# Stride of an array

In computer programming, the **stride of an array** (also referred to as **increment**, **pitch** or **step size**) is the number of locations in memory between beginnings of successive array elements, measured in bytes or in units of the size of the array's elements. The stride cannot be smaller than the element size but can be larger, indicating extra space between elements.

An array with stride of exactly the same size as the size of each of its elements is contiguous in memory. Such arrays are sometimes said to have **unit stride**. Unit stride arrays are sometimes more efficient than non-unit stride arrays, but non-unit stride arrays can be more efficient for 2D or multi-dimensional arrays, depending on the effects of caching and the access patterns used. This can be attributed to the principle of locality, specifically *spatial locality*.

## Contents



## Reasons for non-unit stride

Arrays may have a stride larger than their elements' width in bytes in at least three cases:

### Padding

Many languages (including C and C++) allow structures to be padded to better take advantage either of the word length and/or cache line size of the machine. For example:

```
struct A {
    int a;
    char b;
};

struct A myArray[100];
```

In the above code snippet, `myArray` might well turn out to have a stride of eight bytes, rather than five (4 bytes for the int plus one for the char), if the C code were compiled for a 32-bit architecture, and the compiler had optimized (as is usually the case) for minimum processing time rather than minimum memory usage.

### Overlapping parallel arrays

Some languages allow arrays of structures to be treated as overlapping parallel arrays with non-unit stride:

```c
#include <stdio.h>

struct MyRecord {
    int value;
    char *text;
};

/*
    Print the contents of an array of ints with the given stride.
    Note that size_t is the correct type, as int can overflow.
*/
void print_some_ints(const int *arr, int length, size_t stride)
{
    int i;
    printf("Address\t\tValue\n");
    for (i=0; i < length; ++i) {
        printf("%p\t%d\n", arr, arr[0]);
        arr = (int *)((unsigned char *)arr + stride);
    }
}

int main(void)
{
    int ints[100] = {0};
    struct MyRecord records[100] = {0};

    print_some_ints(&ints[0], 100, sizeof ints[0]);
    print_some_ints(&records[0].value, 100, sizeof records[0]);
    return 0;
}
```

This idiom is a form of type punning.

## Array cross-section

Some languages like PL/I allow what is known as an *array cross-section*, which selects certain columns or rows from a larger array.[1]:p.262 For example, if a two-dimensional array is declared as

```
declare some_array (12,2)fixed;
```

an array of one dimension consisting only of the second column may be referenced as

```
some_array(*,2)
```

## Example of multidimensional array with non-unit stride

Non-unit stride is particularly useful for images. It allows for creating subimages without copying the pixel data. Java example:

```java
public class GrayscaleImage {
  private final int width, height, widthStride;
  /** Pixel data. Pixel in single row are always considered contiguous in this example. */
  private final byte[] pixels;
  /** Offset of the first pixel within pixels */
  private final int offset;

  /** Constructor for contiguous data */
  public Image(int width, int height, byte[] pixels) {
    this.width = width;
    this.height = height;
    this.pixels = pixels;
```

```
      this.offset = 0;
      this.widthStride = width;
    }

    /** Subsection constructor */
    public Image(int width, int height, byte[] pixels, int offset, int widthStride) {
      this.width = width;
      this.height = height;
      this.pixels = pixels;
      this.offset = offset;
      this.widthStride = widthStride;
    }

    /** Returns a subregion of this Image as a new Image. This and the new image share
        the pixels, so changes to the returned image will be reflected in this image. */
    public Image crop(int x1, int y1, int x2, int y2) {
      return new Image(x2 - x1, y2 - y1, pixels, offset + y1*widthStride + x1, widthStride);
    }

    /** Returns pixel value at specified coordinate */
    public byte getPixelAt(int x, int y) {
      return pixels[offset + y * widthStride + x];
    }
  }
```

# References


1. Hughes, Joan K (1979). *PL/I Structured Programming (second ed.)* (https://archive.org/details/pl istructuredpr00hugh). New York: John Wiley and Sons. ISBN 0-471-01908-9.