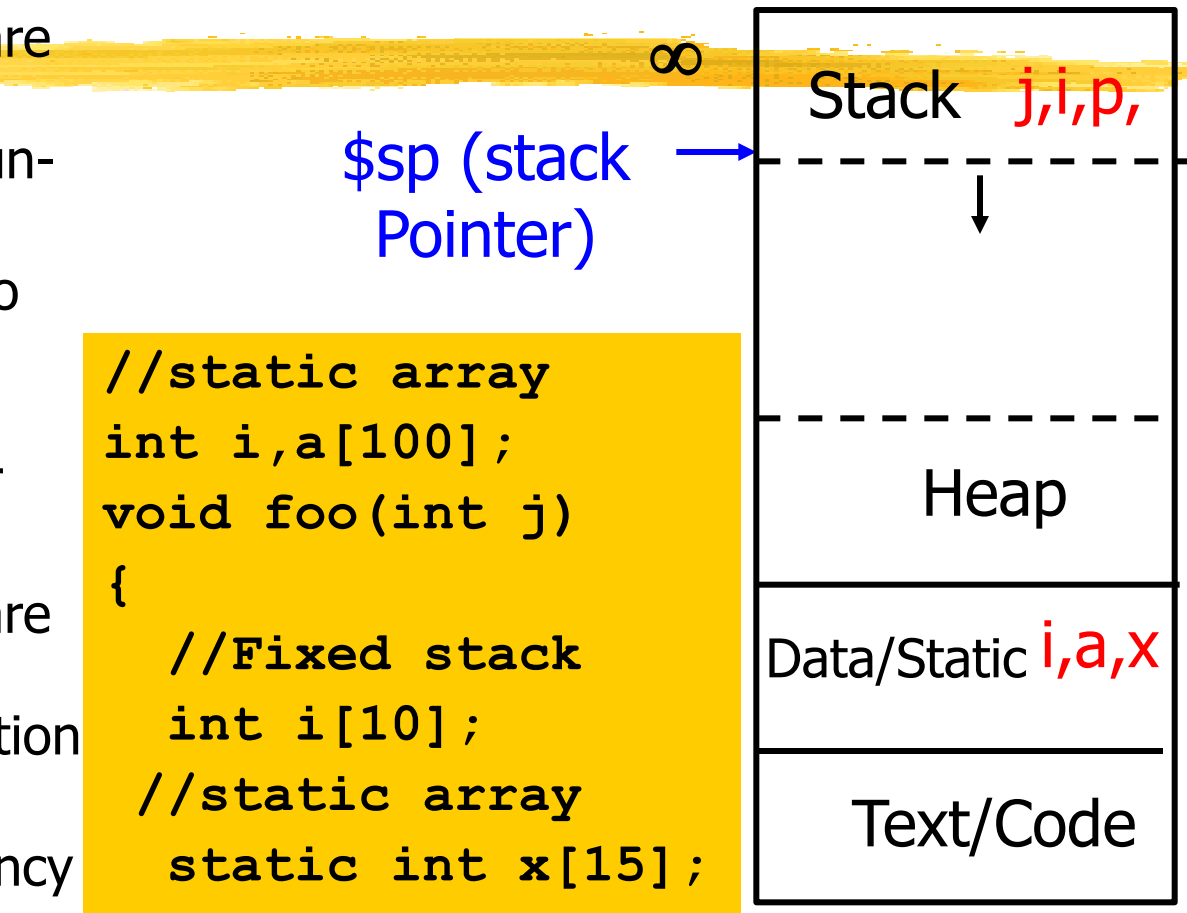
Categories of Arrays (memory management for array)

- *Static array (global &static variable)*
- *Fixed stack-dynamic (local variable)*
- *Stack-dynamic*
- *Fixed heap-dynamic (malloc,new)*
- *Heap-dynamic*

Categories of Arrays

- *Static array (global & static variable)*: subscript ranges are statically bound and storage allocation is static (before run-time)
  - Advantage: efficiency (no dynamic allocation)
  - C and C++ arrays that include `static` modifier
- *Fixed stack-dynamic (local variable)*: subscript ranges are statically bound, but the allocation is done at declaration time during execution
  - Advantage: space efficiency
  - C and C++ arrays without `static` modifier

```
//static array
int i,a[100];
void foo(int j)
{
  //Fixed stack
  int i[10];
 //static array
  static int x[15];
}
```



Categories of Arrays (cont.)

- *Stack-dynamic*: subscript ranges and storage allocation are dynamically bound at elaboration time and remain fixed during variable lifetime
  - Advantage: flexibility (the size of an array need not be known until the array is to be used), e.g., Ada

```
Get(List_Len); // input array size
declare
 List: array (1..List_Len) of Integer;
 begin
        ...
 End // List array deallocated
```

C/C++ 14 support stack dynamic

Fixed stack-dynamic vs Stack-dynamic

main.c

```c
//-----------------------
#include <stdio.h>
void  creatLocalArray(int n)
{
    int X[n];

}
int main(){
    creatLocalArray(100);
    return 0;
}

```

TurboC vs C/C++

https://www.onlinegdb.com/online_c++_compiler

Categories of Arrays (cont.)

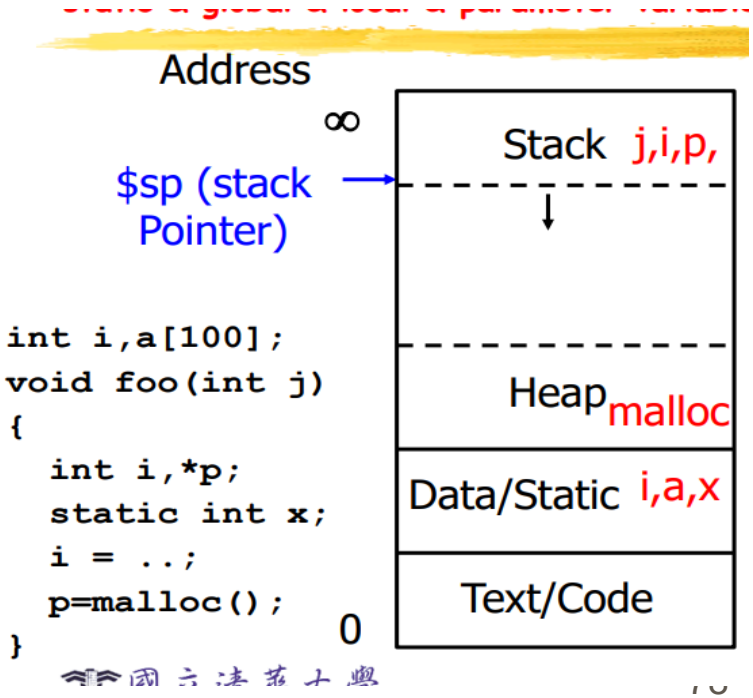
Fixed heap-dynamic: storage binding is dynamic but fixed after allocation, allocated from heap

- Java

```
String array[] = new String[] { "Toyota", "Mercedes};
```

- C#

```
int[] array1 = new int[] { 1, 3, 5, 7, 9 };
string[] weekDays = new string[] { "Sun", "Mon“ };
```



Pointer Data Type

2 First High-Level Pointers – ALGOL 58 & ALGOL 60 (Late 1950s–1960s)

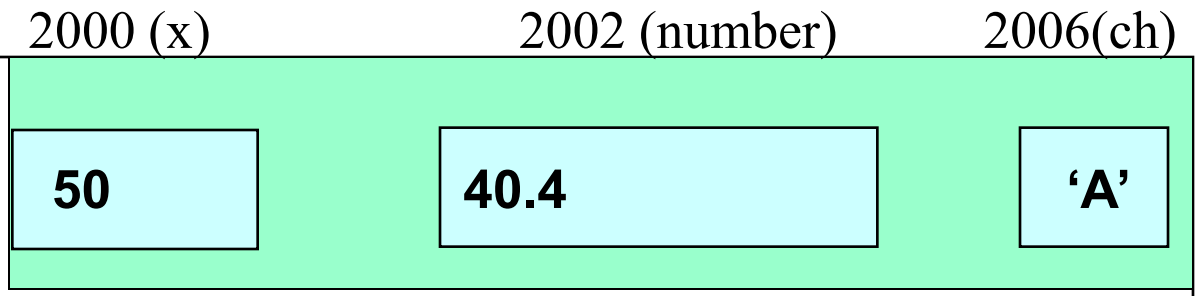
- Introduced the concept of:
  - References
  - Dynamic memory
- Enabled:
  - Recursive data structures

- Definition: **Pointer Data Type**
  - A pointer is a variable that contains the **address of a memory location** as its value, enabling **indirect access** to data stored at that location.

```
int x = 10,y=20;

int *p = &x;     // p stores address of x
printf("%d", *p);  // prints 10

p = y;               // p stores address of x
printf("%d", *p);  // prints 20
```

Pointer and Reference Types (C/C++)

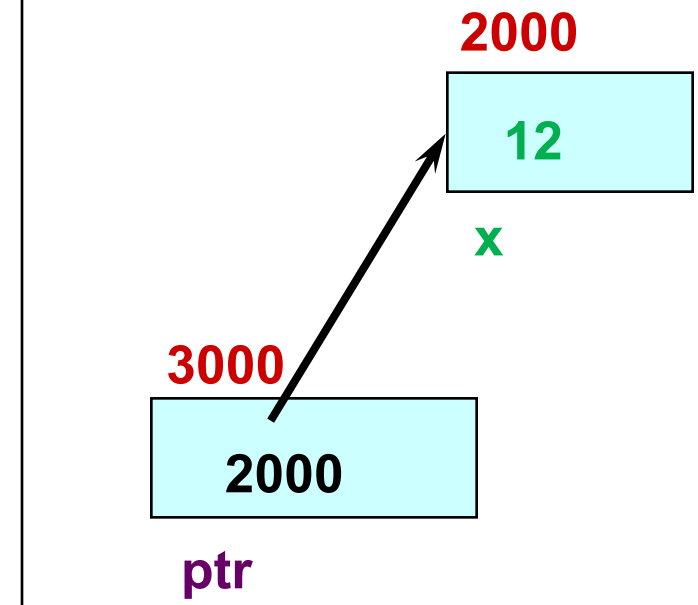
- c/c++ supports "operator" that is able to gets a memory address of variable , the address-of operator &




```
int     x;
float   number;
char    ch;

cout << "Address of x is " << &x << endl;

cout << "Address of number is " << &number << endl;

cout << "Address of ch is " << &ch << endl;
```

*: dereference operator
(การอ้างอิงค่าที่ข้อมูลในตัวแปร pointer ชี้)

```
int  x;
x = 12;

int*  ptr;
ptr = &x;

cout  <<  *ptr;
```



NOTE: The value pointed to by ptr is denoted by *ptr

Design Issues of Pointers (C/C++,Go)

- Design Operator
  - Address Operator
  - Dereference operator
  - Pointer arithmetic
  - Type restricted
- dynamic storage management
  - Allocate /Deallocate memory operator

Design Operator

- Address Operator
- Dereference operator
- Pointer arithmetic
- Type restricted

Feature	C	Rust	Go
Pointers Type restricted	✅ No	✅ Controlled	✅ Restricted
Pointer arithmetic	✅ Yes	❌	❌ No
Dereference operator	*	*	*
Address-of operator	&	&	&

Go Vs C Pointers :

dereference types & arithmetic operator

```cpp
#include <iostream>
using namespace std;

int main (void){
  int  a[] = {1,2,3};
  int *b = &a[0];
  for(int i = 0 ; i < 4 ; i++) {
        cout << i << *b << endl;
       b++;
       *b = 0;
  }
}
```

```
01
10
20
30
*** stack smashing detected ***: terminated
```

main.go

```go
package main
import "fmt"

func main() {
  a := [3] int {1,2,3};
  b := &a;  //b is a pointer to the entire array [3]int

  for i := 0 ; i < 4 ; i++ {
      fmt.Println(i,*b)
//      b++;
//    *b = 0;
  }
}
```

https://www.programiz.com/golang/online-compiler/

array ref.

Element ref.

Go with unsafe memory access

main.go

```go
package main

import (
    "fmt"
    "unsafe"
)

func main() {
    a := [3]int{1, 2, 3}
    b := unsafe.Pointer(&a[0]) // pointer to first element

    size := unsafe.Sizeof(a[0])

    for i := 0; i < len(a); i++ {
        fmt.Println(i, *( (*int)(b) ) )
        b = unsafe.Pointer(uintptr(b) + size)
    }
}
```

Rust : mutable pointer

main.rs

```rust
fn main() {
    let mut a = [1, 2, 3];              // array must be mutable
    let b: &mut [i32; 3] = &mut a;      // mutable reference to entire array

    for i in 0..3 {
        println!("{} {:?}", i, b[i]);
        b[i] = 0;                       // now legal
    }

    println!("{:?}", b);  // [0, 0, 0]
    println!("{:?}", a);  // [0, 0, 0]
}
```

Rust : mutable pointer

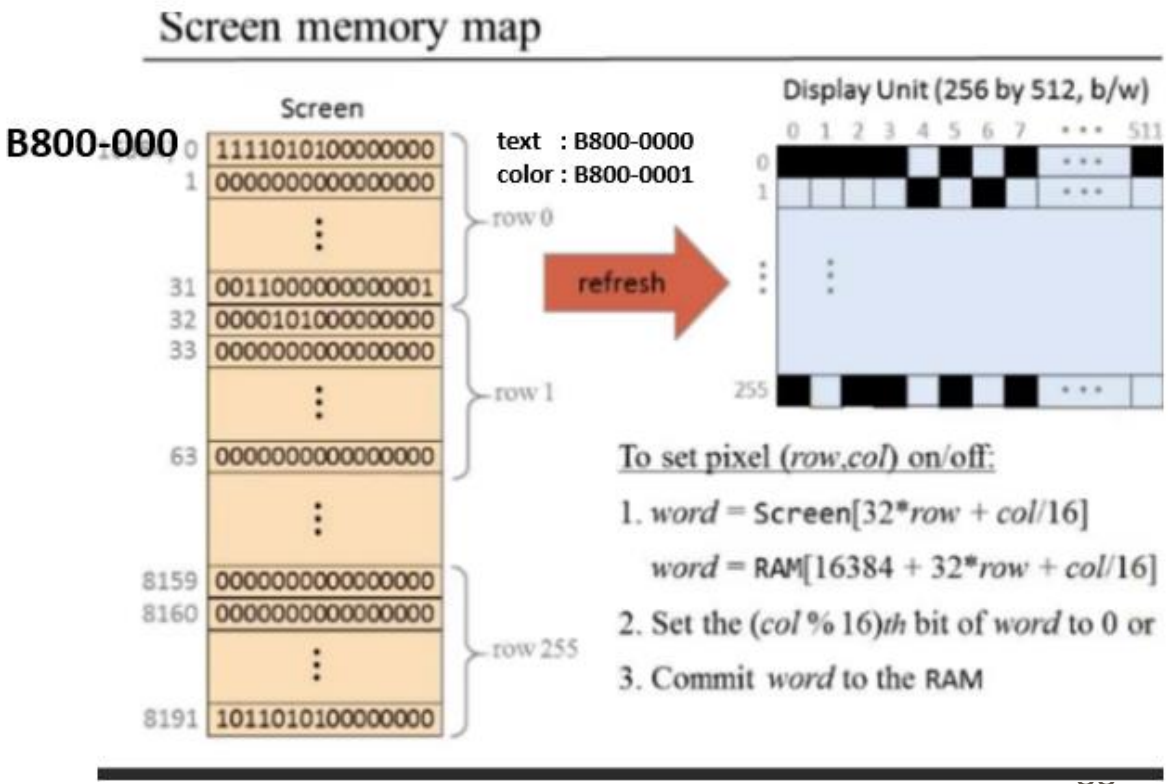
main.rs

```rust
fn main() {
    let mut a = [1, 2, 3];

    let mut b = a.as_mut_ptr();   // same as: int *b = &a[0]

    println!("{:?}", a.as_mut_ptr());
    println!("{:?}", b);

    unsafe {
        for i in 0..3 {   // ⚠ out-of-bounds bug preserved
            println!("{} {}", i, *b);
            b = b.add(1);   // b++
            *b = 0;         // *b = 0;
        }
    }

    println!("{:?}", b);  // [0, 0, 0]
    println!("{:?}", a);  // [0, 0, 0]

}
```

main.c

```c
int main() {
    char far* video_memory = (char far*)0xB8000000;
    char *msg = "0123456789ABCDEFFGH";

    while(*msg!=0)
    {
        *video_memory = *msg;
        video_memory++;
        *video_memory  = 0xB7;
        video_memory++;
        msg++;
    }

    return 0;
}
```

c/c++ :**unrestricted**

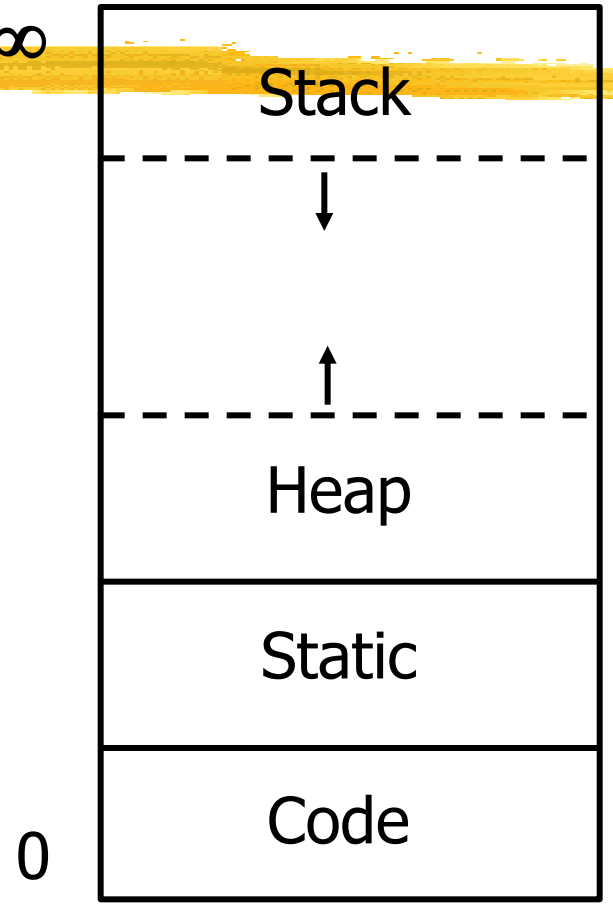


dynamic storage management

Feature	C/C++	Rust	Go
Null pointer	`NULL`	Option<T> → None	`Nil`
Operator	malloc/new free/delete	New/drop	`new` make
Garbage collection	❌ No	Safe -> borrow concept Unsafe->no	✅ Yes

Heap Management

- Two approaches
  - Explicit management : C/C++ (malloc/new,free/delete)
  - Implicit management : Java/C#/GO
    - Garbage collection (*lazy*): reclamation occurs when the list of variable space becomes empty



Explicit management

- *In C*, functions such as malloc() are used to dynamically allocate memory from the **Heap**.
- *In C++*, this is accomplished using the **new/malloc** and **delete/free** operators
- **new/malloc** is used to allocate memory during execution time
  - returns a pointer to the address where the object is to be stored
  - always returns a pointer to the type that follows the **new /malloc**

Allocation Examples

```
p1 = malloc(4)
```

```
p2 = malloc(5)
```

```
p3 = malloc(6)
```

```
free(p2)
```

```
p4 = malloc(2)
```

Problems of dynamic memory allocation

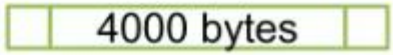
- Dangling pointers (dangerous)
  - A pointer points to a heap-dynamic variable that has been deallocated
    → has pointer but no storage
  - What happen when deferencing a dangling pointer?
- Lost heap-dynamic variable
  - An allocated heap-dynamic variable that is no longer accessible to the user program (often called *garbage*)
    → has storage but no pointer
    - The process of losing heap-dynamic variables is called *memory leakage*

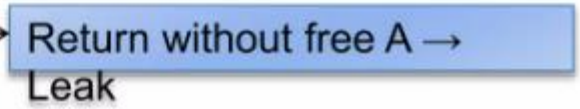
What does memory leaking mean?

- Definition:
  - Particular type of unused memory, unable to release
- Common:
  - Refer to any unwanted increase in memory usage
- Different from memory fragmentation

* for heap memory

```
void Leak()
{
    int *A = new int[1000];
    // some code here
    // ...
    // without delete A
    //
    return;
}
```





- Allocation a series, delete only one unit

```
void leak()
{
    int **list = new int*[10];
    for (int i = 0; i < 10; i++)
    {
        list[i] = new int;
    }

    delete list;
    return;
}
```

Leak !

Solution

```
for (int i = 0; i < 10; i++)
{
    delete list[i];
}
delete list;
```

```
char* GenString()
{
    char *a = new char[10];
    a[9] = '\0';
    return a;
}

void Leak()
{
    char *str = GenString();
    printf("%s\n", str);
    printf("%s\n", GenString());
}
```

Leak memory when using pointer-return-type

Solution

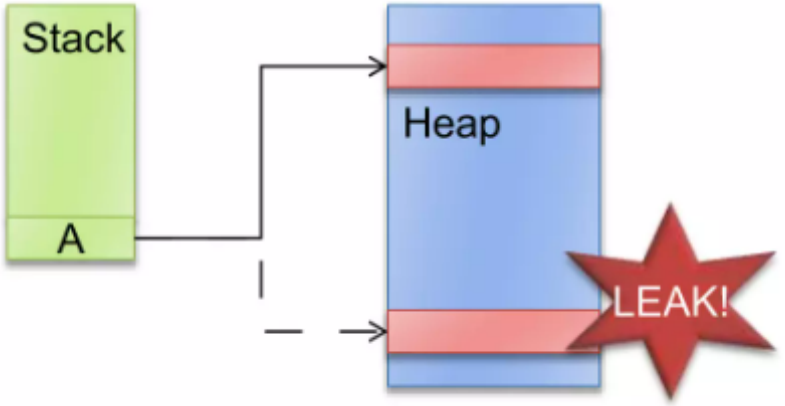
```
delete []str;
```

Avoid to call directly GenString()

Leak memory because of misunderstanding "=" operator in C++

```
#include <cstdio>

void Foo()
{
    int* a = new int[100];
    a = new int[1000];
}

void Foo1()
{
    int* a = new int[100];
    int* b = new int[1000];
    a = b;
}
```

Leak!



Garbage Collection

- Allocation and deallocation caused by assignment statements, regardless of what the variable was previously used for
  - All variables in APL; all strings and arrays in Perl and JavaScript , C# , Java
    - By nulling the reference

```
A a = new A( )
a = null; //unreferenced (system will free)
```

    - By assign ref. to another

```
A b = new A()
a = b;  //two reference
b = null; // one ref. 'a'
a = null; // no ref. (system will free)
```

    - By anonymous object

```
new A() //unrefereced (system will free)
```

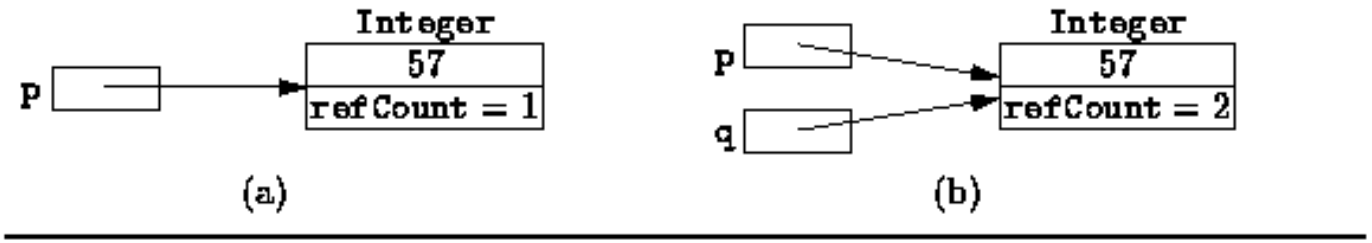


Figure: Objects with reference counters.

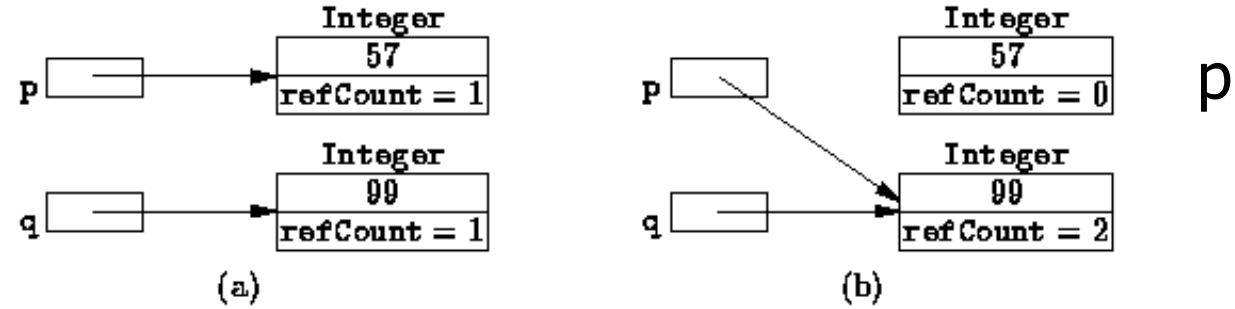
Now consider the following sequence of statements:

```
Object p = new Integer (57);
Object q = p;
```

For example suppose p and q are initialized as follows:

```
Object p = new Integer (57);
Object q = new Integer (99);
```

As shown in Figure (a), two `Integer` objects are created, each with a reference count of one. Now, si
assignment, both p and q refer to the same object--its reference count is two. And the reference count o



p=q

Figure: Reference counts before and after the assignment `p = q`.

main.go

```go
package main

import (
    "fmt"
    "runtime"
    "time"
)

type Data struct {
    id int
}

func main() {
    deleted := false

    d := &Data{id: 101}

    runtime.SetFinalizer(d, func(x *Data) {
        fmt.Println("Finalizer: Object deleted, id =", x.id)
        deleted = true
    })

    fmt.Println("Object created")

    // ตัด reference
    d = nil

    // Loop เช็คการลบ
    for !deleted {
        fmt.Println("Waiting for GC to delete object...")
        runtime.GC()                // บังคับให้ GC ทำงาน
        time.Sleep(500 * time.Millisecond)
    }

    fmt.Println("Confirmed: Object has been deleted by GC")
}
```

https://www.tutorialspoint.com/compile_java8_online.php

```
1
2  class A {
3  String m;
4   public A(String s) {
5       m = s;
6      System.out.println(" constructor : " +m);
7   }
8   @Override
9  protected void finalize() {
10   System.out.println(" deconstructor : "+m);
11   }
12 }
13  public class Main {
14      public static A f1(String s) {
15        return new A( "f1 : " + s);
16      }
17
18      public static void f() {
19          A a1 = new A("obj a1 ");
20          a1 = null; //By nulling the reference
21
22          A a2 = new A("obj a2 ");
23          A a22 = a2; //By assign ref. to another
24
25          A a3 = f1("object a3");
26          f1("object a4");
27          new A("object a5"); //By anonymous object
28          System.gc();
29      }
30
31      public static void main(String[] args) {
32        f();
33        //System.gc();
34      }
35 }
```

```
constructor : obj a1
constructor : obj a2
constructor : f1 : object a3
constructor : f1 : object a4
constructor : object a5
deconstructor : object a5
deconstructor : f1 : object a4
```

Try comment 28 & uncomment 33

Write c++ ?

Rust

- **Owner = ผู้เป็นเจ้าของหน่วยความจำ (memory)** มีสิทธิ:
  - ใช้งาน
  - แก้ไข
  - และ ปล่อย **(free)** หน่วยความจำ
  - ➡ หนึ่ง **memory** มี **owner** ได้เพียงคนเดียว

- Borrow = การยืมใช้ข้อมูลชั่วคราว โดยไม่เป็นเจ้าของ
  - ไม่มีสิทธิ free
  - ใช้ได้เฉพาะช่วงเวลาที่ owner ยังอยู่
  - หมดเวลายืม ⟶ ใช้ต่อไม่ได้

main.rs

```rust

fn main() {
    //allocat string to "S"
    let s = String::from("hello");
    println!("{}", s);
} // <- s ออกจาก scope, memory ถูก free ทันที

```

main.rs Ctrl+S

```rust
//Take Ownership (Move) :
//change ownership from Main to take
fn take(s: String) {

    println!("{}", s);
    //s is free here
}

fn main() {
    let s = String::from("hello");
    take(s);
    // println!("{}", s); ✗ compile error
}

```

Take owner

main.rs

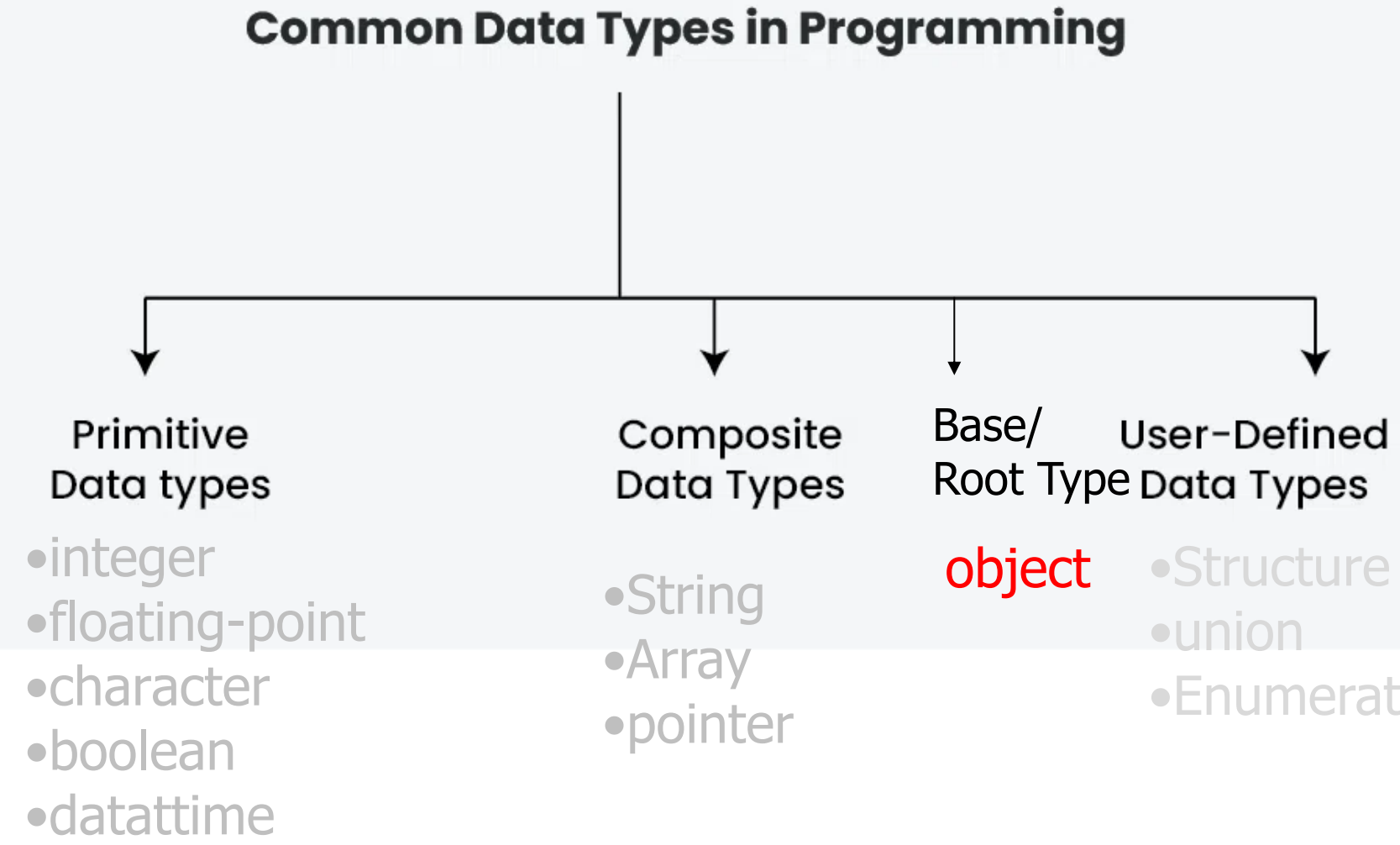
```rust
//brrow
fn read(s: &String) {
    println!("{}", s);
    //cannot update
}

fn main() {
    let s = String::from("hello");
    read(&s);
    println!("{}", s); // ✓ ยังใช้ได้
}

```

borrow

main.rs

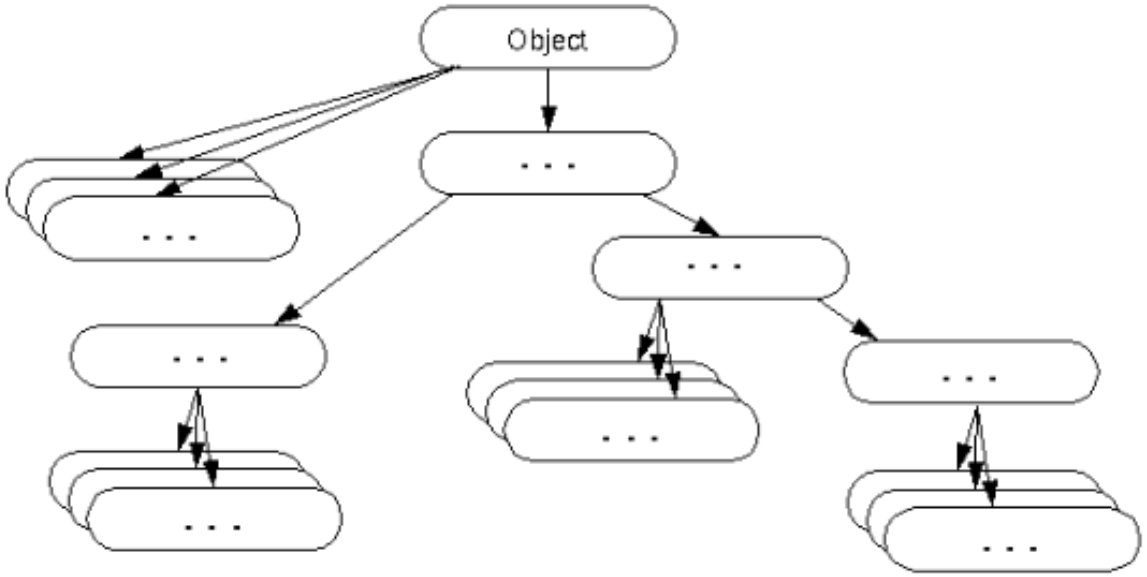
```rust
fn modify(s: &mut String) {
    s.push_str(" world");
    //can update
    println!("{}", s);
}

fn main() {
    let mut s = String::from("hello");
    modify(&mut s);
      drop(s); //force free
//    println!("{}", s);
}

```

Common Data Types in Programming Lang.



Object type in C# , java & Garbage Collection

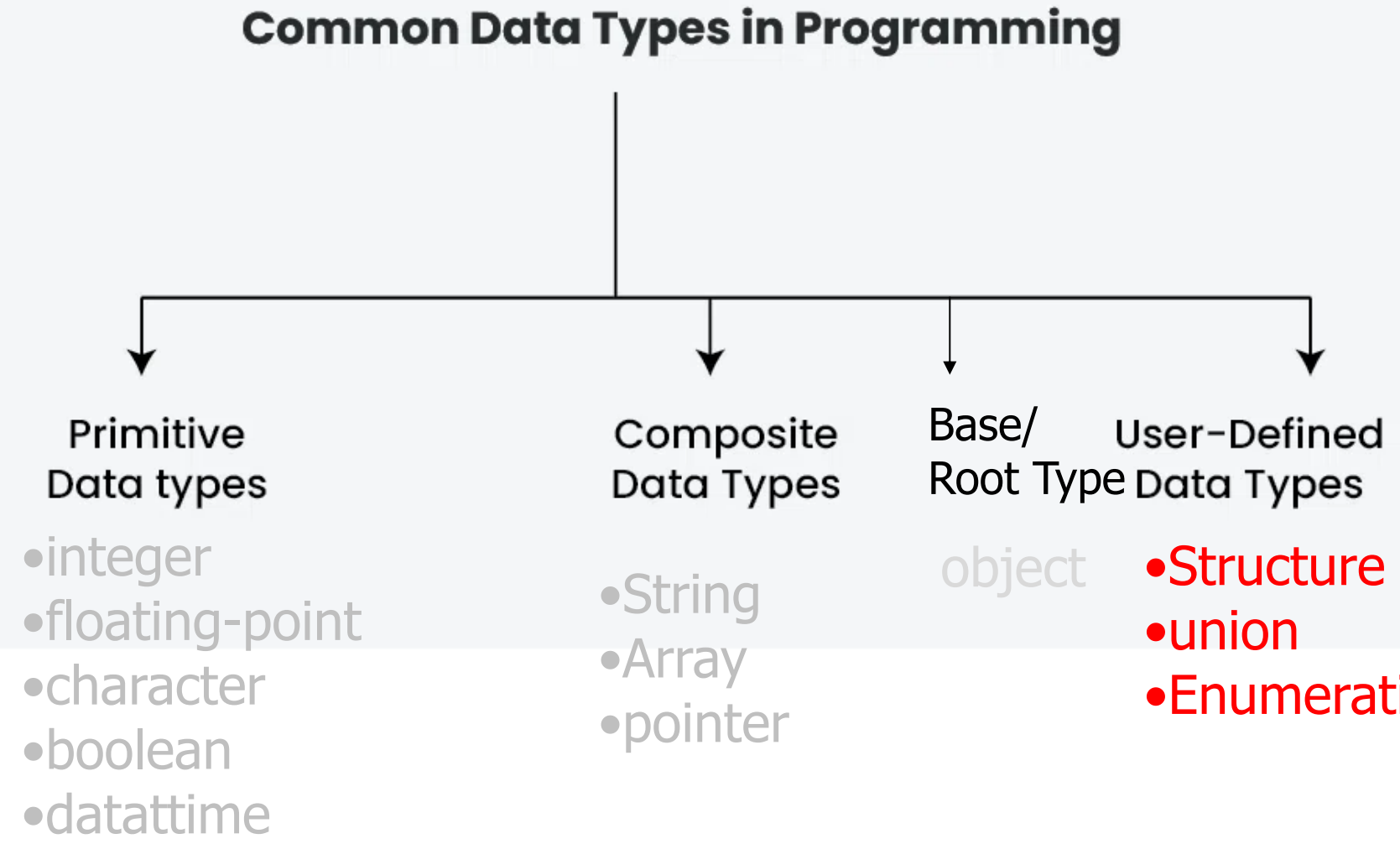
- The **Object class** is the parent class of all the classes in java/c# by default. (object)



```
mono /tmp/NH4k7LxaFH.exe
Type: System.Int32 Value: 1
Type: System.Double Value: 3.14
Type: System.String Value: Hello World
Type: ObjectType.X Value: ObjectType.X
```

```
4  using System;
5  namespace ObjectType
6  {
7      class X {
8              int x;
9              int y;
10             public X(int _x,int _y) { x = _x; y=_y;}
11     };
12     class Program
13     {
14
15         static void Main()
16         {
17             object[] crazyArr = new object[4] { 1, 3.14, "Hello World" ,new X(10,20) };
18
19             foreach(object e in crazyArr)
20             {
21                 Console.WriteLine("Type: {0} Value: {1}", e.GetType(), e.ToString());
22             }
23
24             Console.ReadLine();
25         }
26     }
27 }
```

Common Data Types in Programming Lang.



User-Defined Ordinal Types (Enumeration)

- Ordinal type: range of possible values can be easily associated with positive integers
- Two common user-defined ordinal types
  - **Enumeration**: All possible values, which are named constants, are provided or enumerated in the definition, e.g., C# example
    ```
    enum days {mon, tue, wed, thu, fri, sat, sun};
    ```
  - **Subrange**: An ordered contiguous subsequence of an ordinal type, e.g., `12..18` is a subrange of integer type

Enumeration Types

- A common representation is to treat the values of an enumeration as small integers
  - May even be exposed to the programmer, as is in C:

```
enum coin {penny=1, nickel=5, dime=10, quarter=25};
enum escapes {BELL='\a', BACKSPACE='\b', TAB='\t',
              NEWLINE='\n', VTAB='\v', RETURN='\r' };
```

- If integer nature of representation is exposed, may allow some or all integer operations:

Pascal:
```
for C := red to blue do P(C)
```
C:
```
int x = penny + nickel + dime;
```

Evaluation of Enumerated Type

- Aid to readability, e.g., no need to code a color as a number
- Aid to reliability, e.g., compiler can check:
  - Operations (don't allow colors to be added)
  - No enumeration variable can be assigned a value outside its defined range
  - Ada, C#, and Java 5.0 ,C++(version>11)

C++11 there are "strongly typed enums" now, with a syntax very similar to that of C#:
https://stackoverflow.com/questions/19267954/what-is-the-difference-between-enum-keyword-in-c-sharp-and-c

User-Defined Ordinal Types

main.cpp

```
1
2  #include <stdio.h>
3
4  void showDirct(int d)
5  {
6      switch(d){
7          case 1:
8          printf("\nWe are headed towards North.: %d",d);
9          break;
10         case 2:
11         printf("\nWe are headed towards East.: %d",d);
12         break;
13         case 3:
14         printf("\nWe are headed towards West.: %d",d);
15         break;
16         case 4:
17         printf("\nWe are headed towards South: %d",d);
18         break;
19     }
20
21 }
22 int main(){
23     showDirct(1);
24     showDirct(2);
25     showDirct(2);
26     return 0;
27 }
```

```
We are headed towards North.: 1
We are headed towards East.: 2
We are headed towards East.: 2
```

```
2  #include <stdio.h>
3  enum directions{North=1, East, West, South};
4  void showDirct(directions d)
5  {
6      switch(d){
7          case North:
8          printf("\nWe are headed towards North.");
9          break;
10         case East:
11         printf("\nWe are headed towards East.");
12         break;
13         case West:
14         printf("\nWe are headed towards West.");
15         break;
16         case South:
17         printf("\nWe are headed towards South");
18         break;
19     }
20
21 }
22 int main(){
23     showDirct(North);
24     showDirct(East);
25     directions d = (directions)2;
26     showDirct( d);
27     //directions e = North+East;
28     return 0;
29 }
```

Compare the two codes

Main.cs

```
1  using System;
2  public class HelloWorld
3  {
4    enum directions{North=1, East, West, South};
5     static void showDirct(directions d)
6      {
7          switch(d){
8              case directions.North:
9              Console.WriteLine("We are headed towards North. {0} ({1})",d,(int)d);
10             break;
11             case directions.East:
12             Console.WriteLine("We are headed towards East.{0} ({1})",d,(int)d);
13             break;
14             case directions.West:
15             Console.WriteLine("We are headed towards West.{0} ({1})",d,(int)d);
16             break;
17             case directions.South:
18             Console.WriteLine("We are headed towards South. {0} ({1})",d,(int)d);
19             break;
20         }
21     }
22     public static void Main(string[] args)
23     {
24         showDirct(directions.North);
25         showDirct(directions.East);
26         directions d = (directions)2;
27         showDirct( d);
28     }
29 }
```

Output

```
mono /tmp/6pso3mwfvS.exe
We are headed towards North. North (1)
We are headed towards East.East (2)
We are headed towards East.East (2)
```

Design Issue

- Can define scope & accessibility
- Data type of "enum" member
  - C/C++ it supports only an integer type
  - Data type of "enum" member
- "function(method)" can be member of "enum"
- Operator :

  assignment '='
- Can define "Subrange Type"

Enumeration declaration stm & access property

- Can define scope & accessibility
- Data type of "enum" member
  - C/C++ it supports only an integer type
  - Data type of "enum" member
- "function(method)" can be member of "enum"

Problem

- ระบบต้องการจัดเก็บข้อมูล
  - อาจารย์ (teacher) :
    - ชื่อ
    - อายุ
    - มีสถานะ active, inactive และ resigned
  - นักเรียน(student):
    - ชื่อ
    - เพศ
    - มีสถานะ active, inactive และ graduated

```
using System;

class Teacher
{
    public enum TeacherStatus
    {
        Active,
        Inactive,
        Resigned
    }

    public string Name { get; set; }
    public int Age { get; set; }
    public TeacherStatus Status { get; set; }

    public Teacher(string name, int age,
         TeacherStatus status)
    {
        Name = name;
        Age = age;
        Status = status;
    }

    public void Display()
    {
        Console.Write("Teacher: "+ Name);
        Console.Write(",Age:",Age);
        Console.WriteLine("Status: ",Status);
    }
}
```

Java = C#

Class member

```
enum StudentStatus
{
    Active,
    Inactive,
    Graduated
}

class Student
{
    public string Name { get; set; }
    public string Gender { get; set; }
    public StudentStatus Status { get; set; }

    public Student(string name, string gender,
          StudentStatus status)
    {
        Name = name;
        Gender = gender;
        Status = status;
    }

    public void Display()
    {
        Console.Write("Student: ",Name);
        Console.Write("Gender: ",Gender);
        Console.WriteLine("Status: ",Status);
    }
}

class Program
{
    static void Main()
    {
        Teacher t1 = new Teacher("Somchai", 45,
                         Teacher.TeacherStatus.Active);
        Student s1 = new Student("Anan", "Male",
                         StudentStatus.Graduated);

        t1.Display();
        s1.Display();
    }
}
```

global

TeacherStatus vs StudentStatus

public : accessibility

```cpp
enum  class  StudentStatus { //c++ v11
    Active,
    Inactive,
    Graduated
};


class Teacher {

public:
 enum   TeacherStatus {
    Active,
    Inactive,
    Resigned
};

private:
    string name;
    int age;
    TeacherStatus status;

public:
    Teacher(string n, int a, TeacherStatus s)
        : name(n), age(a), status(s) {}

    void display() {
        cout << "Teacher: " << name
             << ", Age: " << age
             << ", Status: " << static_cast<int>(status)
             << endl;
    }
};
```

```cpp
class Student {
private:
    string name;
    string gender;
    StudentStatus status;
public:

    Student(string n, string g,
            StudentStatus s)
        : name(n), gender(g), status(s) {}

    void display() {
        cout << "Student: " << name
             << ", Gender: " << gender
             << ", Status: " << static_cast<int>(status)
             << endl;
    }
};

int main() {
    Teacher t1("Somchai", 45,
               Teacher::TeacherStatus::Active);
    Student s1("Anan", "Male",
               StudentStatus::Graduated);

    t1.display();
    s1.display();

    return 0;
}
```

1.ทดลองเอามาไวใน class

2.กำหนด public/private

Write code with c

Data type member

- Integer : C#,Java, C/++

```
3  enum class Colors {
4      Red = 1,
5      Green = 2,
6      Blue = 3
7  };
8
9  std::string toString(Colors color) {
10     switch (color) {
11         case Colors::Red:
12             return "Red";
13         case Colors::Green:
14             return "Green";
15         case Colors::Blue:
16             return "Blue";
17     }
18     return "undefine";
19 }
20
21 int main() {
22     Colors color1 = Colors::Red;
23     std::cout << "Color1 raw value: " << toString(color1) << std::endl;   // Output: Red
24     std::cout << "Color1 int value: " << static_cast<int>(color1) << std::endl;   // Output: 1
25     return 0;
26 }
```

Data type member: (Swift :rawdata tyoe)

```swift
enum Colors: String {
    case red = "Red"
    case green = "Green"
    case blue = "Blue"

    var intValue: Int {
        switch self {
        case .red:
            return 1
        case .green:
            return 2
        case .blue:
            return 3
        }
    }

    func printInfo() {
        print("Color: \(self.rawValue), Int Value: \(self.intValue)")
    }
}
// main
let myColor = Colors.blue
myColor.printInfo()  // Output: Color: Blue, Int Value: 3
print( "color ->" , myColor );
```

Data type member: Rust:anydata type

```rust
enum Message {
    Quit,
    Move { x: i32, y: i32 }, //structure
    Write(String),
    ChangeColor(i32, i32, i32), //variant with tuple data
}
fn process_message(msg: Message) {
    match msg {
        Message::Quit => {
            println!("Quit message received");
        }
        Message::Move { x, y } => {
            println!("Move to coordinates: ({}, {})", x, y);
        }
        Message::Write(text) => {
            println!("Write message: {}", text);
        }
        Message::ChangeColor(r, g, b) => {
            println!("Change color to RGB({}, {}, {})", r, g, b);
        }
    }
}
fn main() {
    let msg1 = Message::Quit;
    let msg2 = Message::Move { x: 10, y: 20 };
    let msg3 = Message::Write(String::from("Hello, Rust!"));
    let msg4 = Message::ChangeColor(255, 0, 0);
    process_message(msg1);
    process_message(msg2);
    process_message(msg3);
    process_message(msg4);
}
```

Subrange Types

- Ada's design (PASCAL)
  - Not new type, but rename of constrained versions

```
type Days is (mon, tue, wed, thu, fri,
  sat, sun);
subtype Weekdays is Days range mon..fri;
subtype Index is Integer range 1..100;
Day1: Days;
Day2: Weekday;
Day2 := Day1;
```

Subrange Types

- Usually, we just use the same representation for the subtype as for the supertype
  - May be with code inserted (by the compiler) to restrict assignments to subrange variables
- Subrange evaluation
  - Aid to readability: make it clear to the readers that variables of subrange can store only certain range of values
  - Reliability: assigning a value to a subrange variable that is outside specified range is detected as an error

Run | Debug | Stop | Share | Save | { } Beautify

main.pas

```
1  program EnumSubrangeExample;
2  type
3    TColor = (Red, Green, Blue, Yellow, Orange, Purple);
4    // Subrange for primary colors
5    TPrimaryColor = Red..Blue;       // Red, Green, Blue only
6    // Subrange for secondary colors
7    TSecondaryColor = Yellow..Purple;  // Yellow, Orange, Purple only
8
9  var
10   systemColor: TColor;
11   primaryColor: TPrimaryColor;
12   secondaryColor: TSecondaryColor;
13
14 begin
15   systemColor := Red;          // Allowed for full enum
16   primaryColor := Green;         // Allowed in primary subrange
17
18   secondaryColor := Yellow;     // Allowed in secondary subrange
19
20  //  primaryColor := Yellow;    // Not allowed! Compiler error
21  //  secondaryColor := Green;   // Not allowed! Compiler error
22
23   if primaryColor = Red then
24     WriteLn('primaryColor is Red')
25   else
26     WriteLn('primaryColor is not Red ->',primaryColor);
27
28   if secondaryColor = systemColor then
29     WriteLn('systemColor is ',systemColor)
30   else
31     WriteLn('secondaryColor is ' ,secondaryColor);
32 end.
33
```

Rewrite code by C/C++

Record Types

- A *record* is a possibly heterogeneous aggregate of data elements in which the individual elements are identified by names
- COBOL uses level numbers to show nested records (others use recursive definition)

```
01 EMP-REC.
   02 EMP-NAME.
      05 FIRST PIC X(20).
      05 MID   PIC X(10).
      05 LAST  PIC X(20).
   02 HOURLY-RATE PIC 99V99.
```

Structure : การอ้างข้อมูล

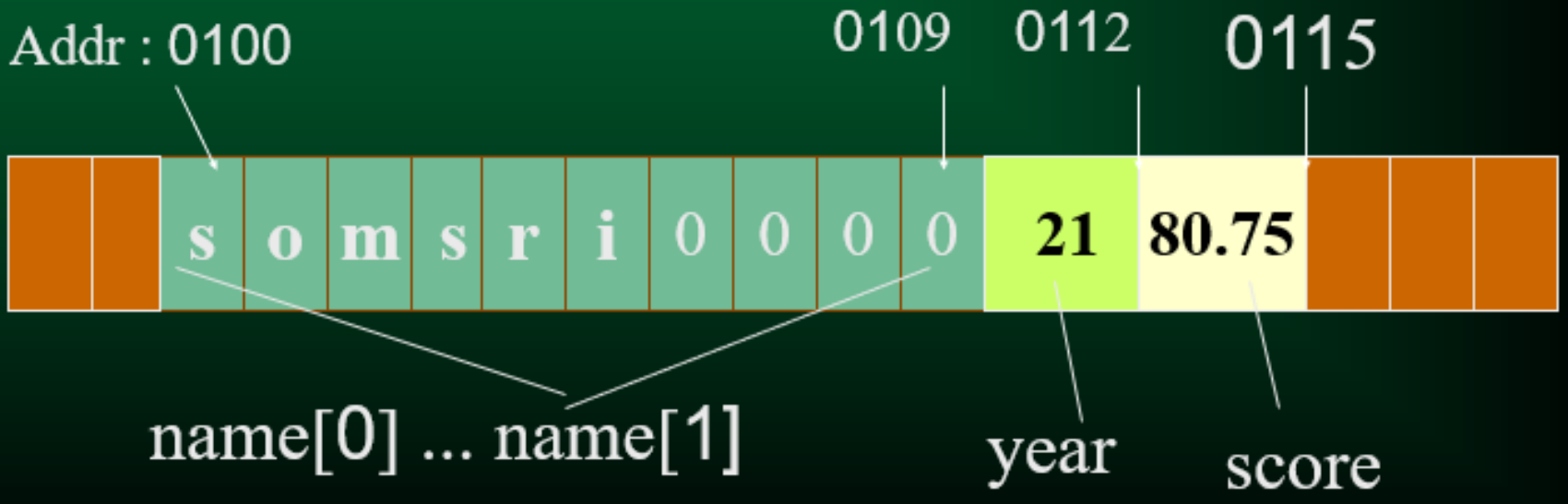
var_name . filed_name

```
struct  student_rec {
        char   name[10] ;
        int      year ;
        real     score ;
} ;
```

struct student_rec Somsri;

```
strcpy ( Somsri.name , "somsri" );
Somsri.year = 21;
Somsri.score = 80.75 ;
```

Suppose int (2 bytes)
long ,float (4 bytes)
char (1) bytes



Structure in C++ vs C

```cpp
struct Point {
//2.
//class Point {
//public:
    int x;
    int y;
    Point()   {}

    Point(int x_val, int y_val) : x(x_val), y(y_val) {}
//1.
    /*virtual*/ void print() const {
        std::cout << "Point(" << x << ", " << y << ")"  << std::endl;
    }

};

int main() {
    Point p1(10, 20);
    Point p2;
    p2 = p1;
    p1.x =100; p2.y =200;
    p1.print();          // Output: Point(10, 20)
    p2.print();          // Output: Point(10, 20)

    std::cout << "sizeof(p) ->" << sizeof(p1);

    return 0;
}
```

structure in “C” just have only data

C# struct

main.cs

```
using System;

public struct Point
//public class Point
{
    public int x;
    public int y;

    public Point(int x_val, int y_val)
    {
        x = x_val;
        y = y_val;
    }

    public void Print()
    {
        Console.WriteLine($"Point({x}, {y})");
    }
}

class Program
{
    static void Main()
    {
        Point p1 = new Point(10, 20);
        Point p2 = new Point(10, 20);
        p2 = p1;
        p1.x = 100; p1.y=200;
        p1.Print();
        p2.Print();
    }
}
```

Unions Types

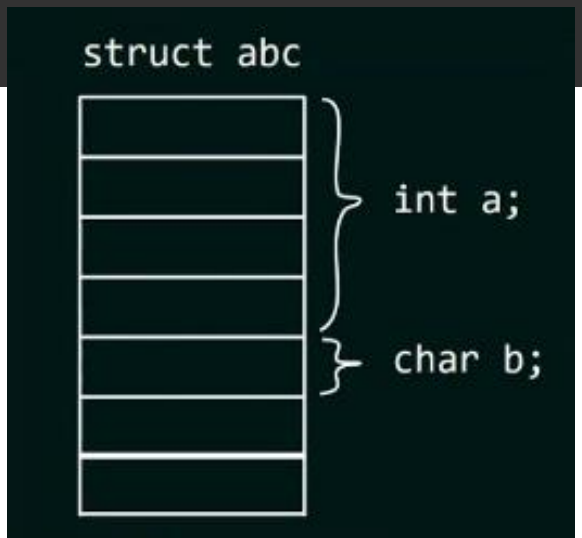
- A *union* is a type whose variables are allowed to store different types of values at different times

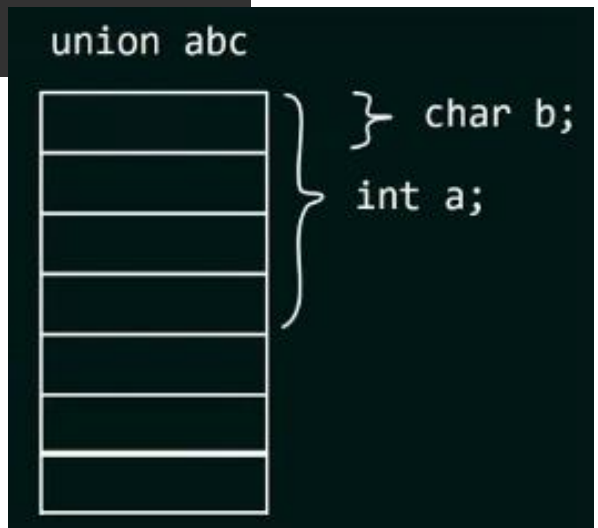
```
union time {
  long simpleDate;
  double perciseDate;} mytime;
...
printTime(mytime.perciseDate);
```

- Design issues
  - Should type checking be required?
  - Should unions be embedded in records?

```c
#include <stdio.h>
union Job {
    float salary;
    int  workerNo;
    char name[12];

} j;

int main() {
    j.salary  = 12.3;
    printf("1.Salary = %.1f\n", j.salary);
    j.workerNo = 100;
    printf("2.Salary = %.1f\n", j.salary);
    printf("3.Number of workers = %d\n", j.workerNo);
    printf("4.sizeof(j.workerNo) = %ld\n", sizeof(j.workerNo) );
    printf("5.sizeof(j.salary) = %ld\n", sizeof(j.salary) );
    printf("6.sizeof(j.name) = %ld\n", sizeof(j.name) );
    printf("7.sizeof(j) = %ld\n", sizeof(j) );
    return 0;
}
```

```
1.Salary = 12.3
2.Salary = 0.0
3.Number of workers = 100
4.sizeof(j.workerNo) = 4
5.sizeof(j.salary) = 4
6.sizeof(j.name) = 12
7.sizeof(j) = 12
```





```cpp
#include <iostream>
#include <string.h>
using namespace std;

int main()
{
    int x1=4;
    int y1=5;
    int t1;

    cout << "x1 =" << x1 << ", y1 = " << y1 <<endl;
    t1 = x1;
    x1 = y1;
    y1 = t1;
    cout << "x1 =" << x1 << ", y1 = " << y1 <<endl;

    char x2[] = "x2-data";
    char y2[] = "y2-data";
    char t2[20];
    cout << "x2 =" << x2 << ", y2 = " << y2 <<endl;
    strcpy(t2,x2);
    strcpy(x2,y2);
    strcpy(y2,t2);
    cout << "x2 =" << x2 << ", y2 = " << y2<<endl;

    return 0;
}
```

```cpp
#include <iostream>
#include <string.h>
using namespace std;
union t {
    int i;
    char s[10];
};
int main()
{
    union t tmp;
    int x1=4;
    int y1=5;

    cout << "x1 =" << x1 << ", y1 = " << y1 <<endl;
    tmp.i = x1;
    x1 = y1;
    y1 = tmp.i;
    cout << "x1 =" << x1 << ", y1 = " << y1 <<endl;

    char x2[] = "x2-data";
    char y2[] = "y2-data";
    cout << "x2 =" << x2 << ", y2 = " << y2 <<endl;
    strcpy(tmp.s,x2);
    strcpy(x2,y2);
    strcpy(y2,tmp.s);
    cout << "x2 =" << x2 << ", y2 = " << y2<<endl;

    return 0;
}
```

```
using System;
class HelloWorld {


  static void Main() {
    dynamic t;
    int x1=4;
    int y1=5;

    Console.WriteLine( "x1 = " + x1 + ", y1 = " + y1 );
    t = x1;
    x1 = y1;
    y1 = t;
    Console.WriteLine( "x1 = " + x1 + ", y1 = " + y1 );

    string x2 = "x2-data";
    string y2 = "y2-data";
    Console.WriteLine( "x2 = " + x2 + ", y2 = " + y2 );
    t = x2;
    x2 = y2;
    y2 = t;
    Console.WriteLine( "x1 = " + x2 + ", y1 = " + y2 );
  }
}
```

```c
// Define sensor types
typedef enum {
    SENSOR_TEMPERATURE,
    SENSOR_HUMIDITY,
    SENSOR_STATUS
} SensorType;

// Union to hold different kinds of sensor
typedef union {
    float temperature;    // Temperature i
    int humidity;         // Humidity in p
    char status[16];      // Status messag
} SensorData;

// Struct combining sensor type and its da
typedef struct {
    SensorType type;
    SensorData data;
} SensorReading;

void printSensorReading(const SensorReading* reading) {
    switch (reading->type) {
        case SENSOR_TEMPERATURE:
            printf("Temperature: %.2f °C\n", reading->data.temperat
            break;
        case SENSOR_HUMIDITY:
            printf("Humidity: %d %%\n", reading->data.humidity);
            break;
        case SENSOR_STATUS:
            printf("Status: %s\n", reading->data.status);
            break;
    }
}

int main() {
    SensorReading r1, r2, r3;
    // Initialize temperature sensor reading
    r1.type = SENSOR_TEMPERATURE;
    r1.data.temperature = 23.5f;

    // Initialize humidity sensor reading
    r2.type = SENSOR_HUMIDITY;
    r2.data.humidity = 65;
    // Initialize status sensor reading
    r3.type = SENSOR_STATUS;
    strcpy(r3.data.status, "OK");

    printSensorReading(&r1);
    printSensorReading(&r2);
    printSensorReading(&r3);

    return 0;
}
```

Evaluation of Unions

- Free unions are unsafe
  - Do not allow type checking
- Java and C# do not support unions
  - Reflective of growing concerns for safety in programming language

Type Conversion

- **Type Conversion** is the process of changing a value from one data type to another.
  - It can occur automatically (**implicit conversion**)
  - It can be explicitly specified by the programmer (**explicit conversion**).
- Some conversions may result in **loss of data**.

Explicit/Implicit Type conversion

main.swift

```swift
import Foundation

// int → float
let a: Int = 10
let b: Float = Float(a)

// int → string
let s1: String = String(a)

// float → int
let x: Float = 3.7
let y: Int = Int(x)   // decimal part is discarded

// string → int
//let s2: String = "123abc"
let s2: String = "123"
let z: Int = Int(s2)! // force unwrap for demo

//cannot convert boolean → int
let flag: Bool = true
let i: Int = flag ? 1 : 0

//cannot convert int → boolean
let n: Int = 0
let ok: Bool = n != 0

print(b, s1, y, z, i, ok)

```

swift

Not support implicit

C

Explicit

main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

int main() {

    int a = 10;
    float b = (float)a;          // int → float

    /* int → string : cannot to convert */
    char s1[20];
    sprintf(s1, "%d", a);

    float x = 3.7f;
    int y = (int)x;              // float → int

    /* string → int  : cannot to convert  */
    char s2[] = "123";
    int z = atoi(s2);

    bool flag = true;
    int i = (int )flag ;         // boolean → int

    int n = 0;
    bool ok = (bool)n;           // int → boolean

    printf("%f %s %d %d %d %d\n", b, s1, y, z, i, ok);

    return 0;
}
```

Implicit

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

int main() {

    int a = 10;
    float b = a;                 // implicit int → float

    /* int → string : NO implicit conversion in C */
    char s1[20];
    sprintf(s1, "%d", a);        // must use function

    float x = 3.7f;
    int y = x;                   // implicit float → int

    /* string → int : NO implicit conversion */
    char s2[] = "123";
    int z = atoi(s2);            // must use function

    bool flag = true;
    int i = flag;                // implicit boolean → int

    int n = 0;
    bool ok = n;                 // implicit int → boolean

    printf("%f %s %d %d %d %d\n", b, s1, y, z, i, ok);

    return 0;
}
```

Pascal

Explicit

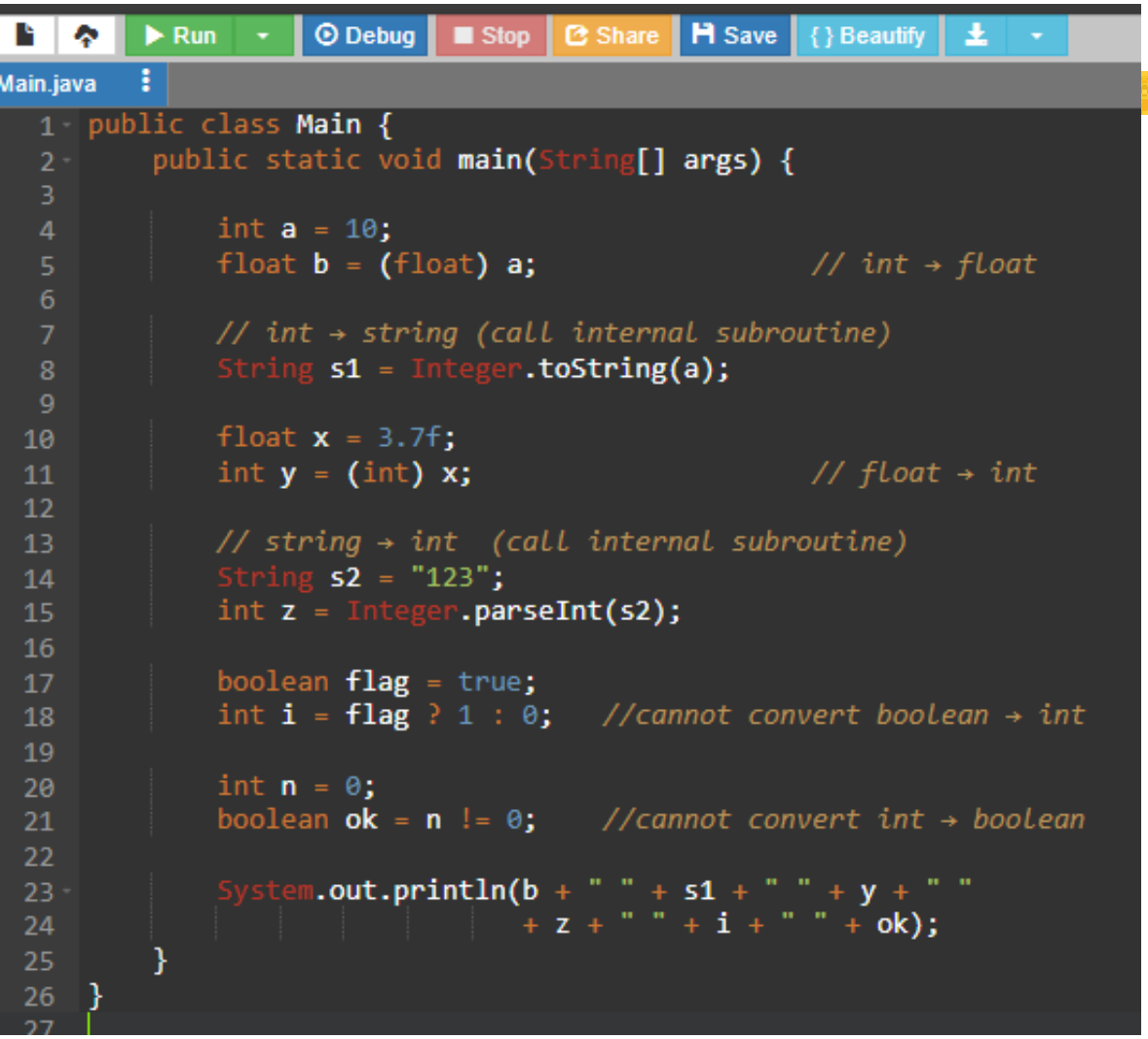
```pascal
program TypeConversionDemo;
uses
  SysUtils;
var
  a    : Integer;
  b    : Real;
  s1   : String;
  x    : Real;
  y    : Integer;
  s2   : String;
  z    : Integer;
  flag: Boolean;
  i    : Integer;
  n    : Integer;
  ok   : Boolean;

begin
  a := 10;
  b := Real(a);                    { int → float (implicit c
  { int → string }
  s1 := IntToStr(a);    //call routine

  x := 3.7;
  y := Trunc(x);                   { float → int }

  { string → int }
  s2 := '123';
  z := StrToInt(s2);    //call routine

  flag := True;
  i := Ord(flag);                  { boolean → int }

  n := 0;
  ok := boolean(n);                { int → boolean }

  Writeln(b:0:1, ' ', s1, ' ', y, ' ', z, ' ', i, ' ', ok);
end.
```

Implicit

Run | Debug | Stop | Share | Save | { } Beautify

main.pas

```pascal
program TypeConversionDemo;

var
  a    : Integer;
  b    : Real;
  s1   : String;
  x    : Real;
  y    : Integer;
  s2   : String;
  z    : Integer;
  flag: Boolean;
  i    : Integer;
  n    : Integer;
  ok   : Boolean;

begin
  a := 10;
  b := a;              { implicit Integer → Real ✅ }

  { int → string : NO implicit conversion }
  Str(a, s1);          { must use routine }

  x := 3.7;
  //y := x;            { implicit Real → Integer ⚠ (tr

  { string → int : NO implicit conversion }
  Val('123', z);       { must use routine }

  flag := True;
  //i := flag;         { implicit Boolean → Integer ✅

  n := 0;
  //ok := n;           { implicit Integer → Boolean ✅

  Writeln(b:0:1, ' ', s1, ' ', y, ' ', z, ' ', i, ' ',
end.
```

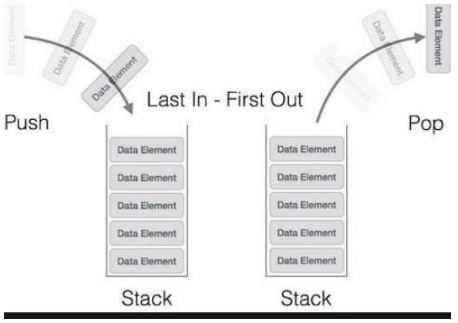
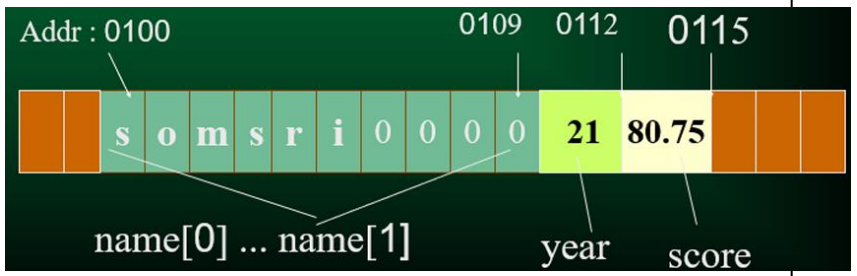


Not support implicit

END

Design datatype : built-in vs Abstract

- Data representation
- Operator
- Operator implement

datatype	structure	stack
Data declaration	`struct student_rec { char name[10] ; int year ; real score ; } ;`	
Data representation	name | year | score somsri | 21 | 80.75	
Operator	=	Push() , pop() , top()
implement		Array , linked list
domain	Field type	Stack <int>

Variable type : Why you should define variable type

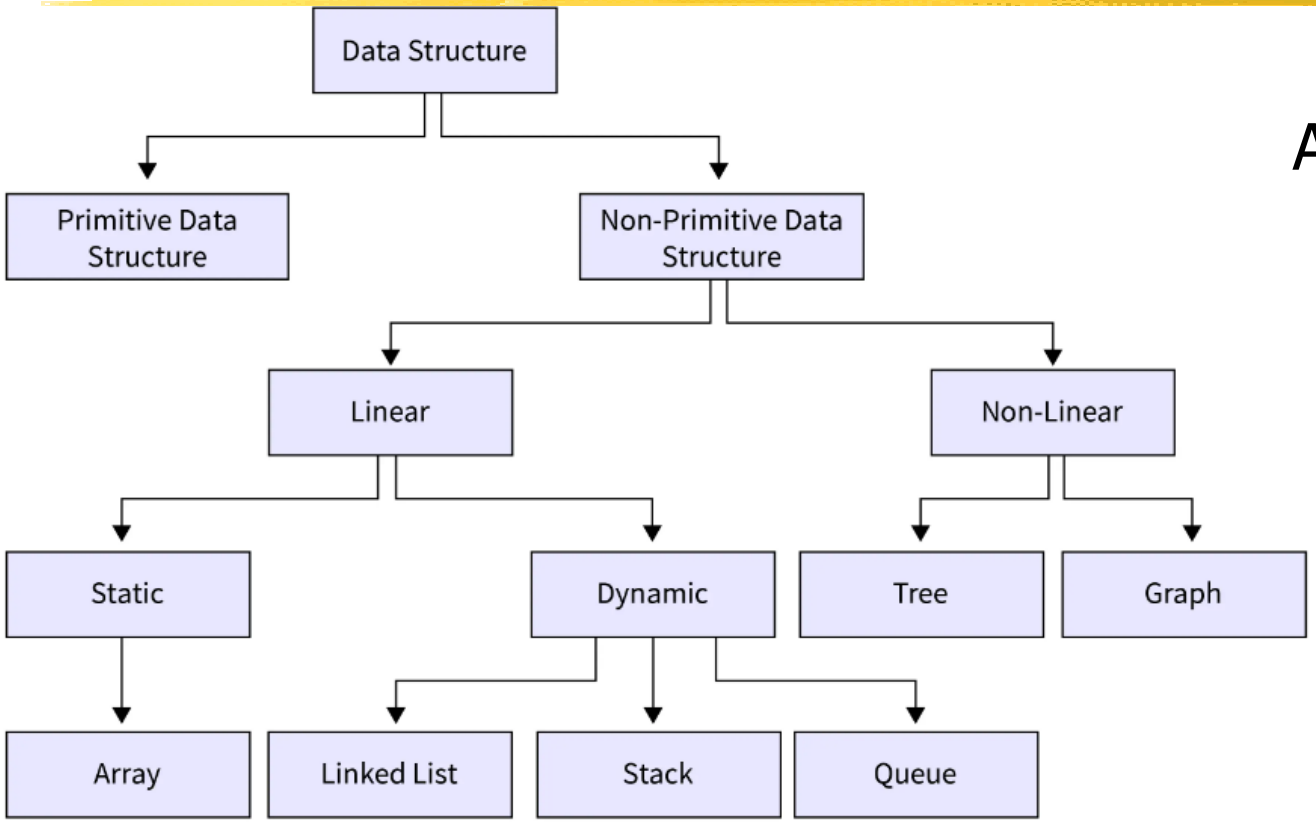
- Specify a collection of operations that can be applied to those value types

```
#include <iostream>
using namespace std;
struct X {
 int a;
};
int main() {
  int x = 5;
  int y = 3;
  struct  X a,b;
  cout << x + y;
  cout << a +b ;
  return 0;
}
```

```
prog.cpp: In function 'int main()':
prog.cpp:11:13: error: no match for 'operator+' (operand types are 'X' and 'X')
   11 |   cout << a +b ;
      |           ~ ^~
      |           |  |
      |           X  X
In file included from /usr/include/c++/9/bits/stl_algobase.h:67,
                 from /usr/include/c++/9/bits/char_traits.h:39,
                 from /usr/include/c++/9/ios:40,
                 from /usr/include/c++/9/ostream:38,
                 from /usr/include/c++/9/iostream:39,
                 from prog.cpp:1:
/usr/include/c++/9/bits/stl_iterator.h:423:5: note: candidate: 'template<class _Iterator> std:
  423 |     operator+(typename reverse_iterator<_Iterator>::difference_type __n,
      |     ^~~~~~~~
/usr/include/c++/9/bits/stl_iterator.h:423:5: note:   template argument deduction/substitution
prog.cpp:11:14: note:   'X' is not derived from 'const std::reverse_iterator<_Iterator>'
   11 |   cout << a +b ;
      |              ^
In file included from /usr/include/c++/9/bits/stl_algobase.h:67,
```

Ex. Compile error because operator "+" not support on "+" operator

There are two types of data types(data structure) in Programming Lang.



Abstract data-type

Class/object

structure

Enumeration

String , array

Int, float, char

Primitive datatype

Var in c#

Type Inference

```
namespace ConsoleApplication3
{
    class Program
    {
        public class User
        {
            public string FirstName { get; set; }
            public string LastName { get; set; }
            public int BirthYear { get; set; }
            public double salary { get;set; }
        }

        static void Main(string[] args)
        {
            var users = new List<User>()
                {
                    new User()
                    {
                        FirstName = "Terrance",
                        LastName = "Johnson",
                        BirthYear = 2005,
                        salary= 5000.0
                    },
                    new User()
                    {
                        FirstName = "John",
                        LastName = "Smith",
                        BirthYear = 1966,
                        salary= 15000.0

                    },
                    new User()
                    {
                        FirstName = "Eva",
                        LastName = "Birch",
                        BirthYear = 2002,
                        salary= 20000.0
                    }
                };
```

Var (implicit) in c#

```
45          int currentYear = 2028;
46          //Get the full combined name for people born in 1990 or later
47          var fullNames = from x in users
48                          where x.BirthYear >= 1990
49                         select new { fullname=x.FirstName,
50                                      x.LastName ,x.salary,
51                                      age = currentYear-x.BirthYear  };
52          foreach (var item in fullNames)
53          {
54              System.Console.WriteLine("fullname={0} ,salary={1},age={2}",
55                      item.fullname,item.salary, item.age);
56          }
57
58          var age1990 = from x in users
59                          where x.BirthYear >= 1990
60                          select new { x.FirstName, x.BirthYear };
61          foreach (var item in age1990)
62          {
63              System.Console.WriteLine("FirstName : {0} , BirthYear: {1}", item.FirstName, item.BirthYear);
64          }
65
66          return;
67      }
68  }
69  }
```

Heterogeneous arrays

```
1  var pets = [ {"fname":"jack",'age':60,'children':5 },
2               {"fname":"john",'degree':'phd','age':50,'children':6},
3               {"fname":"joe",'lname':'may','degree':'bachelor' }];
4
5  for(i=0 ; i < pets.length;i++)
6  {
7    console.log(pets[i].fname);
8    console.log(pets[i].age);
9  }
```

https://playcode.io/javascript

สมมุติว่า การคำนวณค่าขนส่งสินค้า มีสูตร คือ

ค่าขนส่งสินค้า (cost) = น้ำหนักของส่ง (w) x ค่าส่งต่อปอนด์ (R)

ซึ่งค่าส่งต่อปอนด์ (R) ขึ้นอยู่กับโซน (IZ)

รหัสโซน : IZ	ค่าขนส่ง/ปอนด์ : R
-2	.5
-1	.75
-0	1.05
1	1.40
2	1.70

จงเขียนโปรแกรมอ่านโซนปลายทาง (IZ) และน้ำหนักของของส่งของ (W) และพิมพ์โซนน้ำหนัก และ ค่าส่ง

```
1  {https://www.onlinegdb.com/online_pascal_compi
2  program Hello;
3   var
4          ZI : array [-2 .. 2 ] of real;
5          Zone : integer;
6          W : real;
7          R : real;
8  begin
9      ZI[-2] := 0.5;
10     ZI[-1] := 0.75;
11     ZI[0] := 1.05;
12     ZI[1] := 1.4;
13     ZI[2] := 1.7;
14
15     write( 'Zone  (-2..2) = ' ); readln(Zone);
16     write( 'Weigth  = ' ); readln(W);
17     R := ZI[Zone] *w ;
18     writeln( 'Zi (', Zone ,') = ' ,ZI[Zone] );
19     writeln( 'R  = ' ,R );
20
21 end.
```

Categories of Arrays (cont.)

- *Heap-dynamic*: binding of subscript ranges and storage is dynamic and can change
  - Advantage: flexibility (arrays can grow or shrink during execution), e.g., Perl, JavaScript

```
1  @odd = (1,3,5);
2  @even = (2, 4, 6);
3  @numbers = (@odd, @even);
4
5  print "numbers = @numbers\n";
6
7  splice@numbers, 3, 2;
8
9  print "numbers = @numbers\n";
```

```
numbers = 1 3 5 2 4 6
numbers = 1 3 5 6
```

https://www.tutorialspoint.com/execute_perl_online.php

main.cpp

```cpp
#include <iostream>
#include <stdio.h>
using namespace std;

int main()
{
    int     x=50;
    float   number=40.4;
    char    ch='A';

    printf("\nx= %d , addr x= %p",x,&x);
    printf("\nnumber= %f , addr number= %p",number,&number);
    printf("\nch= %c , addr ch= %p",ch,&ch);


    return 0;
}
```

```
x= 50 , addr x= 0x7ffc24bf5950
number= 40.400002 , addr number= 0x7ffc24bf5954
ch= A , addr ch= 0x7ffc24bf594f
```

%p : format display of pointer

What is a pointer variable?

- Pointer data type is a special kind of variable which are meant to store addresses only, instead of values(integer, float, double, char, etc).
- A pointer variable is a variable whose value is the address of a location in memory.
- To declare a pointer variable, you must specify the type of value that the pointer will point to, for example,

```
int*   ptr; // ptr will hold the address of an int

char*  q;   // q will hold the address of a char
struct employee* e; // e will hold the address of
                    //an employee struct
```

```
int x;
int *p = &x;  *q = &x ;
```

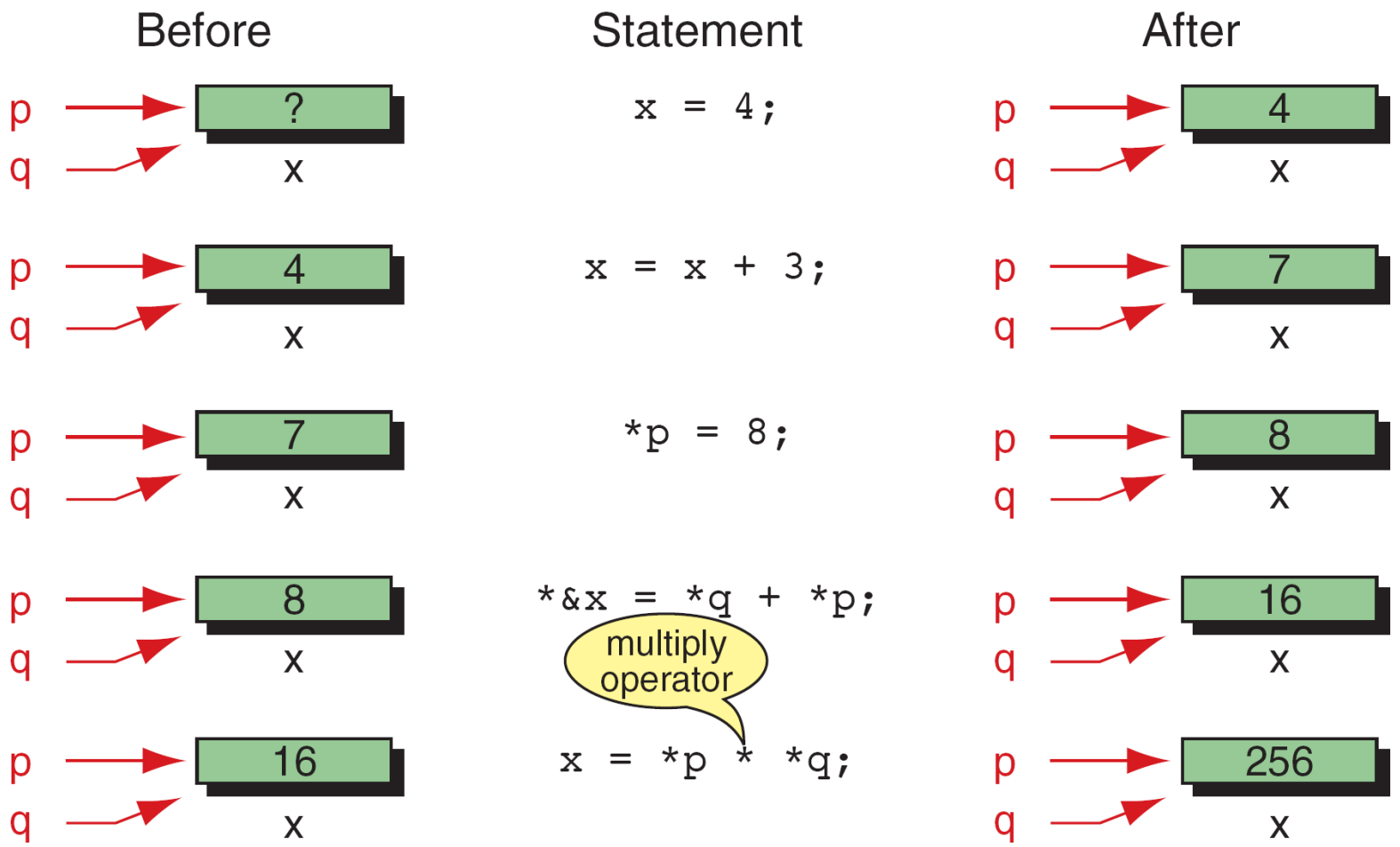


FIGURE 9-8 Accessing Variables Through Pointers

Size of memory of pointer data type

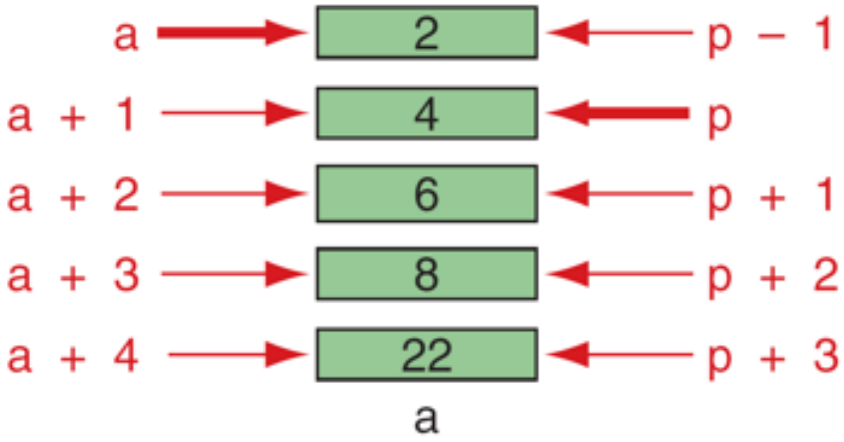
```
sizeof(i) = 4 sizeof(pi) = 8
sizeof(f) = 4 sizeof(pf) = 8
sizeof(c) = 1 sizeof(pc) = 8
sizeof(d) = 8 sizeof(pd) = 8
sizeof(e) = 20 sizeof(pe) = 8
```

```
2  #include <stdio.h>
3
4  using namespace std;
5
6  struct employee {
7      char name[16];
8      int  id;
9  };
10
11 int main()
12 {
13     int  i, *pi;
14     float  f, *pf;
15     char  c, *pc;
16     double  d, *pd;
17     struct employee e, *pe;
18
19     cout << "\n sizeof(i) = " << sizeof(i) << " sizeof(pi) = " << sizeof(pi) ;
20     cout << "\n sizeof(f) = " << sizeof(f) << " sizeof(pf) = " << sizeof(pf) ;
21     cout << "\n sizeof(c) = " << sizeof(c) << " sizeof(pc) = " << sizeof(pc) ;
22     cout << "\n sizeof(d) = " << sizeof(d) << " sizeof(pd) = " << sizeof(pd) ;
23     cout << "\n sizeof(e) = " << sizeof(e) << " sizeof(pe) = " << sizeof(pe) ;
24
25     return 0;
26 }
```

Pointer Arithmetic and Arrays

Given pointer , **p**, **p** ± **n** is a pointer to the value n elements away.

Concept: Suppose p = & a[1]



Go vs C++

Go language provides automatic garbage collection for allocating memory.	C++ language does not provide automatic garbage collection for allocating memory.
Go language contains pointers, but does not contain arithmetic pointer.	C++ language contains both pointers as well as arithmetic pointers.

Reference Variables (reference type C/C++)

Reference variable = ***alias for another variable***

- Contains the address of a variable (like a pointer)
- No need to perform any dereferencing (unlike a pointer)
- Must be initialized when it is declared

```
int x = 5;
int &z = x;            // z is another name for x
int &y ;               //Error: reference must be initialized
cout << x << endl;     -> prints 5
cout << z << endl;     -> prints 5

z = 9;                 // same as x = 9;

cout << x << endl;     -> prints 9
cout << z << endl;     -> prints 9
```

Reference Variables Example

```
#include <iostream>
using namespace std;
// Function prototypes
   (required in C++)

void p_swap(int *, int *);
void r_swap(int&, int&);

int main (void){
 int v = 5, x = 10;
 cout << v << x << endl;
 p_swap(&v,&x);
 cout << v << x << endl;
 r_swap(v,x);
 cout << v << x << endl;
 return 0;
}
```

```
void p_swap(int *a, int *b)
{
  int temp;
   temp = *a;      (2)
   *a = *b;        (3)
   *b = temp;

 }
```

```
void r_swap(int &a, int &b)
{
 int temp;
 temp = a;         (2)
 a = b;            (3)
 b = temp;
}
```

Implicit Heap-dynamic Variables

- Allocation and deallocation caused by assignment statements, regardless of what the variable was previously used for
  - All variables in APL; all strings and arrays in Perl and JavaScript , C# , Java

```
f1( ) {
    A a = new A( );  //memory will be automagical released
 }          //when allocated memory 'a' is unreferenced
```

- Advantage: flexibility
- Disadvantages:
  - Runtime overhead for maintaining dynamic attributes
  - Loss of error detection by compiler

https://www.programiz.com/java-programming/online-compiler/

```
public class MyClass {
    public static void main(String args[]) {
        int x,y;
        x = 10;
        y = x;
        System.out.println( "X="+ x );
        System.out.println( "y="+ y );
        x = 20;
        System.out.println( "X="+ x );
        System.out.println( "y="+ y );


        String[] A = {"Bob", "Jake", "Joe"};
        String[] B ;
        B = A;
        System.out.println( "A="+ A[0] );
        System.out.println( "B="+ B[0] );
        A[0] = "update";
        System.out.println( "A="+ A[0] );
        System.out.println( "B="+B[0] );


      //System.out.println("Sum of x+y = " + z);
    }
}
```

```
java -cp /tmp/bxG58mSP6s/MyClass
X=10
y=10
X=20
y=10
A=Bob
B=Bob
A=update
B=update
```

```
1  {$APPTYPE CONSOLE}
2  program Example;
3
4  type
5    TFontStyle    = (fsNormal,fsBold,fsUnderline,fsItalic);
6    TFontAngsana  = fsNormal..fsUnderline;
7    TFontArial    = fsBold..fsUnderline;
8    TFontCordia   = fsUnderline..fsItalic;
9  var
10   header : TFontAngsana;
11   body   : TFontArial;
12   footer : TFontCordia;
13 begin
14   header := TFontAngsana.fsBold;
15   writeln( header );
16   body := TFontAngsana.fsUnderline;
17   writeln( body );
18   footer := TFontCordia.fsItalic;
19   writeln( footer );
20 end.
```

main.pas

```pascal
{https://www.onlinegdb.com/online_pascal_compiler}
program Hello;
   type
       { This is an enumerated type. }
       Days = (Mon, Tue, Wed, Thu, Fri, Sat, Sun);
       { This is a subrange of `Days'. }
       Working = Mon .. Fri;
 var
   Jack   :Working;
   allday :Days;

begin
for allday := Low(Days) to High(Days) do
    writeln (allday);

 for Jack := Low(Working) to High(Working) do
    writeln (Jack);
end.
```

```
Free Pascal Compiler version 3.
Copyright (c) 1993-2021 by Flor
Target OS: Linux for x86-64
Compiling main.pas
Linking a.out
20 lines compiled, 0.0 sec
Mon
Tue
Wed
Thu
Fri
Mon
Tue
Wed
Thu
Fri
Sat
Sun
```

Convert to C# : compare c# with pascal and compare the pascal &c#

```
2  program Hello;
3     type
4           { This is an enumerated type. }
5           Days = (Mon, Tue, Wed, Thu, Fri, Sat, Sun);
6           { This is a subrange of `Days'. }
7           Working = Mon .. Fri;
8   var
9     manday :Working;
10    allday :Days;
11
12 begin
13          manday := Tue;    // ถูกต้อง
14          manday := Sat; // ผิด (อยู่นอกช่วง Mon..Fri)
15 end.
16
```

Perl
(dynamic data type)

```
@x1 = 10;
@y1 = 100;
@t = @x1;
@x1 = @y1;
@y1 = @t;

print "x1 = @x1\n";
print "y1 = @y1\n";

@x2 = "FNAME";
@y2 = "LNAME";
@t = @x2;
@x2 = @y2;
@y2 = @t;

print "x2 = @x2\n";
print "y2 = @y2\n";
```