• The advantage of this ADT definition is that it clearly points out the fact that the array is a more general structure than a "consecutive set of memory locations"

2.1.2 Arrays in C

• One-dimensional arrays in C : int list[5], *plist[5];
All arrays start at index 0.

• When the compiler encounters an array declaration, such as above, it allocates 5 consecutive memory locations. Each memory location is large enough to hold a single integer. The address of the 1st element list[0], is called the base address.
If the size of an integer on your machine is denoted by sizeof(int), then the memory address of list[i] is α+i*sizeof(int), where α is the base address.

• In fact, when we write list[i] in a C program, C interprets it as a pointer to an integer whose address is list[i] is α+i*sizeof(int).

list[i] = *(list+i) // *list[i] = list+i

Ex-2.1 [One-dimensional array addressing]: Assume that we have the following declaration: int one[] = {0,1,2,3,4}; Write a function that prints out both the address of the ith element of this array and the value found at this address.

```
void print1(int *ptr, int rows)
{ /* print out a 1-D array using a pointer */
  int i;
  printf("Address Contents\n");
  for(i=0; i<rows; i++)
    printf("%8u%5d\n", ptr+i, *(ptr+i));
  printf("\n");
}
```

↳ Output will take up atleast 8 character spaces

• When writing computer programs, we often find ourselves in a situation where we cannot reliably determine how large an array to use. A good soln. to this problem is to defer this decision to run-time and allocate the array when we have a good estimate of the required array size.