C-Tutorial

Class web: www.ccs.neu.edu/course/csu480

Outline

- IO
- Data Types & Variables
- Running and Debugging
- Language Basics
- Arrays & Strings
- Pointers
- Functions Ptrs
- Data Structure
- Memory Allocation
- Programming Tips
- C vs. C++
- Recommended

I/O

I/O Hello World Program

```
#include <stdio.h>
int main()
{
        char ch = ‘A’;
        char str[20] = “thisISaSTring”;
        puts(“Hello World”);
        printf(“This character is :%c , while the string says %s\n”, ch, str);
        return 0;
}
```

printf()

The printf() function can be instructed to print integers, floats and string properly.

- The general syntax is

```
printf( “format”, variables);
```

- An example

```
int stud_id = 5200;
char * name = “Mike”;
printf(“%s ‘s ID is %d \n”, name, stud_id);
```

I/O Hello World Program

Letter	Type of Matching Argument	Example	Output
%	none	printf("%%");	%
d, i	int	printf("%i", 17);	17
u	unsigned int (Converts to decimal)	printf("%u", 17u);	17
o	unsigned int (Converts to octal)	printf("%o", 17);	21
x	unsigned int (Converts to lower-case hex)	printf("%x", 26);	1a
X	unsigned int (Converts to upper-case hex)	printf("%X", 26);	1A
f, F	double	printf("%f", 3.14);	3.14
e, E	double	printf("%e", 31.4);	3.14E+01
g, G	double	printf("%g, %g", 3.14, 0.0000314);	3.14, 3.14e-05
a, A	double	printf("%a", 31.0);	0x1.fp+0
c	int	printf("%c", 65);	A
s	string	printf("%s", "Hello");	Hello
p	void*	int a = 1; printf("%p", &a);	0064FE00

- Format Identifiers

%d	decimal integers
%x	hex integer
%c	character
%f	float and double number
%s	string
%p	pointer

- How to specify display space for a variable?

```
printf(“The student id is %5d \n”, stud_id);
```

The value of stud_id will occupy 5 characters space in the print-out.

- Why “\n”

It introduces a new line on the terminal screen.

escape sequence

\b	backspace	\\	backslash
\n	newline	\?	question mark
\r	carriage return	\’	single quote
\t	horizontal tab	\”	double quote
\v	vertical tab	\xhh	hexadecimal number

I/O Hello World Program

```
#include <stdio.h>
int main()
{
    char ch;
    int x;
    char str[100];
    printf(“Enter any character then integer\n”);
    scanf(“%c %d”, &ch,&x);
    printf(“Entered character is %c \n”, ch);
    printf(“Enter any string ( upto 100 character ) \n”);
    scanf("%s", &str);
    printf(“Entered string is %s \n”, str);
}
```

Data Types

Data Types

Name	Description	Size*	Range*
char	Character or small integer	1 byte	signed: -128 to 127 unsigned: 0 to 255
short	Short integer	2 bytes	signed: -32768 to 32767 unsigned: 0 to 65535
int	Integer	4 bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
long	Long integer	4 bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
float	Floating point number	4 bytes	3.4e +/- 38 (7 digits)
double	Double precision floating point number	8 bytes	1.7e +/- 308 (15 digits)
long double	Long double precision floating point number	8 bytes	1.7e +/- 308 (15 digits)

Other Data Types

Type	Size
int8_t	8 bit signed integer.
uint8_t	8 bit unsigned integer.
int16_t	16 bit signed integer.
uint16_t	16 bit unsigned integer.
int32_t	32 bit signed integer.
uint32_t	32 bit unsigned integer.
int64_t	64 bit signed integer.
uint64_t	64 bit unsigned integer.

Running & Debuging

Running

- How to compile?

```
$ gcc hello.c –o hello
```

gcc	compiling command
hello.c	source file
hello	compiler-generated executable file

Note: the default output filename is “a.out”

Running

- How to execute?

```
./hello
```

“./ ” indicates the following file “hello” resides under the current directory.

- what if hello is not in the current directory?!

Debug

```
CXX = gcc
FLAGS = -g -Wall
main: main.cc
	${CXX} -o helo  hello.c
clean:
	rm -f main
```

```
make
gdb helo
```

Exercise One

- Use C Language under Linux to Do the following
  - Ask the user to input his Name
  - Ask the user to input two numbers
  - Show the user name and the division of the two numbers (approximate the results to the nearest 2 decimal points i.e. 1.2345463 = 1.23 , 1.348 = 1.35)
  - Try to play with the different notations
- Answer the following Questions
  - What happens if you send parameters less than needed to printf or scanf?
  - What happens if you send parameter of different types than the format to printf and scanf ?

Language Basics

Logical Operations

- What is “true” and “false” in C

  In C, there is no specific data type to represent “true” and “false”. C uses value “0” to represent “false”, and uses non-zero value to stand for “true”.

- Logical Operators

A && B	=>	A and B
A || B	=>	A or B
A == B	=>	Is A equal to B?
A != B	=>	Is A not equal to B?

A > B => Is A greater than B?

A >= B => Is A greater than or equal to B?

A < B => Is A less than B?

A <= B => Is A less than or equal to B?

Short circuiting

- Short circuiting means that we don't evaluate the second part of an AND or OR unless we really need to.

- Don’t be confused

  && and || have different meanings from & and |.

  & and | are bitwise operators.

- Some practices

Please compute the value of the following logical expressions?

```
int i = 10; int j = 15; int k = 15; int m = 0;
        if( i < j && j < k)        =>
        if( i != j || k < j)       =>
        if( j<= k || i > k)        =>
        if( j == k && m)           =>
        if(i)                      =>
        if(m || j && i )           =>
```

```
int i = 10; int j = 15; int k = 15; int m = 0;
    if( i < j && j < k)      =>   false
    if( i != j | | k < j)    =>   true
    if( j<= k | | i > k)     =>   true
    if( j == k && m)         =>   false
    if(i)                    =>   true
    if(m | | j && i )        =>   true
```

Did you get the correct answers?

for <==> while

```
for (expression1;
     expression2;
     expression3){

          statement…

 }
```

equals

```
expression1;
while (expression2)
{
statement…;
expression3;
}
```

- An example

```
int x;
for (x=0; x<3; x++)
{
    printf("x=%d\n",x);
}
```

First time: x = 0;

Second time: x = 1;

Third time: x = 2;

Fourth time: x = 3; (don’t execute the body)

Variable Definition vs Declaration

Definition	Tell the compiler about the variable: its type and name, as well as allocated a memory cell for the variable
Declaration	Describe information “about" the variable, doesn’t allocate memory cell for the variable

```
extern int var;
int main(void)
{

 return 0;
}
```

```
#include “something.h”
extern int var;
int main(void)
{
 var = 10;
 return 0;
}
```

```
extern int var;
int main(void)
{
 var = 10;
 return 0;
}
```

```
extern int var = 10;
int main(void)
{
 var = 10;
 return 0;
}
```

Functions

Functions are easy to use; they allow complicated programs to be broken into small blocks, each of which is easier to write, read, and maintain. This is called modulation.

- How does a function look like?

```
returntype function_name(parameters…)
{
    local variables declaration;
    function code;
    return result;
}
```

- Sample function

```
int addition(int x, int y)
 {
      int add;
      add = x + y;
      return add;
 }
```

- How to call a function?

```
 int result;
 int i = 5, j = 6;
 result  = addition(i, j);
```

Array / Strings / Pointers

Arrays & Strings

- Arrays

```
int ids[50];
char name[100];
int table_of_num[30][40];
```

- Accessing an array

```
ids[0] = 40;
i = ids[1] + j;
table_of_num[3][4] = 100;
```

Note: In C Array subscripts start at **0** and end one less than the array size. [0 .. n-1]

- Strings

Strings are defined as arrays of characters.

The only difference from a character array is, a symbol “\0” is used to indicate the end of a string.

For example, suppose we have a character array, char name[8], and we store into it a string “Dave”.

Note: the length of this string 4, but it occupies 5 bytes.

D	a	v	e	\0			

Strings

- Open example
- Stringss.c

Pointers

Pointer is the most beautiful (*ugliest*) part of C **“The International Obfuscated C Code Contest ”**

- What is a pointer?

A pointer is a variable which contains the address in memory of another variable.

In C we have a specific type for pointers.

- Declaring a pointer variable

```
 int * pointer;
char * name;
```

- How to obtain the address of a variable?

```
 int x = 0x2233;
 pointer = &x;
```

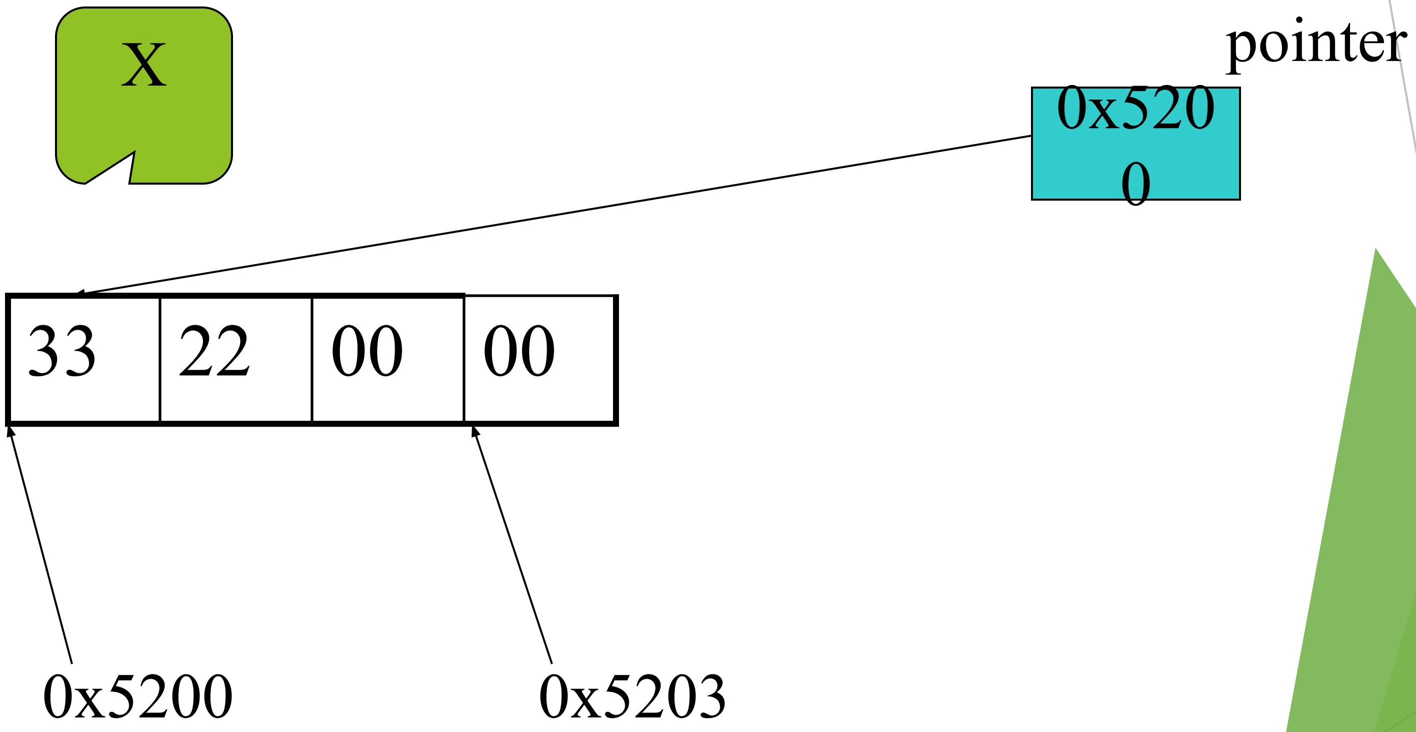
 where & is called address of operator.

- How to get the value of the variable indicated by the pointer?

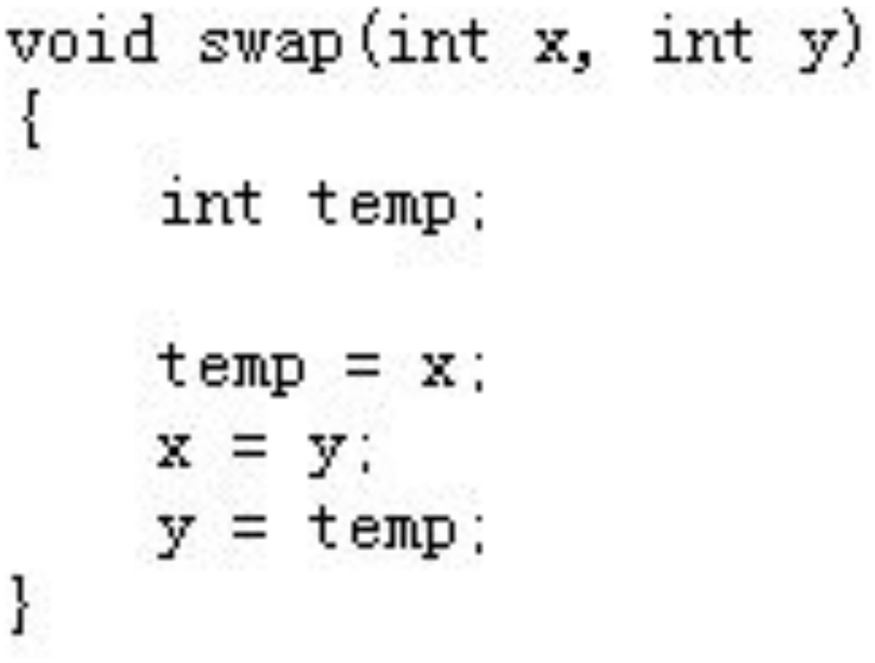
```
 int y = *pointer;
```

- What happens in the memory?

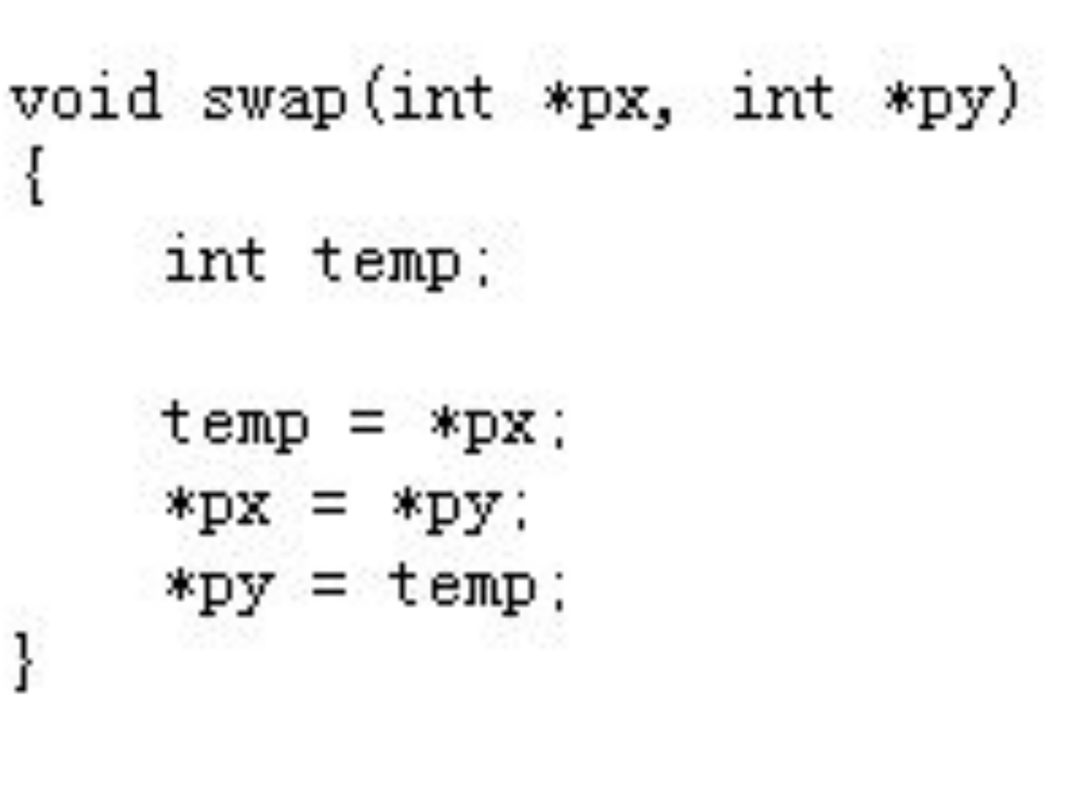
Suppose the address of variable x is 0x5200 in the above example, so the value of the variable pointer is 0x5200.



swap the value of two variables

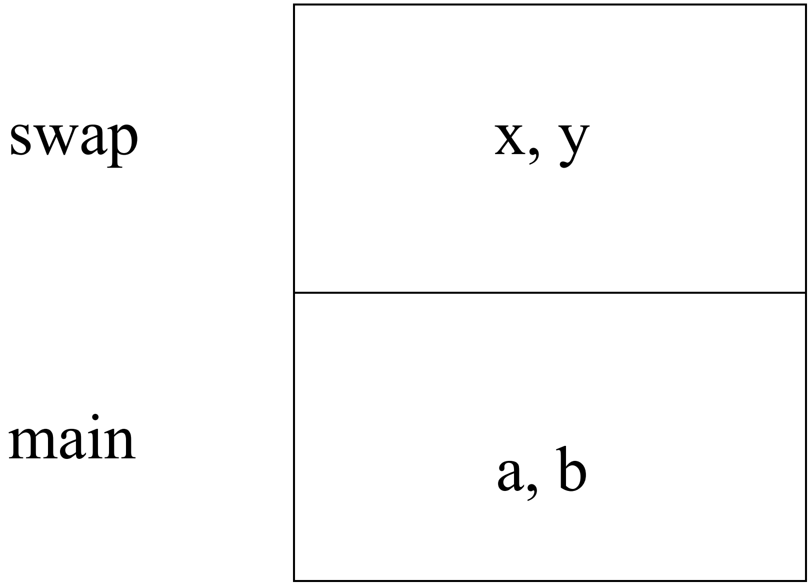


```
void swap(int x, int y)
{
    int temp;

    temp = x;
    x = y;
    y = temp;
}
```



```
void swap(int *px, int *py)
{
    int temp;

    temp = *px;
    *px = *py;
    *py = temp;
}
```

Why is the left one not working?



swap

x, y

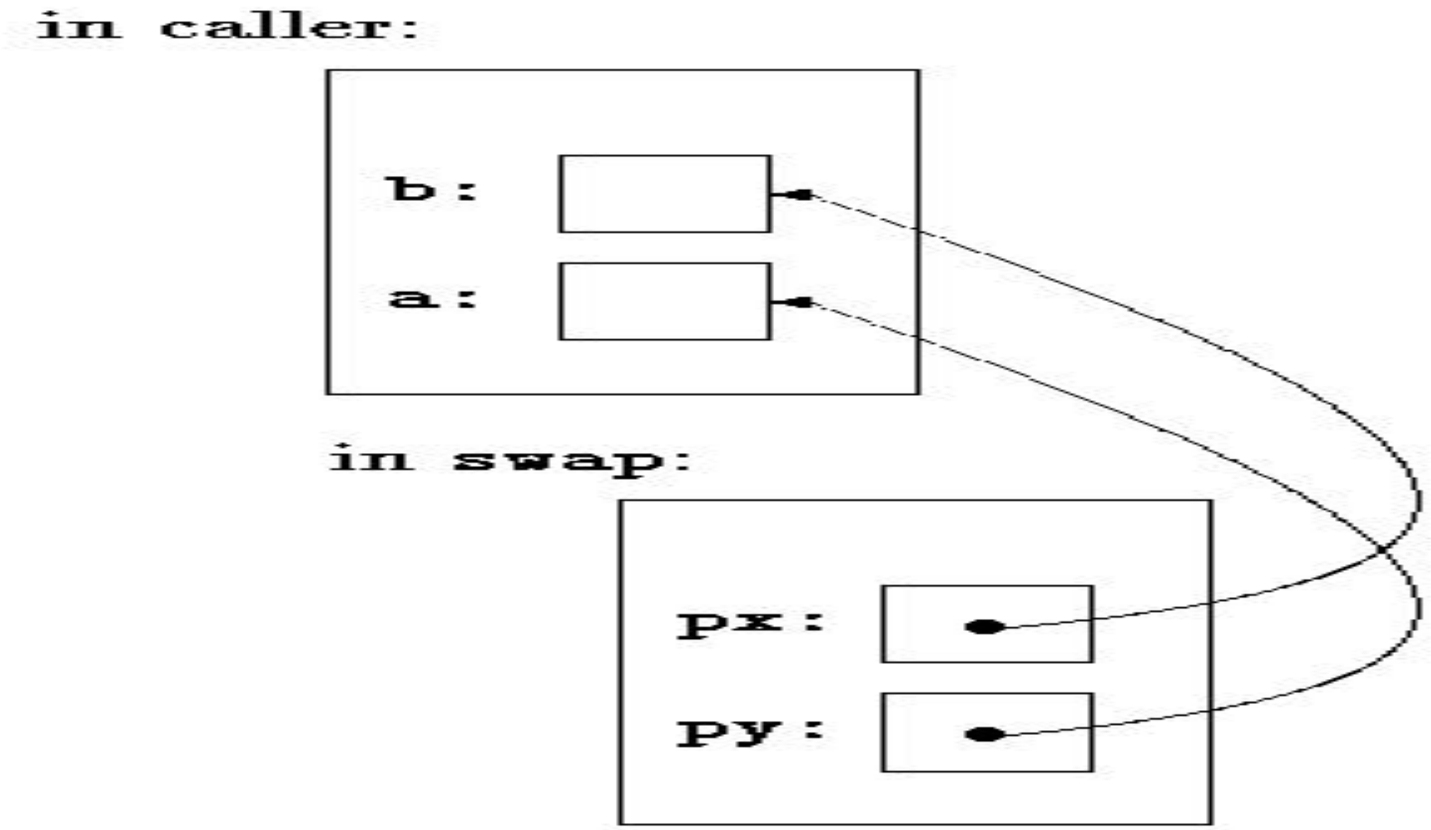
main

a, b

x, y, a, b are all local variables

call swap(a, b) in main

Why is the right one working?



- Pointers and Arrays

  Pointers and arrays are very closely linked in C.

  Array elements arranged in consecutive memory locations

- Accessing array elements using pointers

  int ids[50];

  int * p = &ids[0];

  p[i] <=> ids[i]

- Pointers and Strings

  A string can be represented by a char * pointer.

```
Char name[50];
name[0] = ‘D’;
name[1] = ‘a’;
name[2] = ‘v’;
name[3] = ‘e’;
name[4] = ‘\0’;
char * p = &name[0];
printf(“The name is %s \n”, p);
```

Note: The p represents the string “Dave”, but not the array name[50].

Command-Line Argument

In C you can pass arguments to main() function.

- main() prototype

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

  argc indicates the number of arguments

  argv is an array of input string pointers.

- How to pass your own arguments?

  ```
  ./hello 10
  ```

- What value is argc and argv?

Let’s add two printf statement to get the value of argc and argv.

```
#include <stdio.h>
int main(int argc, char * argv[]);)
{
        int i=0;
        printf("Hello World\n");
        printf(“The argc is %d \n”, argc);
        for(i=0; i < argc; i++){
           printf(“The %dth element in argv is %s\n”, i, argv[i]);
        }
        return(0);
}
```

- The output

```
The argc is 2
The 0th element in argv is ./hello
The 1th element in argv is 10
```

The trick is the system always passes the name of the executable file as the first argument to the main() function.

- How to use your argument?

Be careful. Your arguments to main() are always in string format.

Taking the above program for example, the argv[1] is string “10”, not a number. You must convert it into a number before you can use it.

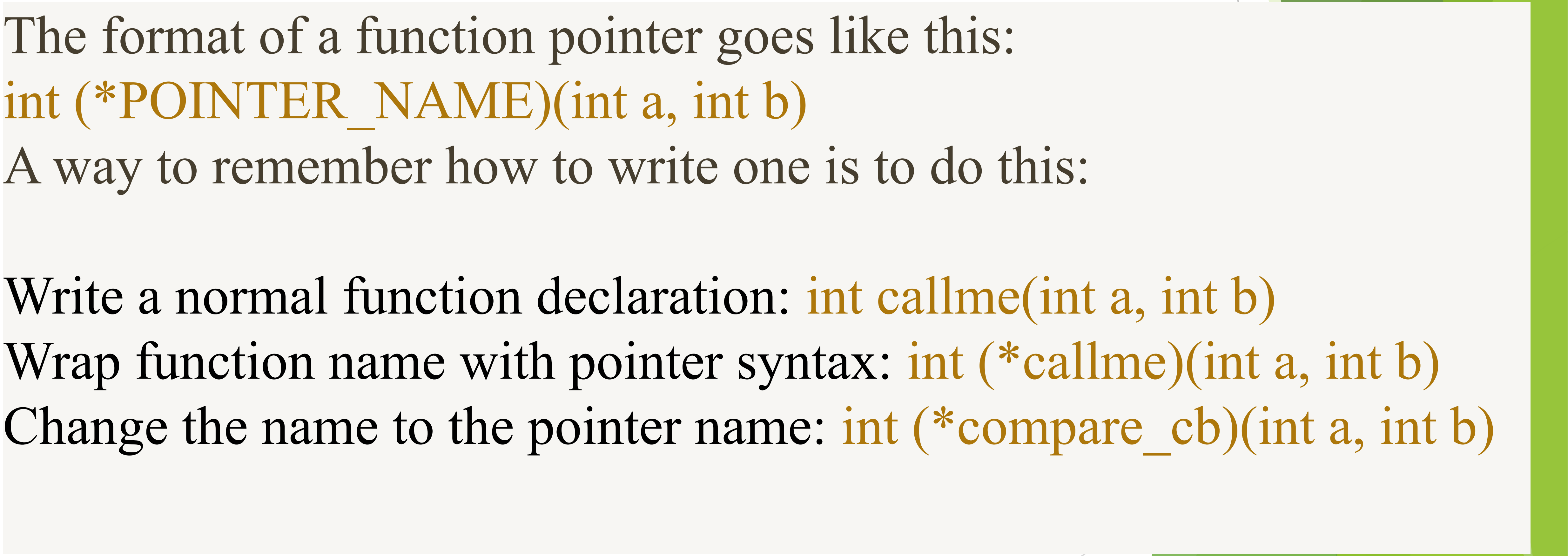
Pointer Arithematics

- Open example pointerArith.C

Exercise 2

- Compile and Run pointers.c
- Open pointer.c and explain why you got the results on the screen

Function Pointer



The format of a function pointer goes like this:

int (*POINTER_NAME)(int a, int b)

A way to remember how to write one is to do this:

- Write a normal function declaration: int callme(int a, int b)
- Wrap function name with pointer syntax: int (*callme)(int a, int b)
- Change the name to the pointer name: int (*compare_cb)(int a, int b)

Function Pointer

```
int (*tester)(int a, int b) = sorted_order;

printf("TEST: %d is same as %d\n", tester(2, 3),
sorted_order(2, 3));
```

Function Pointer

- Open example Ptrfn.c

What do you expect from this code

```
unsigned char *data = (unsigned char *)cmp;

for(i = 0; i < 25; i++) {
  printf("%02x:", data[i]); }

printf("\n");
```

Data Structure

A data structure is a collection of one or more variables, possibly of different types.

- An example of student record

```
struct stud_record{
        char name[50];
        int id;
        int age;
        int major;
        ……
};
```

- A data structure is also a data type

```
struct stud_record my_record;
struct stud_record * pointer;
pointer = & my_record;
```

- Accessing a field inside a data structure

```
my_record.id = 10;  “.”
    or
pointer->id = 10;          “->”
```

Memory Allocation

- Stack memory allocation

  Non-static local variable is an example of stack memory allocation.

  Such memory allocations are placed in a system memory area called the *stack*.

- Static memory allocation

  Static local variable and global variable require static memory allocation. Static memory allocation happens before the program starts, and persists through the entire life time of the program.

- Dynamic memory allocation

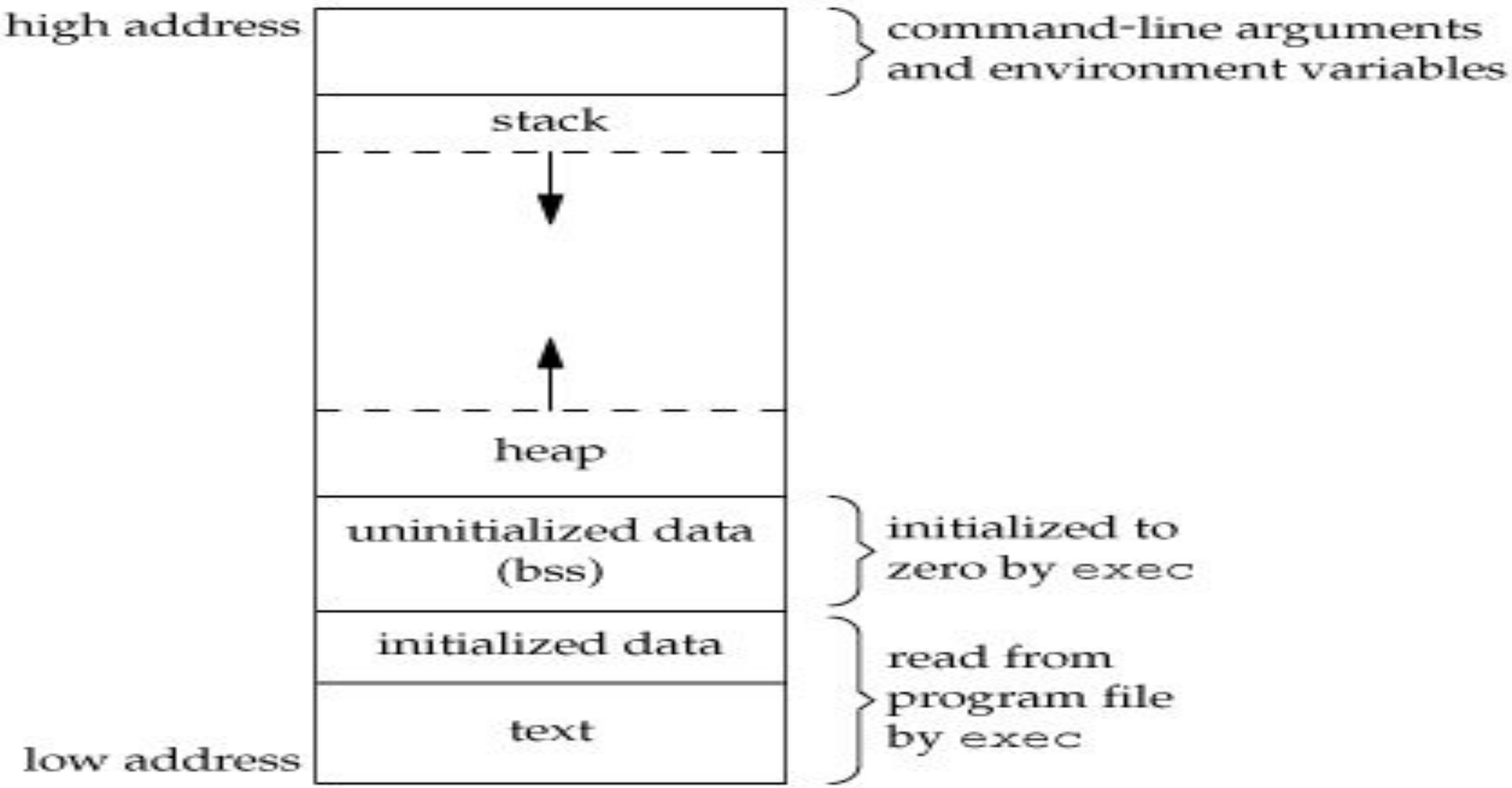
  It allows the program determine how much memory it needs *at run time*, and allocate exactly the right amount of storage.

  The region of memory where dynamic allocation and deallocation of memory can take place is called the heap.

  Note: the program has the responsibility to free the dynamic memory it allocated.

Memory arrangement

Figure 7.6. Typical memory arrangement



- Functions for the dynamic memory allocation

```
void *malloc(size_t number_of_bytes);  //allocates dynamic memory
size_t sizeof(type);     // returns the number of bytes of type
void free(void * p)    //      releases dynamic memory allocation
```

- An example of dynamic memory allocation

```
int * ids;    //id arrays
int num_of_ids = 40;
ids = malloc( sizeof(int) * num_of_ids);
      …….. Processing …...
free(ids);
```

- Allocating a data structure instance

```
struct stud_record * pointer;
pointer = malloc(sizeof(struct stud_record));
pointer->id = 10;
```

Never calculate the size of data structure yourself. The reason is the size of data types is machine-dependent. Give it to sizeof() function.

	size of int
32-bytes machines	32
64-bytes machines	64

Exercise

- Compile and run Structs.c
  - Why we freed (who->name) not just who
  - Try to make the assentation fails without changing the condition?
  - What happens when the assertion fails?
  - How to create a struct on a stack not heap?
  - How to initialize it using the x.y instead of x->y
  - How to pass a structure to other functions without using a pointer

Programming Tips

- Replacing numbers in your code with macros

  - don’t use magic numbers directly

  ```
  #define MAX_NAME_LEN      50;
  char name[MAX_NAME_LEN];
  ```
- Avoiding global variables

  - modulation is more important
- Giving variables and functions a nice name

  - a meaning name
- Don’t repeat your code

  - make a subroutine/function
- Don’t let the function body to exceed one screen

  - hard to debug

- Indenting your code (clearance)

```
 if(expression)
 {
     if(expression)
     {
             ……
     }
 }
```

- Commenting your code
- Don’t rush into coding. Plan first.
- Printing out more debugging information
- Using debugger (gdb)

C vs. C++

- C++ is a superset of C
- C++ has all the characteristics of C
- Using g++ to compile your source code

Resources recommended

- *The C Programming Language* by Brian Kernighan and Dennis Ritchie (known as 'K&R')
- http://c.learncodethehardway.org/book/

Thank you