Arrays

Single-Dimensional Arrays

- Generic declaration:

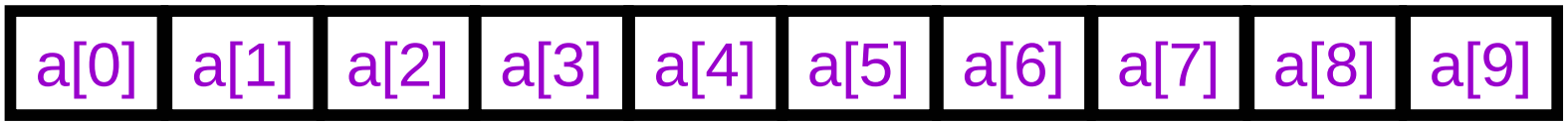
  typename variablename[size]

  - typename is any type
  - variablename is any legal variable name
  - size is a number the compiler can figure out

- For example

  int a[10];

- Defines an array of ints with subscripts ranging from 0 to 9
- There are 10*sizeof(int) bytes of memory reserved for this array.



- You can use a[0]=10; x=a[2]; a[3]=a[2]; etc.
- You can use scanf("%d",&a[3]);

Using Constants to Define Arrays

- It is useful to define arrays using constants:

```
#define  MONTHS 12
int  array [MONTHS];
```

- However, in ANSI C, you cannot

```
int n;
scanf("%d", &n);
int array[n];
```

- In GNU C, the variable length array is allowed.
- In ANSI C, the handling of variable length array is more complicated.

Array-Bounds Checking

- C, unlike many languages, does NOT check array bounds subscripts during:
  - Compilation (some C compilers will check literals)
  - Runtime (bounds are never checked)
- If you access off the ends of any array, it will calculate the address it expects the data to be at, and then attempts to use it anyways
  - may get “something…”
  - may get a memory exception (segmentation fault, bus error, core dump error)
- It is the programmer’s responsibility to ensure that their programs are correctly written and debugged!
  - This does have some advantages but it does give you all the rope you need to hang yourself!

Initializing Arrays

- Initialization of arrays can be done by a comma separated list following its definition.
- For example:

```
int array [4] = { 100, 200, 300, 400 };
```

  - This is equivalent to:

```
int array [4];
array[0] = 100;
array[1] = 200;
array[2] = 300;
array[3] = 400;
```

- You can also let the compiler figure out the array size for you:

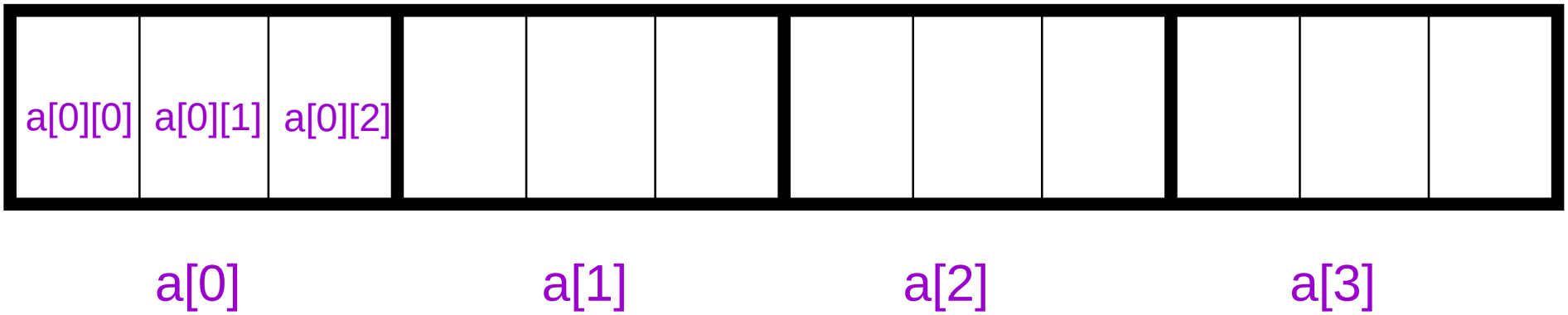
```
int array[] = { 100, 200, 300, 400};
```

A Simple Example

```
#include <stdio.h>
int main() {
  float expenses[12]={10.3, 9, 7.5, 4.3, 10.5, 7.5, 7.5, 8, 9.9,
    10.2, 11.5, 7.8};
  int count,month;
  float total;
  for (month=0, total=0.0; month < 12; month++)
  {
       total+=expenses[month];
  }
  for (count=0; count < 12; count++)
      printf ("Month %d = %.2f K$\n", count+1, expenses[count]);
  printf("Total = %.2f K$,   Average = %.2f K$\n", total, total/12);
  return 0;
}
```

Multidimensional Arrays

- Arrays in C can have virtually as many dimensions as you want.
- Definition is accomplished by adding additional subscripts when it is defined.
- For example:
  - int a [4] [3] ;
    - defines a two dimensional array
    - a is an array of int[3];
- In memory:



Initializing Multidimensional Arrays

- The following initializes a[4][3]:

```
int a[4] [3] = { {1, 2, 3} , { 4, 5, 6} , {7, 8, 9} , {10, 11, 12} };
```

- Also can be done by:

```
int a[4] [3] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
```

  - is equivalent to

```
a[0][0] = 1;
a[0][1] = 2;
a[0][2] = 3;
a[1][0] = 4;
...
a[3][2] = 12;
```

An Example

```
#include <stdio.h>
#include <stdlib.h>
int main ()  {
    int random1[8][8];
    int a, b;
    for (a = 0; a < 8; a++)
        for (b = 0; b < 8; b++)
                    random1[a][b] = rand()%2;
    for (a = 0; a < 8; a++)
    {
        for (b = 0; b < 8; b++)
             printf ("%c " , random1[a][b] ? 'x' : 'o');
        printf("\n");
    }
   return 0;
}
```

The value of the array name

```
#include <stdio.h>
int main(){
    int i;
    int a[3] = { 1, 2, 3 };
    printf( "a ? %d\n", a);
    printf( "a[0] ? %d\na[1] ? %d\na[2]
            ? %d\n", a[0], a[1], a[2]);
    printf( "&a[0] ? %d\n&a[1] ?
            %d\n&a[2] ? %d\n", &a[0],
            &a[1], &a[2]);

    printf( "\na[0] <- 4 \n");
    a[0] = 4;
    printf( "a ? %d\n", a);
    printf( "a[0] ? %d\na[1] ? %d\na[2]
            ? %d\n", a[0], a[1], a[2]);
    printf( "&a[0] ? %d\n&a[1] ?
            %d\n&a[2] ? %d\n\n", &a[0],
            &a[1], &a[2]);
    for (i=0; i<3; i++) {
            printf( "a[%d] <- ",i);
            scanf( "%d", &a[i]);
    }
    printf( "a ? %d\n", a);
    printf( "a[0] ? %d\na[1] ? %d\na[2]
            ? %d\n", a[0], a[1], a[2]);
    printf( "&a[0] ? %d\n&a[1] ?
            %d\n&a[2] ? %d\n", &a[0],
            &a[1], &a[2]);
}
```

- When the array name is used alone, its value is the address of the array (a pointer to its address).
- &a has no meaning if used in this program.

Arrays as Function Parameters

- In this program, the array addresses (i.e., the values of the array names), are passed to the function inc_array().
- This does not conflict with the rule that “parameters are passed by values”.

```
void inc_array(int a[ ], int size)
{
    int i;
    for(i=0;i<size;i++)
    {
            a[i]++;
    }
}
```

```
void inc_array(int a[ ],int size);
main()
{
    int test[3]={1,2,3};
    int ary[4]={1,2,3,4};
    int i;
    inc_array(test,3);
    for(i=0;i<3;i++)
            printf("%d\n",test[i]);
    inc_array(ary,4);
    for(i=0;i<4;i++)
            printf("%d\n",ary[i]);
    return 0;
}
```

An Example -- Sorting

```
void mysort(int a[ ],int size)
{
  int i,j,x;
  for(i=0; i<size; i++)
  {
    for(j=i; j>0; j--)
    {
      if(a[ j ] < a[ j-1])
      {  /* Change the order of a[ j ] and
      a[ j-1] */
          x=a[ j ];a[ j ]=a[ j-1]; a[j-1]=x;
      }
    }
  }
}
```

```
int main()
{
      int i;
      int tab[10] = {3,6,3,5,9,2,4,5,6,0};

      for(i=0;i<10;i++)
                printf("%d ",tab[i]);

      printf("\n");
      mysort(tab,10);

      for(i=0;i<10;i++)
                printf("%d ",tab[i]);

      printf("\n");
      return 0;
}
```