For each of the following variables named x, specify whether they are static, stack-dynamic, or heap-dynamic:

a) in C++:
int* x = new(int); **heap-dynamic**

b) in Java:

```
class book {
    protected string title;
    book(String x)              stack-dynamic
        { title = x; }
}
```

What is the output of the following program? You may assume that the program compiles and runs without error.

```
public class Test {
    public static void f (int counter) {
        counter++;
    }

    public static void g (int counter) {
        f(counter);
        f(counter);
    }

    public static void main (String[] args) {
        int counter = 0;
        f(counter);
        System.out.print(counter);
        g(counter);
        System.out.println(counter);
    }
}
```

(A) 1 3
(B) 0 2
(C) 0 0
(D) 1 1
(E) 1 2

Consider the following program:

```
procedure Main is
        a, b: Integer;
    procedure sub1 is
        a, c, e: Integer;
      begin  -- of sub1
        a := 6; c := 7; e := 8;
        ...                <---------------- 3
      end; --  of sub1
    procedure sub2 is
        a, c, d: integer;
      begin   --- of sub2
        a := 3; c := 4;  d := 5;
        ...               <---------------- 2
        sub1();
        ...
      end;   -- of sub2
  begin
    a := 1; b := 2;   --- of Main
    ...               <---------------- 1
    sub2();
    ...
  end;    ----  of Main
```

(a) Determine what variables are visible at positions 1, 2, 3 and what their values are (given the call sequence indicated) with static scoping.

(b) Determine what variables are visible at positions 1, 2, 3 and what their values are (given the call sequence indicated) with dynamic scoping.

Position	***Static Scoping***	***Dynamic Scoping***
1	*a=1, b=2*	*a=1, b=2*
2	*a=3, b=2, c=4, d=5*	*a=3, b=2, c=4, d=5*
3	*a=6, b=2, c=7, e=8*	*a=6, b=2, c=7, d=5, e=8*

Chapter 6. Data Types

What is a data type?

- A type is a set: when you declare a variable of a certain type you are saying that the values the variable can have are elements of a certain set.

Primitive Data Types

- Any type that a program can use but cannot define

- Numerical types:

- Integer

  Java's signed integer sizes: byte, short, int, long

- Floating point

- Boolean

- introduced in ALGOL 60 and have been included in most programming languages since then

- Character

- stored with numeric coding
- ASCII (Most commonly used coding): 1 byte 0 – 127

  Extended ASCII code 128-255
- Unicode: 2 bytes, include characters from foreign languages

  – Originally used in Java

  – C# and JavaScript also support Unicode

Character String Type

- A sequence of characters

Two main design issues:

1. Should strings be simply a character array or a primitive type?

Pascal, C, C++, Ada:

- Not primitive, strings are just a character array and are defined as such in programs.
- these languages provide string operations: substrings, concatenation, assignments, etc.

C and C++

- use a string library (string.h): strcmp, strcat, etc.

• SNOBOL4 (a string manipulation language)

–Primitive

• Java

– Primitive via the String class

2. Should strings have static or dynamic length?

- Static length strings
- Limited dynamic length strings
- Dynamic length strings

(1) Static length strings:

- the length is specified in the string declaration(FORTRAN, COBOL, Pascal)

CHARACTER (LEN = 15) NAME1, NAME2
(FORTRAN 90)

- Static length strings are always full: if a shorter string is assigned to a string variable, the empty characters are set to blanks.

(2) limited dynamic length strings (C and C++):

- allow strings to have varying length up to a declared and fixed maximum set by the variable's definition.
- a special null character '\0' is used to indicate the end of a string's characters, rather than maintaining the length

```
void main(){
 char text_1[100], text_2[100];
 int i;
 char message[] = "Hello, I am a string;
                   what are you?";
  printf("Original message: %s\n", message);
   i=0;
 while ((text_1[i] = message[i]) != '\0')
    i++;
 printf("Text_1: %s\n", text_1);}
```

(3) dynamic length strings (no maximum) (JavaScript, Perl):

- allow strings to varying length with no maximum.

Perl is a typeless language, you can create strings by simply assigning any text to a scalar variable:

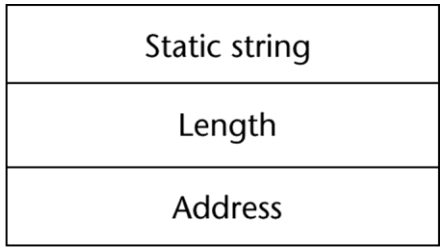
```
$name = "Sherzod";
$length = length($name);
```

Character String Implementation

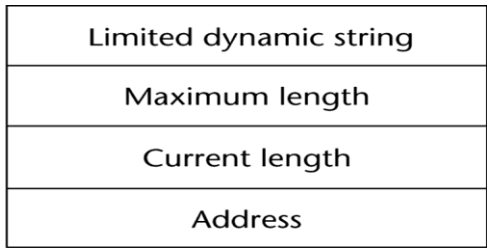
Software is used to implement string storage, retrieval and manipulation

- A descriptor for static length string: compile-time descriptor (required only during compilation time) such as Fortran.
- Limited dynamic length string: need a run-time descriptor for length (store both fixed maximum length and the current length) such as C or C++.

Compile- and Run-Time Descriptors



Compile-time descriptor for static strings



Run-time descriptor for limited dynamic strings

- `Dynamic length string` require a simpler run-time descriptor because only the current length needs to be stored

Dynamic length strings: how does one manage storage

(1) storing strings in a linked list
  - allocation and deallocation processes are simpler
  - large amount of storage and overhead

(2) store complete strings in adjacent storage cells
  - requires less storage
  - allocation process is slower

Array Types

- Index Syntax
  - –FORTRAN, PL/I, Ada use parentheses
  - –Most other languages use brackets

Arrays Index Types

- FORTRAN, C: integer only
- Pascal: any ordinal type (integer, Boolean, char)
- Ada: integer , Boolean or char.
- Java: integer types only
- C, C++, Perl, and Fortran do not specify range checking
- Java, ML, C# specify range checking

They give a run-time error if the program tries to use an out-of-range subscript.

Array Initialization

• Some language allow initialization at the time of storage allocation

–C, C++, Java, C#

int list [] = {4, 5, 7, 83}

–Character strings in C and C++ are implemented as array of char

char name [] = “freddie”;

– Arrays of strings in C and C++ can be initialized with string literal

char *names [] = {“Bob”, “Jake”, “Joe”};

–Java initialization of String objects

String[] names = {“Bob”, “Jake”, “Joe”};

- Ada provide two mechanisms for initialize arrays in the declaration statement: by listing them in the order in which they are stored, or by direct assigning them to an index position using =>operator.

List: array(1..5) of Integer := (1,3,5,7,9)

Brunch array(1..5) of Integer:= (1=> 17, 3=> 34, others => 0)

The **other** clause is used to initialize the remaining elements

Associative Arrays

• An unordered collection of **data elements** that are indexed by an **equal number** of values called ***keys***

Example in Perl

%hi_temps = ("Mon" => 77, "Tue" => 79, “Wed” => 65);

Associative Arrays in Perl

- Names begin with `%;` literals are delimited by parentheses

%hi_temps = ("Mon" => 77, "Tue" => 79, “Wed” => 65);

- Subscripting is done using braces and keys

```
$hi_temps{"Wed"} = 83;
```

% sign has changed to a **$** to access an individual element

- Elements can be removed with **`delete`**

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

User-Defined Ordinal Types

- The range of possible values can be easily associated with the set of positive integers
- Enumeration Types and Subrange Type

Enumeration Types
Every possible value of the type can be named, we say the data are enumerable.

Days of the week (sunday, monday, tuesday, etc), months in the year, colors in the rainbow.

```
enum Bread {wholewheat, ninegrain, rye, french}
Bread todaysLoaf;
todaysLoaf = Bread.rye;
System.out.println("bread choice is: " + todaysLoaf);
```

Output of: bread choice is: rye

Ordinal Values

Each value of an enumerated type is stored as an integer, called its *ordinal value*

- The *ordinal* method returns the ordinal value of the object
- The *name* method returns the name of the identifier corresponding to the object's value

```
public class IceCream {
    enum Flavor {vanilla, chocolate, strawberry, fudgeRipple, coffee, rockyRoad,
                 mintChocolateChip, cookieDough}
    public static void main (String[] args)   {
        Flavor cone1 = Flavor.rockyRoad;
        System.out.println ("cone1 value: " + cone1);
        System.out.println ("cone1 ordinal: " + cone1.ordinal());
     System.out.println ("cone1 name: " + cone1.name());}
```

```
cone1 value: rockyRoad
cone1 ordinal: 5
cone1 name: rockyRoad
```

User-Defined Ordinal Types

Subrange Types

• An ordered contiguous subsequence of an ordinal type

–Example: 12..18 is a subrange of integer type

• Ada's design

Subranges are included in the category of types called subtypes.

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;

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 the specified range is detected as an error

Record Types

An aggregate of data elements in which individual **elements** are **identified by names**

- for collection of **heterogeneous** data.
- Records can be nested.
- Introduced by COBOL, popular since then.

```
typedef struct {
  int age;
  char *name;
  enum sex { male, female };
} Person;
```

- Different languages use different syntax for records

```
struct PERSON {
    int age;
    long ss;
    float weight;
    char name[25];
} family_member;     // Define object of type PERSON
```

```
int main() {
    struct PERSON sister; // C style structure declaration
    PERSON brother; // C++ style structure declaration
    sister.age = 13; // assign values to members
    brother.age = 7; }
```

- The class structure in OO supports records.

```
class Person {
      String name;
      int id_number;
      Date birthday;
      int age;   }
```

References to record Field

- Dot-notation to select a field of a record.
  rec.field
- Nested records need a selection “path”:
  rec.fieldr1. … . fieldrn.field

Structures can generally be nested.

- Example in C.

```
struct point {
    int x;
    int y; };

enum colors {blue, green, yellow};

struct rectangle{
    struct point lleft;
    struct point uright;
    colors color;
    int fillpercent;};

struct rectangle r1;

r1.lleft.x = 4.5;

r1.lleft.y = 5;

struct cell {
   char data;
   struct cell *ptr;
}
```

- COBOL uses level numbers to show nested records;

```
01 EMPLOYMENT-RECORD.
   02 EMPLOYEE-NAME.
        05 FIRST     PICTURE IS X(20).
        05 MIDDLE  PICTURE IS X(10).
        05 LAST      PICTURE IS X(20).
   02 EMPLOYEE-ADDRESS
        ......
```

MIDDLE OF EMPLOYMENT-NAME OF EMPLOYEE-RECORD

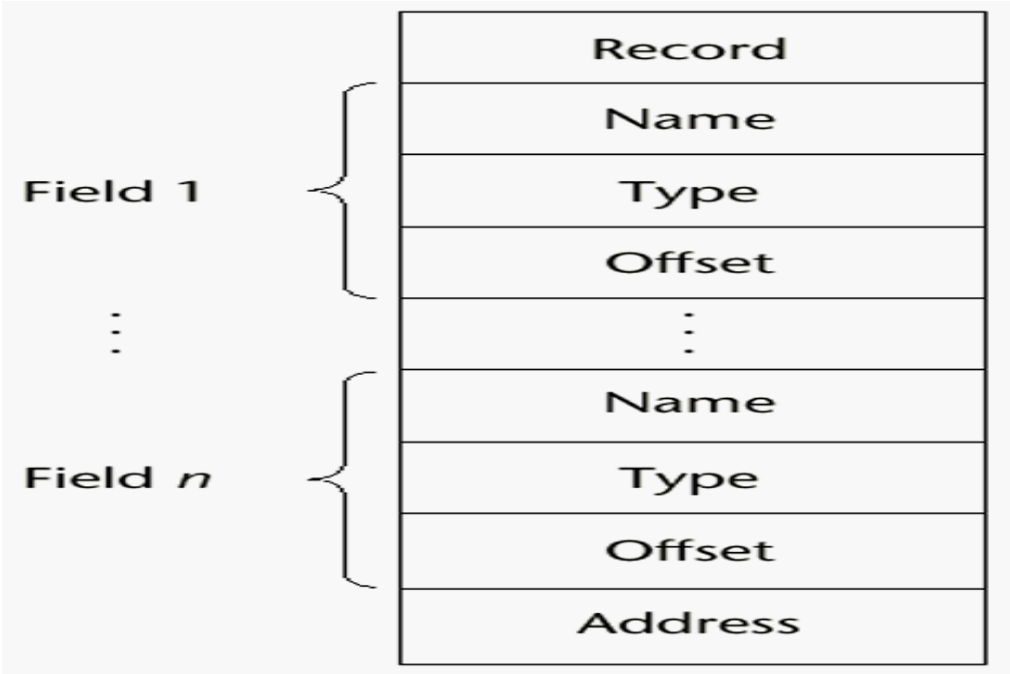
Evaluation and Comparison to Arrays

- Records are used when collection of data values is heterogeneous
- Access to array elements is much slower than access to record fields, because subscripts are dynamic (field names are static)

Implementation of Record Type

- Implementation: Stored contiguously in memory

- Offset address relative to the beginning of the records is associated with each field



Unions Types

- a user-defined data type that, at any given time, contains only one object from its list of members
- may store different type values at different times during a program execution.

• C/C++ language
• Similar to record type, it contains multiple data types.
• Members share the same data space. If a new assignment is made, the previous value is automatically erased
• Can manipulate different kinds of data in a single unit of storage
A union may be a member of a record. A record may be a member of union.

```
union number{
   int x;
   /*4 bytes*/
   double y;
   /*8 bytes*/
   char z[20];
   /*20 bytes*/
};
```

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

 main(){
    union {
          float u_f;
          int u_i;
    } var;

    var.u_f = 23.5;
    printf("value is %f\n", var.u_f);
    var.u_i = 5;
    printf("value is %d\n", var.u_i);
    exit(0);}
```

• Design issues
– Should type checking be required?

Discriminated vs. Free Unions

- *free union:* Fortran, C, and C++ provide union constructs in which there is no language support for type checking; programmer are allowed complete freedom from type checking.

```
#include <stdio.h>

int main() {
  union {
        int i;
        char *s;
  } x;
```

```
x.i = 'a';
printf(x.i);
```

- Discriminated unions or variant records (introduced in ALGOL 68)
  – Type checking is done
  – Used in ALGOL 68, Pascal, Ada

Example: (Ada Union Type)

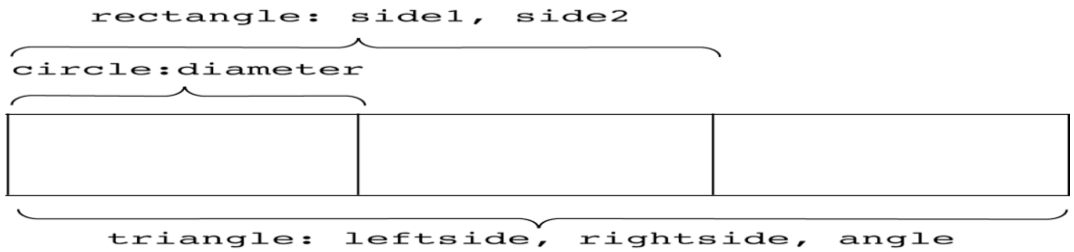
```
type Shape is (Circle, Triangle, Rectangle);

type Figure (Form: Shape) is
  Record
      Filled : Boolean;
      case Form is
            when Circle => Diameter: Float;
            when Triangle =>
                  Leftside, Rightside: Integer;
                  Angle: Float;
            when Rectangle => Side1, Side2: Integer;
      end case;
 end record;
```

The following statement declare variables of Figure:

Figure_2 : Figure(Form=> Triangle)

Figure_2 declared is constrained to be a triangle and cannot be another variant.



Evaluation of Unions

- Potentially unsafe construct
  - Do not allow type checking
- Java and C# do not support unions