Part I — Recognition (20)

1. Which best defines a *data type*?
   a) A set of heterogeneous values and one operation
   b) A homogeneous collection of values and the operations on them
   c) A memory block reserved at run time only
   d) A compiler directive for optimization
2. A type system is used primarily for:
   a) Code formatting
   b) Error detection and program organization
   c) Faster I/O
   d) GUI rendering
3. A type system typically includes:
   a) Only built-in types
   b) New-type mechanisms and rules for equivalence/compatibility/inference
   c) Garbage collection policy only
   d) Linker scripts
4. Scalar types are:
   a) Composite and heterogeneous
   b) Atomic and can compose other types
   c) Only booleans and integers
   d) Only hardware-independent
5. In IEEE-754 single precision, the fields are:
   a) 1 sign, 11 exponent, 52 fraction
   b) 1 sign, 8 exponent, 23 fraction
   c) 0 sign, 10 exponent, 53 fraction
   d) 1 sign, 7 exponent, 24 fraction
6. In an **enumeration type**, a key design issue is whether:
   a) Enum constants may appear in more than one enum definition
   b) The enum must start from value 100
   c) It must be 64-bit
   d) It must be printable only as an integer
7. A *subrange type* is:
   a) A dynamic array
   b) A contiguous subset of an ordinal type
   c) Any user class
   d) A record with only two fields
8. Jagged vs rectangular arrays:
   a) Jagged use row-major, rectangular use column-major
   b) Jagged allow rows of different lengths; rectangular have uniform shape
   c) Rectangular are only in C++
   d) Jagged are only in Fortran
9. A *slice* of an array/list is:
   a) A deep copy only

b) A referencing mechanism for a substructure
c) A pointer to the first element only
d) Only valid for 1-D arrays

10. In row-major storage, the second subscript varies:
    a) Slowest
    b) Fastest
    c) Neither
    d) Unspecified
11. An *associative array* (map/dict) is indexed by:
    a) Only integers
    b) Only strings
    c) Keys chosen by the programmer (hashable or comparable)
    d) Memory addresses
12. String length strategies include:
    a) Static, limited dynamic, and dynamic length
    b) Only dynamic
    c) Only static
    d) Only limited dynamic
13. A *record* is:
    a) Homogeneous aggregate
    b) Heterogeneous aggregate with named fields
    c) Linked list node
    d) Fixed-point number
14. Two fundamental pointer operations are:
    a) Hashing and concatenation
    b) Assignment of addresses and dereferencing
    c) Subtyping and overloading
    d) Boxing and unboxing
15. A *dangling pointer* is:
    a) A pointer set to null
    b) A pointer to deallocated storage
    c) A pointer to constant memory
    d) A reference parameter
16. Compared to pointers, C++ references:
    a) Can be reseated to another object
    b) Can be null by default
    c) Must be bound at initialization and cannot be reseated
    d) Do not alias their targets
17. *Type equivalence* "by name" means two types are equivalent if:
    a) They have identical structure only
    b) They have the same declared type name
    c) They occupy the same number of bytes
    d) One can be cast to the other
18. *Type compatibility* is best described as:
    a) The same as equivalence

b) Allowing a value of T wherever S is admissible under certain rules
c) Only allowing implicit casts
d) Only allowing explicit casts

19. *Coercion* vs *cast*:
a) Coercion is explicit; cast is implicit
b) Both are explicit
c) Coercion is implicit; cast is explicit
d) Neither changes representation
20. Polymorphism kinds include:
a) Ad hoc (overloading) and universal (parametric, subtyping)
b) Only subtyping
c) Only parametric
d) Only templates

Part II — Application (20)

21. The largest **signed** integer with 7 bits in two's complement is:
a) 63 b) 64 c) 127 d) 32
22. Assume `char=1B, short=2B, int=4B, float=4B`, 4-byte alignment. What is the size of

```
struct X { char a; int b; char c; short d; };
```

a) 11 b) 12 c) 14 d) 16
23. Row-major 2-D array `a[0..2][0..3]` of 4-byte ints, base address α=1000. Address of `a[2][1]` is:
a) 1016 b) 1024 c) 1036 d) 1048
24. Column-major 2-D array with 1-based indices `A[1..3,1..4]`, 4-byte ints, base α=1000. Address of `A[3,2]` is:
a) 1008 b) 1016 c) 1020 d) 1032
25. The set type `set of -2..13` represented by a bit chain needs how many bits?
a) 14 b) 15 c) 16 d) 17
26. In C, which output is *well-defined* for

```
union U { int data; unsigned char bt[4]; } x; x.data = 0x00007A12; printf("%u %u\n",
x.bt[0], x.bt[1]);
```

(Assume little-endian)
a) `18 122` b) `122 18` c) `7 162` d) Implementation-defined, but on little-endian it is `18 122`
27. For the Pascal-style 3-D array `X[2..4, -3..5, -2..4]` of 4-byte integers stored **column-major** (rightmost subscript varies slowest), base α. Which linearization formula is correct for address of `X[i,j,k]`?
a) α + (((i-2)*9 + (j+3))*7 + (k+2))*4
b) α + (((k+2)*9 + (j+3))*3 + (i-2))*4

c) α + (((j+3)*3 + (i-2))*7 + (k+2))*4
d) α + (((i-2)*7 + (k+2))*9 + (j+3))*4

28. With 32-bit IEEE-754 single precision, how many fraction bits are stored?
a) 22 b) 23 c) 24 d) 52
29. In most languages that allow it, `char name[] = "abc";` initializes the array with length:
a) 3 b) 4 (including `'\0'`) c) 5 d) Implementation-defined
30. Given the *slice* in Python:

```
v = [2,4,6,8,10,12]; v[1:5:2]
```

The result is:
a) `[4,8]` b) `[4,6,8,10]` c) `[4,6,8]` d) `[4,8,12]`

31. In C/C++, which statement about pointers is true?
a) `void*` can be dereferenced safely
b) Pointer arithmetic on `int* p` adds bytes, not elements
c) `p[index]` is syntactic sugar for `*(p+index)`
d) `&` returns the value stored at the pointer
32. With 1-byte `char`, 2-byte `short`, 4-byte `int`, structure padding often makes the final struct size:
a) A multiple of the smallest member's alignment
b) Exactly the sum of field sizes
c) A multiple of the largest member's alignment
d) Always a power of two
33. In C#, which pair is most accurate?
a) `string` is a primitive, fixed-length type
b) `string` is immutable and reference-semantics
c) `string` is always a 1-byte char array
d) `string` can be resized in place
34. Which is **not** a typical string operation listed in the handout?
a) Assignment b) Concatenation c) Pattern matching d) Matrix inversion
35. Regarding parameter passing in C++:
a) Reference parameters cannot alias the actuals
b) References can be reseated by assignment
c) `int& r = a;` binds `r` to `a` and `++r` increments `a`
d) Taking address of a reference is ill-formed
36. For a set type implemented by a **bit chain**, membership test `x in S` is O(1) because:
a) Hashing is used
b) The ith bit can be masked directly
c) A linked list is traversed
d) A binary search tree is used
37. For dynamic arrays in C++ `vector<int> v`, subscripting is:
a) Always range-checked
b) Never range-checked
c) Range-checked by `at`, unchecked by `operator[]`
d) Only at compile time

38. In Java, arrays are:
    a) Value types
    b) Reference types with runtime bounds checking
    c) Unchecked pointer arithmetic blocks
    d) Always jagged with different row lengths enforced
39. Which statement about **unions** is true?
    a) Java supports C-style unions
    b) C free unions do not carry a discriminant for type checking
    c) Ada unions are free unions
    d) Unions cannot overlap memory
40. Which is a correct *type expression* form?
    a) `array(I,T)` for index type `I` and element `T`
    b) `function(T1,T2)` for product types
    c) `pointer → T`
    d) `record(name:T1;name:T2)` is not a type expression

Part III — Advanced Application (20)

For Q41–Q60, assume the following library typings (when used):

- `map : (T1 → T2) × List[T1] → List[T2]`
- `filter : (T → bool) × List[T] → List[T]`
- `reduce : (U × V → U) × U × List[V] → U`
- `index : (T → bool) × List[T] → List[int]`
- `floor : real → int`
- List literal `[ ... ] : List[T]` if elements are of type `T`.

41. Consider pseudo-Python:

```
def foo(x,y,z): return reduce(z(x), y, [])
```

If `x` is an `int`, which is the most precise principal type for `foo`?
a) `(int × (int→T) × (T→ int→ T)) → T`
b) `(int × T × (T × int → T)) → T`
c) `(int × T × (T × int → T)) → int`
d) `(int × (int→T) × (int × T → T)) → T`

42. In a strict language, `if y(z(x)) then x else 0` type-checks only if:
    a) `y(z(x)) : bool` and `x, 0` share a common supertype
    b) `y(z(x)) : int`
    c) `x : bool`
    d) `0 : bool`
43. Bit-chain question: `X : set of -2..13`. If the least significant bit encodes the **smallest** element, which pair encodes `{ -2, 5 }`?
    a) `1000...0001`

b) `0000...0001`
c) A bitstring with bit 0 and bit 7 set
d) A bitstring with bit 2 and bit 13 set

44. Column-major 3-D array `A[0..2, 1..3, 0..1]` of 4-byte ints, base α=2000. Address of `A[2,3,1]` is:
(Use order: last index varies slowest.)
a) 2012 b) 2032 c) 2060 d) 2084

45. C struct alignment. Assume 8-byte alignment for `double`, 4-byte for `int`, 1-byte for `char`:

```
struct R { char a; double b; int c; char d; };
```

Total size is:
a) 18 b) 24 c) 32 d) 40

46. Free union punning (little-endian), choose the **most correct** statement:

```
union V { float f; unsigned int u; } v; v.u = 0x3F800000;
```

a) `v.f` is exactly `1.0f` per IEEE-754
b) Behavior is undefined
c) It is defined and always yields `0.0f`
d) It traps

47. Given:

```
def h(x): return lambda y,z: filter(y, map(z, x))
```

With the library typings above, the principal type of `h` is:
a) `List[T] → ((T → bool) × (U → T) → List[T])`
b) `List[T] → ((T → bool) × (U → T) → List[U])`
c) `List[T] → ((U → bool) × (T → U) → List[U])`
d) `List[T] → ((U → bool) × (U → T) → List[T])`

48. Consider:

```
def g(x): def k(z): return y(z) + x return k
```

In a statically typed setting with `+` overloaded for `int` and `real` only, pick a valid typing when `y : (int → int)`.
a) `x : int`, result `k : (int → int)`
b) `x : bool`, result `k : (int → bool)`
c) `x : real`, result `k : (int → real)`
d) a or c

49. Two's complement: the range of an `n`-bit signed integer is:
a) `[-2^(n-1), 2^(n-1)-1]`
b) `[0, 2^n - 1]`
c) `[-2^n, 2^n-1]`
d) `[-2^(n-1)+1, 2^(n-1)]`

50. IEEE-754 single precision. Which bit pattern encodes **−0.0**?
    a) Sign=1, all exponent and fraction bits 0
    b) Sign=0, exponent all ones, fraction nonzero
    c) Sign=0, exponent bias, fraction all zeros
    d) Sign=1, exponent all ones, fraction all zeros
51. With `index : (T→bool) × List[T] → List[int]` and `floor : real → int`, consider:

```
def q(u,v): return index(v, map(floor, u))
```

A principal type for `q` is:
a) `(List[real] × (int → bool)) → List[int]`
b) `(List[int] × (real → bool)) → List[int]`
c) `(List[real] × (real → bool)) → List[int]`
d) `(List[T] × (T → bool)) → List[int]`

52. Name vs structural equivalence. Which pair is **equivalent by structure** but not by name?
    a) Two distinct typedefs of the same record layout
    b) An `int` and a `float`
    c) `array(0..9, int)` and `array(1..10, int)`
    d) A class and its subclass
53. Parametric polymorphism example:

```
template <class T> void swap(T& x, T& y) { T t=x; x=y; y=t; }
```

Which is true?
a) Works only for built-ins
b) Requires `T` to be a pointer
c) Works for any `T` with copy/move semantics
d) Requires virtual dispatch

54. Subtyping polymorphism example:

```
struct Polygon { virtual float area() = 0; }; struct Rect : Polygon { float area(){return
h*w;} float h,w; }; Polygon* p = new Rect{3,4};
```

The dynamic call `p->area()` is resolved:
a) At compile time by static type
b) At run time via dynamic dispatch
c) By template instantiation
d) Never, because abstract

55. Python lists are:
    a) Immutable and fixed-length
    b) Mutable, dynamic, and can contain heterogeneous elements
    c) Only numeric
    d) Value-semantics like C arrays
56. In C, which causes a **dangling pointer**?
    a) `int *p = NULL;`

b) `int *p = malloc(4); free(p); *p = 5;`
c) `int a=3; int *p=&a;`
d) `int *p = (int*)0x0;`

57. Consider:

```
def r(f, xs): return reduce(lambda acc, t: acc + [f(t)], [], xs)
```

Given list concatenation and single-element append via `acc + [f(t)]`, the principal type is:
a) `((T→U) × List[T]) → List[U]`
b) `((T→U) × List[U]) → List[T]`
c) `((T→bool) × List[T]) → List[T]`
d) `(T × List[T]) → List[T]`

58. Column-major `B[1..4, 1..5]` of 8-byte doubles, base α=5000. Address of `B[4,5]` is:
a) 5000 + ((5-1)*4 + (4-1))*8
b) 5000 + ((4-1)*5 + (5-1))*8*
*c) 5000 + (4*5)*8*
d) 5000 + ((5-1)*5 + (4-1))*8

59. Record vs array comparison (from the evaluation slide). Which is accurate?
a) Arrays are heterogeneous, records homogeneous
b) Arrays are processed in the same way per element; record fields can be processed differently
c) Records support dynamic subscripting
d) Arrays never allow dynamic subscripting

60. Reference vs pointer in C++. Which is valid?
a) `int& r = a; r++;` mutates `a` and `r` cannot be reseated
b) `int& r;` is fine without initialization
c) `int* p = &a; p = &b;` reseats pointer the same way a reference reseats
d) References can be null by default

Answer Key (letters)

1 b
2 b
3 b
4 b
5 b
6 a
7 b
8 b
9 b
10 b
11 c
12 a
13 b
14 b

15 b
16 c
17 b
18 b
19 c
20 a
21 a
22 b
23 c
24 c
25 c
26 d
27 b
28 b
29 b
30 a
31 c
32 c
33 b
34 d
35 c
36 b
37 c
38 b
39 b
40 a
41 b
42 a
43 c
44 d
45 c
46 b
47 a
48 d
49 a
50 a
51 a
52 a
53 c
54 b
55 b
56 b
57 a
58 a
59 b
60 a