Variable Length Arrays (VLA)

- In C90 the size of objects must be known at compile time which means that you cannot declare an array whose size varies with the needs of the program as it runs
- If you only need a single dimensional array, you can get the effect of an array with variable bounds by dynamically allocating it.

OLD, most common way:

```
void process(int n)
{
   // Set up a buffer of n
     characters
   char *b = malloc(n);
   // do the work
   ...
   // free the buffer
   free(b);
}
```

- VLAs gives arrays the same flexibility and elegance that pointer types in C have traditionally had.
- Storage management and the complexities of index calculations of arrays whose bounds are not compile-time constants are handled automatically by the compiler.

NEW, VLA way:

```
void process(int n)
{
   // Set up a buffer of n
     characters
   char b[n];
   // do the work
   ...
}
```

Variable Length Arrays (VLA)

OLD, most common way:

```
float *a;
a = malloc(m * n *
  sizeof(float));
for (i = 0; i < m; ++i)
  for (j = 0; j < n; ++j)
    a[i*n + j] = 1.0;
```

New, VLA way:

```
void f(int m, int n)
{
  float a[m][n];
  int i, j;
  for (i = 0; i < m; ++i)
    for (j = 0; j < n; ++j)
      a[i][j] = 1.0;
}
```

Variable Length Arrays (VLA)

Example in which VLA is used to simplify the code:

```
FILE* concat_fopen (char *s1, char *s2, char *mode)
{
  char str[strlen (s1) + strlen (s2) + 1];

  strcpy (str, s1);
  strcat (str, s2);

  return fopen (str, mode);
}
```

Variable Length Arrays (VLA)

VLA allows:

- bounds of an array can be a run-time expression
- permits pointers arithmetic
- **sizeof** operator is evaluated at run time

VLA rules:

- must be an local (automatic) variable of a block,
- cannot be declared at file scope.
- cannot have static or extern storage class.
- cannot be a member of a struct or union.
- cannot be initialized.

Variable Length Arrays (VLA)

- Advantages
  - **Fast**: adjusting the stack pointer and/or the frame pointer would have been done anyway so the cost of a VLA is nearly 0.
  - **Easy**: a simple definition, no pointer to initialize, to check to free and no risk of memory leaks.
  - **Readable** : it's really a simple concept, so less likely to introduce subtle bugs.

- Drawbacks
  - **Size limited**: with the stack size, the stack can blow up.
  - **Buffer overflows** are a bit more serious than on heap memory
  - **Portability**: not all compilers implement it