**C Programming**

Basics Part 2

**Pointers**

A pointer is like a type of variable. It is a variable which stores an address in memory. Remember, C is a low-level language so you need to be careful when using pointers

Declaration

Int \*pointerInt;

Char \*letter;

Double \*price;

Int size = 15;

Int \*ptr;

Int = &size; // This stores the location in memory of the *size* variable

Printf(“address %x points to %d”, ptr, \*ptr);

// ptr = hex address, \*ptr = value associated with memory

Remember, all memory in the computer is labelled by hex numbers or binary. When you store a value in a variable, you are picking one of those hexadecimal labels of memory locations and storing some data to it. A pointer stores the location of that memory, not the value.

**Performing pointer arithmetic**

Int sizes[] = {15, 20, 30};

Int \*ptr = sizes; // pointer points to the start of the array

Ptr++; // points to the next value address of the array which could be a completely different address to the first

Ptr--; // points back to index 0

**Understanding relationships between pointers & array**

Arrays store data in a sequential block. Array notation is essentially pointer notation.

Pointers enable powerful access to memory.

Int main(void) {

Int data[] = {15, 20, 30, 45, 60, 80};

Int \*ptr = data;

Printf(“data[] starts at address %x with value %d\n”, ptr, \*ptr);

// calculate length of array

Int numElements = sizeof(data) / sizeof(int);

Printf(“There are %d elements in data\n”, numElements);

For (int I = 0; I < numElements; i++) {

Printf(“Address %x stores value %d\n”, ptr, \*ptr);

Ptr++;

}

// note, in for loop there is no use of the index of array

Return EXIT\_SUCCESS;

}

//sizeof() returns size of memory in bytes, an integer is 4 bytes, there are 6 elements in data array above, so 6(elements) \* 4(byte of 1 integer) = 24 bytes of total array. Divide by 4 to get the total length of array (for integers)

**Managing Memory using allocation and release**

Static arrays waste memory: int array[20]; guaranteed to allocate 20 integers. Whether you use 20 ints or not, that memory is set aside.

Dynamic arrays save memory by using a pointer:

Int \*array = malloc(20 \* sizeof(int)); // malloc = memory allocation

Dynamic arrays just point to the address, you don’t need to use it.

To release unused memory: *free(array);*

Int \*dynamicArray = malloc(20 \* sizeof(int));

\*dynamicArray = 10; // 0 index

dynamicArray[1] = 20; // 1 index

Printf(“The size of dynamic array: %d\n”, sizeof(dynamicArray));

// 8 – 2 elements \* 4. If this wasn’t a dynamic array, the size would be 80 bytes (20 \* 4)

Free(dynamicArray);

**Structures**

Structures allow us to combine multiple types of variables in to a single one. They enable the creation of complex variables.

Struct book {

Char title[MAX\_TITLE\_LENGTH];

Char author[MAX\_AUTHOR\_LENGTH];

Char publisher[MAX\_PUBLISHER\_LENGTH];

}

// book is now a data type you can use

Struct person {

Char firstName[20];

Char lastName[20];

Int age;

Char eyeColour[10];

Float height;

}

Struct person ali; // ali is a variable of type *person*

Ali.age = 28;

Ali.height = 5.9;

Strcpy(ali.firstName, “Ali”); // remember to include <string.h> to use this method

Void printStructure(struct person individual) {

Printf(“%s”, individual.lastName);

}

Struct person teams[10]; // array of 10 persons

Teams[0].age = 25; // the first person in teams age

**Union operations**

Similar to structures, unions enable the storage of different data types in the same location.

Union symbol {

Int number;

Char letter;

}

Union symbol value; // value is of type *symbol*

Value.number = 25;

Value.letter = “m”; // This has replaced value.number now as is stored in the same location and therefore overwriting it

Union symbol \*ptr;

Ptr = &value;

Char myLetter = ptr->letter;

Printf(“letter field %c\n”, myLetter);

Printf(“number field %d\n”, ptr->number);

**Files**

A file is a named portion of a storage area, e.g. harddrive

Two types of files:

1. Sequential – The programme accesses the data sequentially. You can read, write and close the file
2. Random access – Only allow access to the data at the point at which it should be read or written. You can open, close, advance to specific location, read and write the file

Void readEntireFile() {

Int ch;

FILE \*fp; // pointer to a file type

Fp = fopen(“path/to/file”, “r”); // read file

Ch = getc(fp);

While(ch !- EOF) { // built-in macro – End Of File

Putchar(ch);

Ch = getc(fp);

}

Fclose(fp);

}

// This will print the file out to the terminal

Void writeToFile() {

FILE \*fp;

Fp = fopen(“path/to/file”, “w”); // write to file

Char name[10] = “Ali”;

Fprintf(fp, “This message is for %s\n”, name);

Fclose(fp);

}

Void readAndVerify() {

Int ch;

FILE \*fp;

Char messageToWrite[] = “This message will be written”;

Char textBuffer[30];

Fp = fopen(“path/to/file”, “w+”); // read and write

Fwrite(messageToWrite, strlen(messageToWrite) + 1, 1, fp);

Fseek(fp, SEEK\_SET, 0)’ // go back to start of file

Fread(textBuffer, strlen(messageToWrite) + 1, 1, fp); // verify it works

}

Void addTextToLocation() {

FILE \*fp;

Fp = fopen(“/path/to/file”, “w+”);

Fputs(“not much to say today”, fp);

Fseek(fp, 8, SEEK\_SET); // begin writing at index 8

Fputs(“ to talk about”, fp);

Fclose(fp);

}

// output = not much to talk about

|  |  |
| --- | --- |
| R | Read file |
| W | Write to file |
| A | Append |
| A+ | Read and write in append mode |
| R+ | Read and write |

**User defined functions**

Very similar to JS – only a few points to note:

Two ways to pass variables to a function:

1. By value – argument values are copied into their function parameter
2. By reference – The argument addresses are copied in to function parameters

Void someFunction(int a, int b); // by value – someFunction(x, y);

Void anotherFunction(int \*a, int \*b); // by reference – anotherFunction(&x, &y);

Passing by value will not affect global variables with the same name, however, passing by reference will as you will be modifying the location at which a variable is set

// where x and y are global variables passed, and a and b are the parameter names in func

|  |  |  |
| --- | --- | --- |
|  | By value | By reference |
| Before function call | X = 0; y = 0; | X = 0; y = 0; |
| In function | A = 0; b = 0; | A = 0; b = 0; |
| In function modified | A = 111; b = 222; | A = 111; b = 222; |
| After function | X = 0; y = 0; | X = 111; y = 222 |

Passing array to function

Void function1(int array[]);

Void function2(int \*array);

Int list[5] = {1, 2, 3, 4, 5};

Function1(list);

Function2(list);

Returning

3 ways:

1. Explicit return using the *return* keyword
2. Modify global variable
3. Passing arguments by reference

Prototypes

Function prototypes allow for a function to be used before it is defined.

The prototype does not have to include the parameter names, e.g. *func1(double, double);*

**Macros**

Macros are a code fragment that has a label or name.

#define NUMBERS 1, 2, 3

Int x[] = {NUMBERS};

#define MAX(x, y) ((x) > (y) ? (x) : (y))

Note that macros don’t care about the data types

**Conditional directives**

#ifdef MACRO

//code

#endif

#if *expression*

//code

#endif

Constants.h

#define MAX\_NUMBERS 20

Main.c

#undef MAX\_NUMBERS

#define MAX\_NUMBERS 25

The above example shows that constants.h can be used in multiple files with the value 20, however, in main.c we can overwrite that macro by first undefining it, and then defining it again with a new value

Predefined macros

* \_\_FILE\_\_
* \_\_DATE\_\_
* \_\_TIME\_\_
* \_\_LINE\_\_

**Quiz**

**Part 1**

1. What is the difference between a pointer and a variable?
2. Create a pointer and print its address and value it points to
3. Create an array pointer of 5 items and assign values – print in console using a loop
4. Get the size of an array containing floats
5. Create a dynamic array remembering to free up space afterwards
6. Print out the size of the above array and compare it to its static array counterpart

**Part 2**

1. Create a struct and assign values to their properties
2. What is the difference between a struct and a union?
3. Create a union
4. What are the 2 different types of files?
5. Read and write to a file
6. Write to a file at a specific location and read to verify

**Part 3**

1. What is meant by passing by value vs by reference and how would you do this?
2. 3 ways to ‘*return’* from a function
3. What is a macro?
4. Create a macro. Check its existence first and undefine it before assigning a value to it