Computer memory is often abstracted as a sequence of bytes, grouped into words . Each byte has a unique address or index into this sequence .The size of a word (and byte!) determines the size of addressable memory in the machine . A pointer in C is a variable which contains the memory address of another variable (this can, itself, be a pointer) .Pointers are declared or defined using an asterisk(*); for example: char *pc; or int **ppi; .

The value “pointed to” by a pointer can be “retrieved” or dereferenced by using the unary * operator; for example: int *p = ... int x = *p; . The memory address of a variable is returned with the unary ampersand (&) operator; for example int *p = &x; . Dereferenced pointer values can be used in normal expressions; for example: *pi += 5; or (*pi)++. A C array uses consecutive memory addresses without padding to store data .

Pointer arithmetic can be used to adjust where a pointer points; for example, if pc points to the first element of an array, after executing pc+=3; then pc points to the fourth element . A pointer can even be dereferenced using array notation; for example pc[2] represents the value of the array element which is two elements beyond the array element currently pointed to by pc . In summary, for an array c, *(c+i)≡c[i] and c+i≡&c[i] . A pointer is a variable, but an array name is not; therefore pc=c and pc++ are valid, but c=pc and c++ are not.

Recall that all arguments to a function are copied, i.e. passed-by-value; modification of the local value does not affect the original . In the second lecture we defined functions which took an array as an argument; for example void reverse(char s[]) . Why, then, does reverse affect the values of the array after the function returns (i.e. the array values haven’t been copied)? .

C allows the creation of arrays of pointers; for example int *a[5].Arrays of pointers are particularly useful with strings . An example is C support of command line arguments: int main(int argc, char *argv[]) { ... } . In this case argv is an array of character pointers, and argc tells the programmer the length of the array. Multi-dimensional arrays can be declared in C; for example: int i[5][10]; .

To define an instance of the structure circle we write struct circle c; . A structure can also be initialised with values: struct circle c = {12, 23, 5}; .An automatic, or local, structure variable can be initialised by function call: struct circle c = circle_init(); .A structure can declared, and several instances defined in one go: struct circle {int x; int y; unsigned int r;} a, b. A structure declaration can contain a member which is a pointer whose type is the structure declaration itself.

C allows the programmer to use pointers to functions . This allows functions to be passed as arguments to functions ! For example, we may wish to parameterise a sort algorithm on different comparison operators (e.g. lexicographically or numerically) . If the sort routine accepts a pointer to a function, the sort routine can call this function when deciding how to order values. A structure is a collection of one or more variables . It provides a simple method of abstraction and grouping ..

A structure member can be accessed using ‘.’ notation: structname.member; for example: pt.x . Comparison (e.g. pt1 > pt2) is undefined . Pointers to structures may be defined; for example: struct circle *pc . When using a pointer to a struct, member access can be achieved with the ‘.’ operator, but can look clumsy; for example: (*pc).x . Alternatively, the ‘->’ operator can be used. A union variable is a single variable which can hold one of a number of different types .