COMPUTERS. Primary memory: is memory directly accessible by the CPU (included RAM and ROM - not volatile) and memory with CPU itself. CPU consists of: registers, a decode unit, control unit, ALU and FPA. COMPLIERS. Assembly languages: are readable version of machine language, each manufacturer provides AL for its CPU. High-level languages: exhibit no connection to machine language, their instructions are more human-like and less machine-like. They are easy to read and relatively easy to run across diff platforms. Algorithms is step-by-step instructions to instruct a computer perform tasks by using programming languages (codes). Features of C: case-sensitive, quite-compact (has small # of keywords, closet to machine language in all high-level ones, faster and more powerful than most, used extensively in high-performance comp, Linux/Unix/Widows OS are written in C and C++. In Linux, use command [ gcc filename.c ] to compile, to run type [ a.out ]. TYPES: Typed programming language is a language that use types of data to define storing values in memory, which operations are admissible on values, defines the # of bytes available for storing and ranges of possible values. 4 arithmetic types are *char* (1byte)*, int* (short 2byte /long 4byte /long-long 8byte)*, float* (4byte)*, double* (long 8byte). Floating-point data store value into 3 distinct components (a sign, an exponent and a significant). Value ranges: range of *char, int* has limit by a #s, floating- point *(float, double)* limit by their component. EXPRESSIONS: Evaluate exprs: arithmetic, relational, logical (ALU on integer types and FLU on floating types). Arithmetic exprs consist of binary operands: integral (+,-,\*,/,%) and floating-point (all except %). Binary refer to (+,-,\*,/,%) and unary uses to assign a sign to # (positive / negative). Relational exprs: to evaluate a cond (1 true,0 false) They are ==, >, >=, <, <=, != (both integral and floating -point). Logical exprs: && (and), | | (or), ! (not true) ex. !(a && b) Shorthand assign’: same on integral and floating-point types. Binary operands are +=, -=, \*=, /=, %= (integral only). Unary operands are (++i, i++, --i, i--). Casting: to convert one data type to another (long double, double, float, long long, long, int, short, char). Ex (float) minutes. Mixed-type exprs: C ranks long double> double> float> long long>long> int>short>char. For operands with different types, assignment exprs: C will promote or narrow the right to the type on the left (ie, (float) cash = (int) change => *float* (promotion), if (int) change => (float) change => *int* (narrowing)). Arithmetic and relational exprs: C will promote the operand of lower type to a higher type before evaluating expr (ie 10 \* 1.0=10.0 ; 1.00 \* 1 = 1.00 ). Compound exprs: is an expr that contains an expr as one of its operands. C evaluates according to the *rules of precedence.* C operates on the one with highest rank first ( ++ -- (post fix) > ++ -- (prefix) > unary (+,-,&,!) > . LOGIC. Structured programming: sets of simple construct that has one entry point and one exit point. So that anyone can replace, modify or upgrade without affecting other parts in the program or causing bugs. A sequence is either a simple statement or a code block that enclosed in brackets { } to be run sequentially. Use pseudo-coding and flow-charts to achieve a clear, concise and better designed codes. Pseudo-code: is a set of shorthand notes in human language that shows in sequence of instructions to the solution. Flow-charts: is a set of conventional symbols connected by arrows to show the flow of control throughout the program (diamond for selection, rectangle for action, pentagon for iteration, oolong for start and exit).

Selection constructs:

1.Optional path (do sth only when cond is true) 2. Alternative paths: has binary selections *(if-else*) and multiple selections *(if – else if – else*): {compound conds (*if (cond1 && cond2) – else*): use logical to form more than one conds. Case-by-case *(switch*): use when we have a constant value for each case} 3.Conditional expr: short way to do alternative path (if-else, 2 cases only). It has the form ( *condition ? operand1 : operand2 )*. If the cond is true, the expr evaluates operand1. If the cond is false, operand2 is run.

Interation constructs:

While: runs its sequence as long as the cond is true. Do-while: runs the codes at least once and will continue as long as cond is true. For: groups initialization, test cond and change together.

Flags: flagging is a method of coding iteration within the single entry-and-exit rule of structured programming. Flags are variables that determine whether an iteration continues or stops.

Nested Constructs: one logic construct within another. 2 types: Nested selections (if-else within if-else). Dangling else: will be associate with the innermost if (ie. If if else, the else will go with the 2nd else). Nested loops: a loop in another loop. An inner loop will run full, then the outer loop run and form a full cycle of the inner loop again. STYLES: good program is easy to read and to debug. Identifiers should be self-descriptive, meaningful, short, concise identifiers (variables, functions, naming in general!). Practice good layout (spacing, indentation, line length, braces, comments). Magic numbers: prefer to values that is not generated from the program (could be math constants, default values), name them (ex const pi 3.14 OR #define pi = 3.14). TESTING. 2 types of Errors: Syntactic errors are ones that breaking the rules of the language. Semantic errors: ones that fail to implement the intent and meaning (use = instead of ==, dangling else, mismatch data types). Testing techniques: black box tests: are data driven, internal logics are hidden, output alone decide the success or failure of tests. Large possible values to test require narrow down into equivalent classes with boundary values. An equivalent class is a set where test wise any member is as good as another other. Errors occurs often at boundaries. White box testing: a complimentary test to black box test, it is logic driving, all internal logics are visible, it is path- oriented. We run each possible path once using a flow graphs. WB test criteria: run every statement, edge coverage, (compound) conds check, path coverage, iteration coverage. Debugging techniques: use IPE or command-line debugger. Walkthrough tables: consist of 2 parts: a record of every change in the value of every program variable, a listing of the output. Sufficient form: tables that show data type, variables, address, initial values and values that changing while on. ARRAYS: list of variables of the same type, we use arrays to perform same instructions on multiples variables. A list is called *arrays* and the variables in array are elements, we refer to any element by its index starts from 0. Arrays consist of ordered set of elements of common type that are stored contiguously in memory (without gap). C compliers can’t check if an index is within array, programmers need to ensure that codes call or include with size of arrays. Define array != initialize array. Parallel arrays is a convenient way to store tabular info. One array holds the key, while the other hold values. The arrays are parallel because the elements at the same index hold data that are related (ie student – marks, product – price). STRUCTS. A structure type is a collection of data that are not necessarily same types. We use struct to define a group of variables as a single object. A type directs how C interpret stored data in memory: 2 types (char, int, float, double) types and derived type (everything else). Struct declaration: We like to declare a struct globally and store in a header file for easy modification, correction or changes. NOTE: there are no allocation of memory when we declare a struct! When we define an object, allocation of memory will take place. Initialization of struct is similar to a primitive type separate using { {a,b}, {b,c} } by groups. FUNCTIONS. Modular design is an approach in programming involves separating code into self-contained components that can be accessed multiple times from diff locations in a complete program. Modular design identifies which components can be developed separately. Each module consists a set of instructions that are related together. Design principles (in module): easy to upgrade, not too long, may be utilize more than once. For structure design (as a whole): each module has one entry and one exit point, highly cohesive, low coupling. Cohesion: describes the focus: highly cohesive module performs a single task and only that task (unrelated tasks should be left out). Coupling describes the degree of interrelatedness of a module with other modules. The less info passes between modules the better. A procedural programming language supports modular design through function syntax. Functions: transfer control between one another (or *call* other functions). Once it completes its task, it transfer control back to its *caller*. A function consists of a header and a body [ *type identifier (type parameter, …., type parameter) { //function body return x: }* ]. Type specifies the type of the return value or the function’s parameter. Identifier specifies the name of function. Parameter is a variable that holds data received from its caller. Special cases: 1.Void functions: a function that doesn’t return any values (return empty and can be omitted). 2.No parameters: a function that doesn’t receive data passes from caller. Parameters are null (ex. void alphabet (void) { // } ). 3.Main main() function is a to which the operating system transfers control after loading the program into RAM. Main() returns a value of *int* tyoe (value 0) to the operating system once it has completed execution. A caller function will transfer control to the called function. Once its done, it returns control to caller. To call a function, we use [ *identifier (argument, … argument)* ] in which identifier is the name of the function being called, argument specifies values to be pass for it to utilize. An argument may be a constant, a variable, or an expression. Numbers of argument should match the number of parameters in function header. Pass by value: a function will pass a copy of arguments to called function for them to work on. Mixing types: if arguments types don’t match the parameter when we declare, the complier will (promote or narrow) argument types to the ones match in declaration. Use walkthrough tables for a modular program is an extension where we include each function and its handled variables. POINTERS.C makes sure values passed to function cannot be change outside of the function itself, but in cases when we need to change the values, we can do this through pass by address. Every variable occupies a unique address in memory when the program is run (this change when program shuts off and reruns). A variable that holds an address is called a pointer. To store an address, we define a pointer type and assign address to that pointer (variable’s address holder). It takes form: [ *type \*identifier;* ] ie int \*apple (an *int* that hold address of variable named apple). The \* operator is called *dereferencing* or *indirection* operator. Pointer types (char\*, short\*, int\*, long\*, float\*, etc) allows C to know what type of data is stored at that address and how to evaluate them. C store address in 4 bytes of memory. NULL address: each pointer has a special value called its null value. We can initialize a pointer to NULL before address is known [ int \*p = NULL; ]. Parameters: a function can receive addresses in its parameters this is called *pass by address.* This will allows us to access the original values and change them from the function. Multiple Return Values: C function syntax only allows them to return of a single value. If program design requires a function to return more than one value, we do so through parameter pointers that hold the addresses of the variables.

Size 6 size 7 Size 8 Size 9 1985