**C for Everyone: Programming Fundamentals Notes**

**Week 1**

**Chapter: Example-Marathon**

* void in int main(void){} means there are no arguments.

**Chapter: Simple input/output – fahrenheit**

* %d in scanf(“%d”,&fahrenheit) means integer

celsius = (fahrenheit – 32)/1.8 . Even the two variables are integer. After divided by a float type, the product will also become float type.

**Chapter: Simple input/output – miles**

* A normal machine is 4 byte machine, int type is 4 bytes.

For a longer bytes int, we can use long long, which is 8 bytes.

**Code Practise (Week 1)**

**Example-Circle code:** Circle and Area

**Example-Marathon:** The distance of a marathon in kilometers

**Simple input/output-fahrenheit:** Conversion of Fahrenheit to Celsius

**Simple input/output-miles:** Distance of a marathon in yards

**Assignment: Fix Dr. P’s mistake (week 1)**

**Week 2**

**Chapter: Comments**

* How the C compiler works?

First off, there is a preprocessor, which puts it in whatever code is necessary. There’s what’s called the tokenizer, and the tokenizer goes and looks through things and discard the comment.

* There are two comment styles. /\* \*/ and //

**Chapter: Keywords**

* Special words for the pre-processor like include may not be reserved as the keywords. Also, main is not a reserved/keyword, but a identifier.
* Search keywords in google by typing ANSI C/C++ keyword Table

**Chapter: Identifiers**

* Identifiers can be interpreted as the name of variables

**Chapter: Expressions and precedence (File included)**

* Precedence:

\*is higher precedent than binary + and binary –

But unary – and unary + (e.g. +8, and -8) has a higher precedence.

* Associativity:

a+b+c: from left to right -> ((a+b)+c)

a=b=c=3: from right to left -> (a=(b=(c=3)))

**Chapter: Expressions and evaluation**

c = ++a + b++; d += 5;

Output: a = 6, b = 8 b = c = 5 + 7 + 1 = 13, d = -12 + 5 = -7

First will perform ++a, as ++ is in the left hand side.

Then, perform a + b and finally b++

b will perform self-addition after the addition of c is done.

**Chapter: Fundamental types and sizeof**

* unsigned means rule out the negative numbers (only positive), this type is strictly be used in integers.
* long can be applied to both integers and to the doubles, which is used for large value number.
* 3 types of doubles: float, double, long double. Long double can store the largest value.

**Chapter: The integer and floating point types**

* Different representation of the same number:

%e or %E print as 1.23456e+00

%f print as 1.23456

%g or %G will represent like %e or %f depending on which one has the shortest representation on printing to the screen.

* float type should use %f in prinf()
* double and long float should use %lf in printf()

**Code Practise (Week 2)**

**Expression and evaluation:** Expression Evaluation

**Declarations:** Fundamental Types Declaration and Assignment

**Fundamental types and sizeof:** Fundamental Types sizeof operator

**The char type:** Char in c

**The int type:** Fundamental Types INT

**Assignment:** Fix Dr. P’s mistake (week 2)

**Week 3**

**Chapter: Logical operators, expressions, and short-circuit evaluation**

* short-circuit evaluation:

If one part of conditions already determines the result, we can use else to represent other conditions. It can save a lot of computations.

For example:

if (outside && weather)

printf("\nPlease use an umbrella.\n");

else

printf("\nDress normally.\n\n");

There is only one truth result, so we can use short-circuit evaluation.

* The comparison and relational operator such as >, <, && has a very low precedence. For example, (a<3) && (a>5) will first compare a and 3 and a and 5. Then, operates the && conditions and returns a 0.
* if(!a) equals if(a == 0)

**Chapter: The conditional statement if and if-else**

* if statement is done by determining the expression equals zero or non-zero value. For example: if (speed < 65). When speed > 65, expression is false and evaluates to 0. Otherwise, it evaluates to 1.

**Chapter: The iterative statement while**

* The while statement:

while(expression):

statement

If expression is false (0), we omit the statement.

If expression is true (1), we execute the statement and repeat until the expression is false.

* Syntax

e1;

while (e2)

{

statement;

e3;

}

In typical case: e1 is Initializer

e2 is terminating condition

e3 is increment/ decrement

**Chapter: The for statement and its while analog**

* The for statement:

for (i = 1, i <= 5, i++) :

statement

* Syntax

for(e1, e2, e3):

statement

In typical case: e1 is Initializer

e2 is terminating condition

e3 is increment/ decrement

**Chapter: for statement code example**

* The initializer is not necessary. For example:

for (; (c = getchar()) != EOF; total\_chars++)

There is no initializer, as the terminating condition gives sufficient information to know when should the loop be ended. ctrl+c to exit the program to reach EOF.

**Chapter: oddball operators-conditional and comma**

* c = (a < b)? a:b means if (a < b) then c = a; else c = b;
* Syntax

expression 1?expression 2:expression3

**Chapter: ternary-operator code example**

* In the code practice

speed = (speed <= 65) ? (65) : (speed <= 70) ? (70) : (90);

can be split into two sessions.

(speed <= 65) ? (65) : (speed <= 70) ? (70) : (90)

Means if speed <= 65 then speed = 65, else = the speed is determined by the second sessions.

Then (speed <= 70) ? (70) : (90);

means if speed <=70 then speed = 70, else = 90

**Chapter: Break and continue and switch**

* Example of switch:

switch (i)

{

case 1: a = 2 \* i; break;

case 2: a = 2 \* i; break;

default: a = 4 \* i; break;

}

break; is to prevent executing the next case.

* The labels in the switch can only be integer types like char, int, long.

**Code Practise (Week 3)**

**Logical operators, expressions, and short-circuit evaluation:**

logical operators and short circuit evaluation

**Logic operators quiz**

**The conditional statement if and if-else:** if and relations and flow of control

**while-cnt-char explained:** while-cnt-char-explained

**while-code-example:** Love you how much

**While loop questions**

**for statement code example:** demonstrate loop with for statement

**ternary-operator code example:** ternary operator and flow of control

**Switch questions**

**Assignment:** Fix Dr. P’s mistake (week 3)

**Cond-comma-ops quiz**

**Week 4**

**Chapter: Function definition**

* If a function return type is void, then the function does not need any return statement.

**Chapter: Function code example**

* void wrt\_very(int count) is the function declaration/signature of the function.

**Chapter: Function code example**

* return 0; means return of value zero to the operating system and the program finished in an expected or orderly way, a normal exit.
* If the return statement has an integer expression and the function return type is double, then the integer is converted to the double type and returned.

**Chapter: Function prototype**

* Prototype can be just with the parameter types or with names for the parameters.
* It allows a function declaration to precede a definition.
* For example:

double cube(double); // The prototype

int main() {

a = cube(3.5);

}

double cube(double x)

{

return x \* x \* x;

}

**Chapter: Function Prototype – code example**

* \t means horizontal tab.

**Chapter: Function variables-with call-by-value explained**

* call by value means the value of parameter passed into a function is copied and assigned to the local variable inside the function. The original parameter is never changed.
* For example:

int compute(int local\_n)

{

return local\_n\*local\_n

}

compute(n)

In this example, n is call by value. The value of n is assigned to local\_n and the variable n is not changed.

* The difference between call by value, call by address, and call by reference please check
  + <https://www.geeksforgeeks.org/difference-between-call-by-value-and-call-by-reference/>
  + <https://eeepage.info/call-by/>

**Chapter: Function definitions and scope rules**

* When variable is declared, it’s put on stack, which is the internal way that the compiler manages storage for variables.
* A variable declared in a block has a lifetime throughout this block, but disappears upon block exit.
* Variables in {} is local variable. For example:

int i = 0;

{int i = 1};

printf("%d", i);

0 will be printed, as I = 1 has already disappears after it exits the block.

**Chapter: Storage class code example**

* extern int reps = 0; extern means global variable that can use in different files.
* static int called = 0; static means global variable that can use in this cpp file.
* The key point in using a local to a function static variable is that its value is retained upon function exit.
* const int Limit = 10; const means the variable should not be changed.

**Chapter: Recursion-factorial code**

* function main() can be called recursively but it is not common to do so.
* Recursion use “call stack” to manage function call, LIFO (last in first out), so it uses more time and memory than loop
* When we don’t know the total number of the loop, recursion will be suitable to use.
* More info. About recursion:

<https://medium.com/learn-or-die/recursion-%E9%81%9E%E8%BF%B4-%E7%B0%A1%E4%BB%8B-php-%E7%AF%84%E4%BE%8B-6c3aba352772>

**Chapter: Recursion Fibonacci code**

* Recursive function gives a more logical easier to understand the program. However, it occupies more memory as it uses “call stack”. Hence, using while/for loop may be more efficient than recursive function.
* In the code practice, recursive Fibonacci shows one pitfall in using it versus an iterative calculation: Fibonacci has an exponential number of calls that can lots of stack space and extra time.

**Code Practise (Week 4)**

**Function code example:** demonstrate loops with while

**Function Prototype – code example:** function prototypes

**Storage class code example:** Storage class

**Recursion-factorial code:** simple recursion

**Recursion Fibonacci code:** Fibonacci

**Week 5**

**Chapter: Pointers and simple arrays**

* int data[100];

**type: int**

**identifier: data**

**size: 100 size of array should be const int.**

**Chapter: Initialize arrays**

* Few examples of array initialization below:
  + int data[5];
  + int data[5] = { 1,2,3,4,5 };
  + int data[5] = { 0 }; // All the elements be zero
  + int data[] = { 1,2,3 };
* String is not a data type in C language, it is an array of char.

|  |  |  |
| --- | --- | --- |
| a | b | \0 |

String “ab” is which is same as char str[] = "a,b";

a and b are 1 byte char, and \0 is exit character (an ASCII char) to indicate the end of the string.

**Chapter: What is a pointer**

* int data[4] = { 2,4,6,8 };

|  |  |  |
| --- | --- | --- |
| Index | Address | Element |
| data[0] | 7006 | 2 |
| data[1] | 7010 | 4 |
| data[2] | 7014 | 6 |
| data[3] | 7018 | 8 |

Array is pointing to a base address, in this case is 7006.

The address of each element = base address + 2\*length of data type

For data[2] = 7006 + 2\*4 = 7014.

* Pointer:
  + int\* p = &a;

\* means pointer, &a is the address of a, which will be stored in p.

To get the value stored in address of a. We can use\* p

to dereference the pointer.

**Chapter: Pointer code example**

* printf("sum is at %p, or %lu and is %lf\n", ptr\_to\_sum, ptr\_to\_sum, \*ptr\_to\_sum);

%p means hexadecimal, address is stored as hexadecimal format.

%lu is long unsigned.

The output is: **sum is at 00ECF9BC, or 15530428 and is 408.000000**

* int grades[SIZE] = { 78,67,92,83,88 };

printf("grades are at %lu to %lu\n", grades, grades + 5);

The output is: **grades are at 15530444 to 15530464**

As grades is a address of the array.

The base address of grades + 5\*4 = **15530464.**

* The base type of a pointer affects how it looks at memory.

For example, pointer to int and pointer to double are different. An int may be in 4 bytes and a double may be in 8 bytes. This difference can affect both how much memory is pointed at.

**Chapter: Call-by-reference simulated**

* Call-by-value is local, variables are unchanged when the function is exited
* Normal passing parameters are call by value but also can pass an address or a pointer, which is called call-by-reference.
* Call-by-value example:

void swap\_call\_by\_value(int i, int j)

{

int temp = i;

i = j; j = temp;

}

Value won’t changed !

* Call-by-reference example:

void swap(int \*i, int \*j)

{

int temp = \*i;

\*i = \*j;

\*j = temp;

}

The value of a and b can be swapped.

* Steps of call-by-reference:

1. Declare parameter as a pointer
2. Use dereference pointer in body
3. Pass in address in swap(&a,&b)

**Code Practise (Week 5)**

**Array-grade-code example:** array-grade-code example

**Pointer code example:** Use of pointers

**Call-by-reference simulated:** Swap

**Assignment**

**C for Everyone: Structured Programming Notes**

**Week 1**

**Chapter: Enums code**

* Enum is a form of int type

In the code practice:

enum day { sun, mon, tue, wed, thu, fri, sat }; // declare type

enum day is a data type, therefore we can create a function like this:

enum day next\_day(enum day d)

{ return(d + 1 % 7); }

* typedef enum day day; // Define types enum day

typedef can be used to define a new type and declare a new variable as that type. So we can use day directly in this example.

**Chapter: Preprocessor code**

* #define can create a macro that predefine a function or variable

For example:

#define FILL for(i=0;i<POPULATION;i++)\

data[i] = WEIGHT

We can call the for loop directly by using FILL.

**Chapter: Assert code**

* # include <assert.h>

assert(expression)

if expression is true, keep running

else call abort

Assert can be used to prove correctness.

* #define NDEBUG

can stop the assertion if the compiler we find that the program is running correctly.

**Code Practise (Week 1)**

**Enums code:** Enumerated Types

**Preprocessor code:** Use of Macro’s

**Assert code:** Use asserts within a trivial program

**Week 1 Quiz**