CSC 2400: Computer Systems

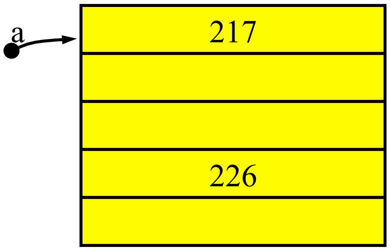
Arrays and Strings in C

Lecture Overview

- Arrays
  - List of elements of the same type
- Strings
  - Array of characters ending in ‘\0’
  - Functions for manipulating strings

Arrays in C

```
int a[5];
```



Label	Address
a[0]	400
a[1]	404
a[2]	408
a[3]	412
a[4]	416

What is "**a**" in the picture above?

a is the ***address*** of the first array element a[0]

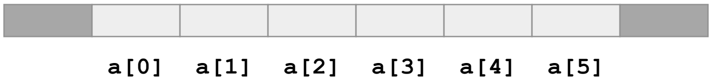
- not five consecutive array elements
- we will see that a is a constant pointer (covered in next lecture)

Array Indices

- Logically, valid indices for an array range from `0` to **`MAX-1`**, where **`MAX`** is the dimension of the array

```
int a[6];
      stands for
      a[0], a[1], a[2], a[3], a[4] and a[5]

      Logically, there is no a[6]!
```

- Memory



Arrays: C vs. Java

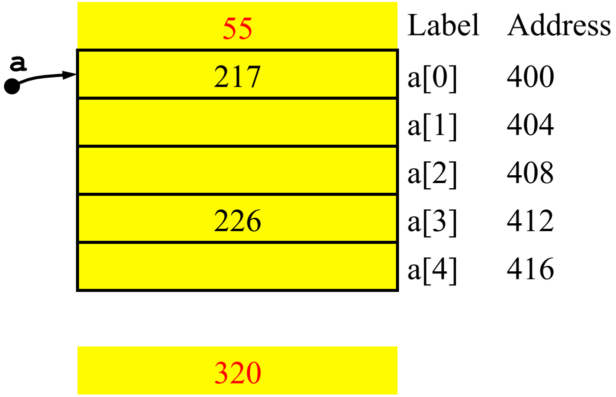
	Java	C

Arrays	`int [] a = new int [10];`
	`float [][] b =`
	`    new float [5][20];`
		`int a[10];`
		`float b[5][20];`

Array bound checking	`// run-time check`	`/* no run-time check */`

C Does Not Do Bounds Checking!

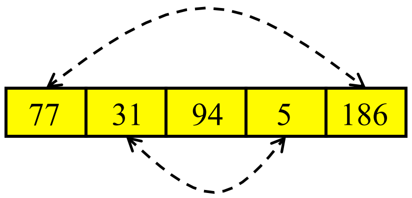
```
int a[5];


a[0] = 217;
a[3] = 226;


a[-1] = 55;
a[7] = 320;
```



Unpleasant if you happened to have another variable before the array variable **`a`**, or after it!

Example Program: Reverse Array

- Reverse the values in an array
  - Inputs: integer array **`a`**, and number of elements **`n`**
  - Output: values of **`a`** stored in reverse order
- Algorithm
  - Swap the first and last elements in the array
  - Swap the second and second-to-last elements
  - …



77	31	94	5	186

Example of Array Code

```
void reverse (int a[], int n) {
  int l, r, temp;
  for (l=0, r=n-1; l<r; l++, r--) {
    temp = a[l];
    a[l] = a[r];
    a[r] = temp;
  }
}

int main(void) {
  int fib[] = {1,2,3,4,5};
  reverse(fib, 5);
}
```

NO Aggregate Array Operations

- Aggregate operations refer to operations on an array as a whole, as opposed to operations on individual array elements.

```
#define MAX  100
int x[MAX];
int y[MAX];
```

- There are no aggregate operations on arrays:

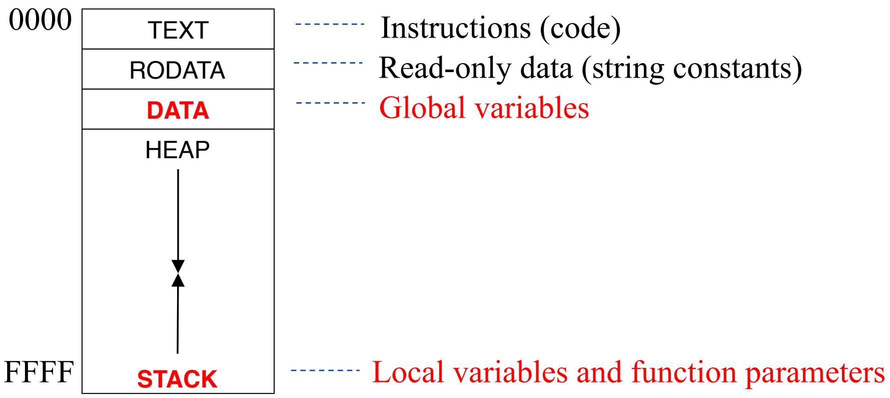
Assignment	`x = y;`	Error !
Comparison	`if (x == y) …`	Error !
I/O	`printf(“%d”, x);`	Error !
Arithmetic:	`x = x + y;`	Error !

Activity

- Write a small program that uses aggregate array operations. What error messages do you get?

Stack vs. Data

- At run-time, memory devoted to program is divided into **sections**:



Instructions (code)

Read-only data (string constants)

Global variables

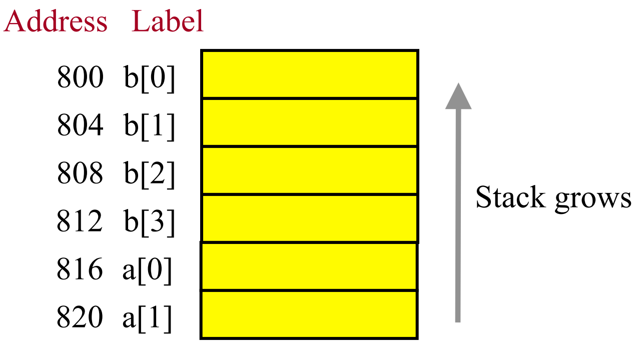
Local variables and function parameters

Clobbering Example 1

```
        /* This program accesses an invalid array cell.
        ** Why does it work? Draw the memory map. */
main()
{
  int a[2];    /* 2 cells, each cell 4 bytes (32 bits) */
  int b[4];    /* 4 cells, each cell 4 bytes (32 bits) */
  int c[4];    /* 4 cells, each cell 4 bytes (32 bits) */
  char d[5];   /* 5 cells, each cell 1 bytes (8 bits) */

  a[0]=5;
  b[1]=4;
  c[0]=9;
  d[4]='a';

  b[4]=10;
  printf("%d\n",b[4]);
  printf("%d\n",a[0]);   /* Why did a[0] change? */
}
```

Local Variables

- Are allocated on the stack
- The stack grows from high memory addresses towards low memory addresses.



Address	Label
800	b[0]
804	b[1]
808	b[2]
812	b[3]
816	a[0]
820	a[1]

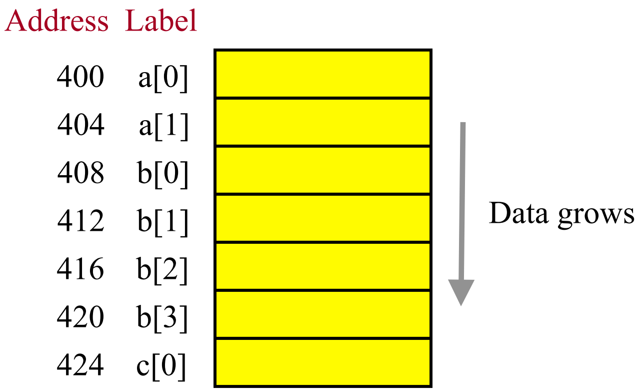
Stack grows

Clobbering Example 2

```
        /* This program accesses an invalid array cell.
        ** Why does it work? Draw the memory map. */

 int a[2];      /* 2 cells, each cell 4 bytes (32 bits) */
 int b[4];      /* 4 cells, each cell 4 bytes (32 bits) */
 int c[4];      /* 4 cells, each cell 4 bytes (32 bits) */
 char d[5];     /* 5 cells, each cell 1 bytes (8 bits) */

 main()
 {
   a[0]=5;
   b[1]=4;
   c[0]=9;
   d[4]='a';

   b[4]=10;
   printf("%d\n",b[4]);
   printf("%d\n",c[0]);   /* Why did c[0] change? */
 }
```

Global Variables

- Are allocated in the data section
- Memory in the data section is allocated from low memory addresses towards high addresses.



Address	Label
400	a[0]
404	a[1]
408	b[0]
412	b[1]
416	b[2]
420	b[3]
424	c[0]

Data grows

Strings in C

C vs. Java Strings

	Java	C

Strings	`String s1 = "Hello";`
	`String s2 = new`
	`    String("hello");`
		`char s1[] = "Hello";`
		`char s2[6];`
		`strcpy(s2, "hello");`

String concatenation	`s1 = s1 + s2`
	`s1 += s2`
		`#include <string.h>`
		`strcat(s1, s2);`

Strings

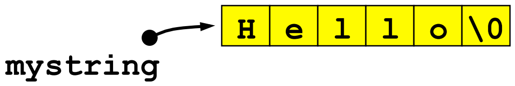
- Unlike Java, there is no String data type in C
- A string is just an array of characters (pointer to character), terminated by a ‘\0’ char (a null, ASCII code 0).

char mystring[6] = {’ H’ ,’ e’ ,’ l’ ,’ l’ ,’ o’ ,’ \0’ };

char mystring[6] = “Hello”;

char mystring[] = “Hello”;

Equivalent

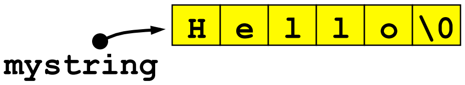


H	e	l	l	o	\0

mystring

Printing a String

printf(“%s”,mystring);



mystring

H e l l o \0

```
int i;
for (i=0; mystring[i]; i++)
   putchar(mystring[i]);
```

Reading Into a String

- Always use fgets:

```
#define MAX_BUFFER 20
char buffer[MAX_BUFFER];

fgets(buffer, MAX_BUFFER, stdin);
```

- Avoids going past the boundary of the array

String Termination

char mystring[] = “Hello”;

mystring	H	e	x	l	o	!

mystring[2] = 0; *equivalently,* mystring[2]=’\0’ ;

printf(“%s\n”,mystring);

He

mystring[2] = ‘x’ ; mystring[5] = ‘!’ ;

printf(“%s\n”,mystring);

What will happen?

Functions for Manipulating Strings

- C provides a large number of functions for manipulating strings. Four important ones are:

```
strlen(s)
// returns the length of s

strcpy(toS, fromS)
// copy fromS to toS (toS must be large enough)

strcmp(s1, s2)
// returns 0 if s1 == s2
// returns an integer < 0 if s1 < s2
// returns an integer > 0 if s1 > s2

strncmp
sprintf
strcat -   read online to find out what
           these functions do
```

Hands-On: Understanding strlen

- Step 1: Write a simple test program to see how strlen behaves.
- Step 2: Write a function length that mimics strlen:

```
int length (char s[])
```

// Input: string `s` terminated by `'\0'`
// Output: length of `s` (not counting `'\0'`)

Do not use *strlen* in your code.

strlen() - Computing the length of a string

```
#include <stdio.h>
#include <string.h>

main()
{
  int length;
  char s[6];

  s[0]='S'; s[1]='u'; s[2]='e'; s[3]='\0';

  length=0;
  while (s[length] != '\0')
    length++;

  printf("%d\n",length);
}
```

Equivalent to:

```
length=strlen(s);
```

Hands-On: Understanding strcmp

- Step 1: Write a simple test program to see how strcmp behaves.
- Step 2: Write a function compare that mimics strcmp:

```
int compare (char s1[], char s2[])
 // Input: strings s1 and s2 terminated by '\0'
 // Output: 0 if s1 == s2, else the difference between
 // the first pair of characters that do not match
```

Do not use *strcmp* in your code.

Hands-On: Understanding strcpy

- Step 1: Write a simple test program to see how strcpy behaves.
- Step 2: Write a function copy that mimics strcpy:

```
void copy (char t[], char s[])
// Input: strings t(arget) and s(ource) terminated by '\0'
// Action: copy s into t
// Assumption:  t is big enough memory to hold s
```

Do not use *strcpy* in your code.

Hands-On: Understanding strcat

- Step 1: Write a simple test program to see how strcat behaves.
- Step 2: Write a function concat that mimics strcat:

```
void concat (char t[], char s[])
// Input: strings t(arget) and s(ource) terminated by '\0'
// Action: append s at the end of t
// Assumption:  t is big enough to hold s concatenated to t
```

Do not use *strcat* in your code.

sprintf() – Print formatted output into a string

```
#include <stdio.h>
#include <string.h>

main()
{
  char a[24];
  float f;
  int i;

  f=3.72;
  i=9;

  sprintf(a,"Price %f, qty %d",f,i);

  printf("%s\n",a);
}
```

Command Line Arguments

```
      /* Print out the command line arguments
      ** - they are an array of strings */

int main(int argc, char *argv[])
{
int i,j;

  for (i=0; i<argc; i++) {
    j=0;
    while (argv[i][j] != '\0')
    {
      printf("%c",argv[i][j]);
      j++;
    }
    printf("\n");
  }
}
```

Equivalent to:

```
printf("%s\n",argv[i]);
```

Multi-Dimensional Arrays

2D Arrays

```
       /* How does a 2D array fit in memory?
       ** Draw the memory map. */

#include <stdio.h>

main()
{
   int a[3][2];

   a[0][1]=7;
   a[1][0]=13;
}
```

3D Arrays

```
       /* How does a 3D array fit in memory?
       ** Draw the memory map. */

#include <stdio.h>

main()
{
   int b[2][3][4];

   b[0][2][0]=7;
   b[1][0][2]=13;
}
```

Summary

- Arrays
  - Lists of elements of the same type
  - No bounds checking in C !!!!!!
  - No aggregate array operations

- Strings
  - Arrays of characters
  - Special end-of-string character ‘\0’
  - Special manipulating functions (string.h)