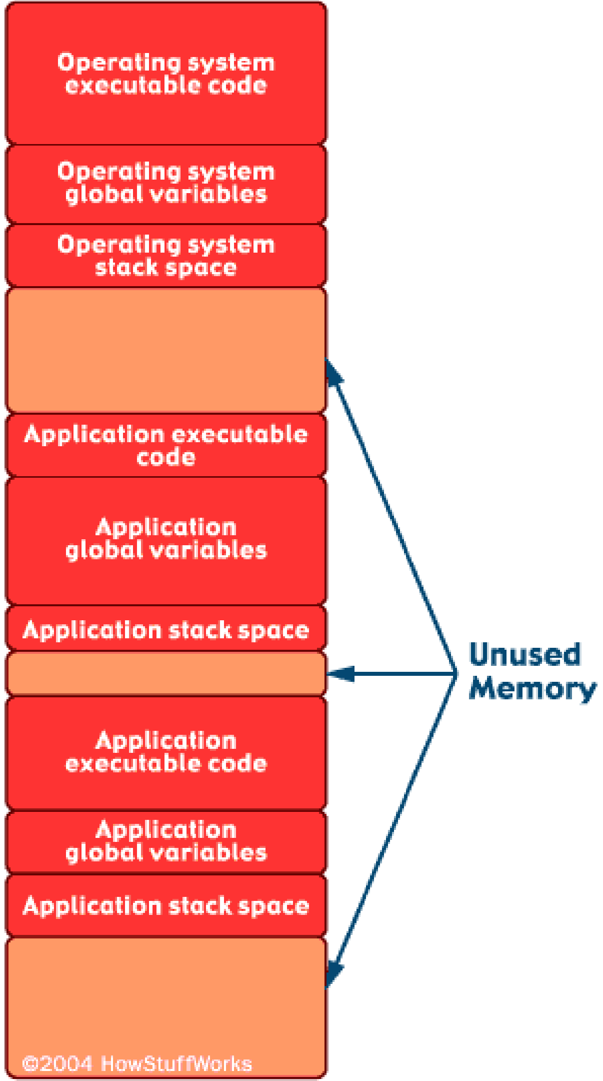
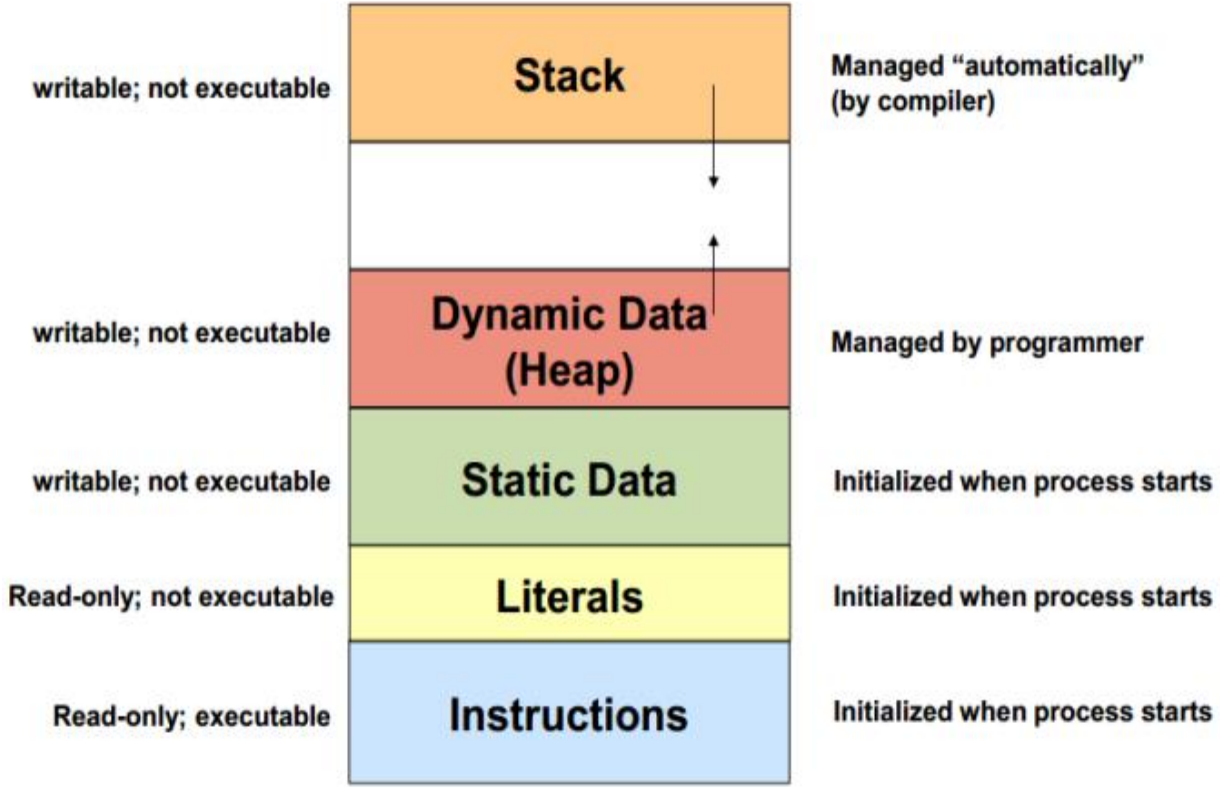
Computer Systems & Low-Level Programming

C: multidimensional arrays, pointers to functions, preprocessor, chars and strings

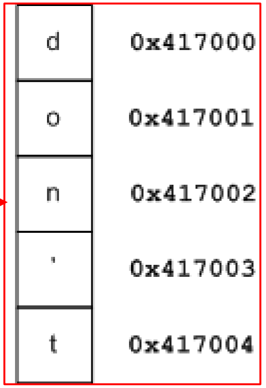
Marija Stanojevic
Spring 2019

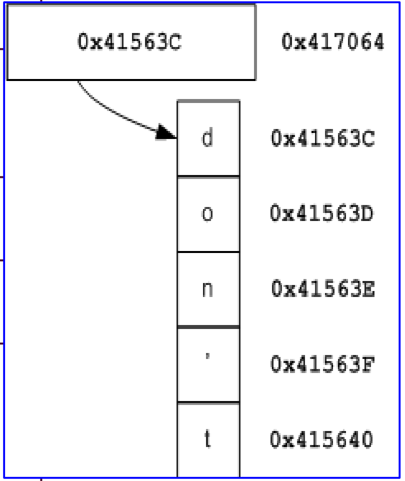
Memory details


Operating system executable code
Operating system global variables
Operating system stack space
Application executable code
Application global variables
Application stack space
Application executable code
Application global variables
Application stack space
Unused Memory
©2004 HowStuffWorks



1	char a[10]; // allocates place for 10*int in stack and stores their address in **a**	char *p; // allocates place for one pointer in memory, **p** value is null
2	**a** is an array	**p** is a pointer
3	char a[10] = “don’t”; //stores don’t as first 6 elements of **a** in stack; &a = a	char *p = “don’t”; //p points to instructions section where “don’t” is; &p!=p
4	a[2]; //gives ‘n’ as *(a+2); *a ⇔ a[0]	p[2]; //gives ‘n’, same as *(p+2); *p ⇔ p[0]
5	a = “hello”; //gives an error; we can change only element by element	p = “hello”; //p now points to place in instruction section where “hello” is
6	a++ //gives an error	p++ //shows on the next address
7	a[0] = ‘c’; // now we have “con’t” in a	p[0] = ‘c’; //gives an error
8	char a[5] = “Welcome”; //gives an error because Welcome size is > than 5	char *p = “Welcome”; //p now points to place in instruction section where “Welcome” is





Code: shows each of those properties

1D array and its length in the function

```
void reverseArray(int arr[], int n) {

        int i;
        for (i = 0; i < n/2; i++) {
            int tmp = arr[i];
            arr[i] = arr[n-i-1];
            arr[n-i-1] = tmp;
    }
} //Changes array in original space

int lengthOfArray(int arr[]) {
        return
sizeof(arr)/sizeof(arr[0]);
} //returns 8/4 = 2
```

- Variables are always passed by value to functions
- Arrays are always passed by reference to functions
  - => Changing array in function changes it in it's original space
  - => don't need to return array
- Can't return array from function, because it exists only until function exist. Make sure to define all arrays used from multiple functions globally or in main function.

Multidimensional arrays and functions

- Always passed by reference to functions
  - => Changing multidimensional array in function changes original matrix
  - => don’t need to return it
- Can’t access matrix created in some function after function is finished (don’t return it).
- Calls:
  - printMatrix(n, m, mat);
  - changeToOne((double*)mat, n, m);

○ Check Lab 5 code

```
void printMatrix(int n, int m, double mat[n][m]) {
        printf("Matrix is: \n");
        for (int i = 0; i < n; i++) {
                for (int j = 0; j < m; j++) {
                        printf("%f, ",
mat[i][j]);
                }
                printf("\n");
        }
}
```

```
void changeToOne(double* mat, int n, int m)
        int i, j;
        for (i = 0; i < n; i++) {
                for (j = 0; j < m; j++) {
                        *(mat + i * m + j)
= 1;
                }
        }
```

Pointers to functions

- void printArray(int arr[], int len);
- void reverseArray(int arr[], int len);
- void readArray(int arr[], int len);
- int maxOfArray(int arr[], int len); // can’t be included in array of pointers to function because it doesn’t have the same data type as the other functions
- void (*f[3])(int [], int) = {printArray, reverseArray, readArray};
  - f is name of array of pointers to functions and f has three elements
  - all functions have void return type and (int[], int) arguments types (in this case)
- (*f[2])(arr, 5); // calls reverseArray(arr, 5);
  - calls 2nd element of array f with arguments arr and 5
- int* f(); // function returning a pointer to an int
- int (*f)(); // pointer to function returning integer

Advanced pointers and order of operations

char ** cpp; //pointer to pointer to char

int (*arr)[13]; // pointer to array[13] of int

int *arr[13]; // array[13] of pointers to integer

void *fun(); // function returning pointer to void

void (*fun)(); // pointer to function returning void and without parameters

char*(*v[10])(); //array of 10 pointers to functions which return char pointer

void (*fun)(int); //pointer to a function that has int argument and returns nothing

- Type conversions:
    - Implicit (to bigger data types, int=>long, int => float, float => double, char => int,...)
    - Explicit with (cast) operator (e.g. (int)3.5; => 3, (float)0.333333333333; => 0.333333,...)
    - String to integer: atoi(“1234”); => 1234. String to float: atof(“12.34”); => 12.34
- Register variables:
    - Registers are located on CPU, the fastest memory, but very small
    - register int i = 10; // 10 is stored in registry; use this only if you will use i a lot in calculations
- int main(int argc, char* argv[]) - main can have those two parameter
    - argc is number of arguments and argv is array of strings with length argc; each string is different argument; those two parameters are optional
- Generating random numbers:
    - import<time.h>
    - srand(time(NULL)); // uses time to generate random values
    - rand() % (100 - 50 + 1) + 50; // gives random numbers between 50 and 100

Preprocessor

- #include and #define are preprocessor statements
- #define SQUARE(x) ((x)*(x)) - macro definition
- Other such statements: #if, #elif, #endif, #ifndef, #ifdef (conditional inclusion)
- #undef (undefine a defined value)
- #pragma startup or #pragram exit (call a function before/after main function)
- During the preprocessing step of compiling those are executed/checked:
    - All constants defined by #define are substituted in code with the value
    - All libraries included by #include are connected to the main code
    - If conditional inclusion is used compiler checks if those are satisfied
- Very important in complex projects when same stuff may be defined in multiple files or where different modules should execute for different cases

Handling characters and strings

- **<ctype.h>**
  - isdigit('0');
  - isalpha('A');
  - isalnum('A');
  - isxdigit('A');
  - islower('p');
  - isupper('p');
  - toupper('p');
  - tolower('P');
  - isspace('\n');
  - iscntrl('\t');
  - ispunct(':');
  - isprint('$');
  - isgraph('\n');

- **<string.h>**
  - char str[40] = strcat(x, y); //concatenates x and y and stores that string in str
  - char str[30] = strncat(x, y, 6); // concatenates x with first 6 characters of y and saves in str
  - strcmp(x, y); //compares lexically x and y
  - strncmp(x, y, 6); //compares first 6 characters from x and y lexically (returns -1, 0, 1)
  - strchr(str, c); //returns pointer to first position of c in str
  - strcspn(s1, s2); //number of characters on the begining in s1 which are not in s2
  - strrchr(s1, c); //part of s1 which starts with c
  - strspn(s1, s2); //initial part of s1 containing only characters from s2

Handling strings and memory

- **<string.h>**
  - strcpy(y, x); //copies from x to y
  - strncpy(y, x, 10); //copies first 10 chars from x to y
  - strstr(s1, s2); //first occurence of s2 in s1
  - strtok(s, " "); //tokenize sentence s
  - memcpy(s1, s2, 5); // copies first 5B from s2 to s1
  - memmove(s, &s[5], 6); //first 5 chars moved to pos 6
  - memcmp(s1, s2, 4); //compares first 4 letters of s1 and s2 and returns -1, 0, 1
  - memchr(s, 'a', 2); //part of s which starts with 'a'
  - memset(s, 'b', 3); //write 'b' to first 3 positions of s
  - strerror(1); //prints error which has code 1
  - strlen(str); //length of string, excluding '\0'
  - **Some of these functions are not secure**

- **<stdlib.h>**
  - double d = strtod(str, &strPtr); //numerical part of str goes into d and rest into strPtr
  - long x = strtol(str, &strPtr, 0); // same for long
  - unsigned long int x = strtoul(str, *strPtr, 0); // same for unsigned long int