1. **Introduction**

C is a procedural programming language initially developed by Dennis Ritchie as a system programming language to write operating system. The main features:

1. Low-level access to memory
2. Simple se of keywords
3. Clean style

These features make C language suitable for writing operating system or compilers. Many later languages have borrowed syntax/features directly or indirectly from C. Like Java and PHP. C++ is nearly a superset of C few programs that may compile in C but not in C++.



1. **Beginning with C programming.** 
   1. **Structure of a C program**

C program has a rigid structure and can only be written in this structure to avoid compilation error.

Header #include<stdio.h>

Main() int main()

{

Variable int a = 10;

Body printf(“%d”, a);

Return return 0; }

1. **Parameter: Void**

A conforming implementation may provide more versions of main() but they must all have return type int. the int return is a way for a program to return a value to the “system” that invokes it. On some system that doesn’t provide such a facility the return value is ignored, but “void main()” is still illegal in both C/C++.



void loop\_1()

{ // This program works like a 'for' loop.

static int i = 5;

if (--i){

printf("%d\n", i);

loop\_1(10); // this parameter will be ignored.

}

}

void loop\_2(void)

{ // This program will not compile.

static int i = 5;

if (--i){

printf("%d\n", i);

loop\_2(10); // Because we cannot pass parameter into 'void'.

}

}

**Arrays basics**

An array has three parts:

1. The address of the first indexed variable
2. The type of the array
3. The size of the array

**Arrays as Function Argument**

When an array is used as an array argument to a function

The address of the first element is given to the function.

The base type of the array argument must match to the formal parameter.

The size of the array is still unknow to the function,

thus, we must have another argument telling the function the size of the array. Since the function does not know everything about the array (only the first part), it is not considered as call-by-reference parameter type.

**Entire arrays as function arguments**

This can be achieved by adding “[ ]” brackets.

void fill\_up(int array[], int size);

**Const parameter**

As a precaution, we can add “const” to the argument to tell the compiler that you do not intend to change the array argument.

void print\_result(const int array[], int size);

**Consistent use of the “const” Parameters**

When using const with the parameter in one function that calls another function without the const on the same parameter, the compiler will return an error message. Thus, one should use the modifier for both calling function and the called function,

**Functions that return an Array**

This can be achieved by using pointers. Make up after I read Chap 9

**Partially Filled Arrays**

When the exact size needed of an array is unknown, we can declare the array to be of the largest size the program could possibly need. Few points to keep in mind:

1. The program must keep track of how much the array is being used and never reference any indexed variable that has not been given a value.
2. Keeping the MAX as a const int global variable is a good idea when we intend to reuse the function multiple times.

**Dynamic arrays**

In C++, an array variable is a pointer that points to the first indexed variable of the array. To have a dynamic control of arrays, we can tell the system to reserve a number of spaces in the type of our choice and then create another pointer in the same type that points to the start of the reserved spot.

typedef int\* IntArrayPtr;

IntArrayPtr scores = new int[numUsed]; // Initialize new Dynamic array.

Remember that when you call new, the size of the dynamic array is given in the square bracket. If there is no bracket or the bracket is empty, the computer will allocate space enough only for one variable in the type of your choice.

In the end, also remember to free up the allocated space after use in order to save it for other programs.

delete [] scores; // Free-up space after use.

**Vectors**

Vectors can be thought of as arrays that can grow and shrink in length while your program is running. In C++, you cannot change the length of the array after you create it.

**Vector Basics**

Just like array, vector needs a base type when we declare it.

vector<int> v;

// changes the size of the Vector

// but does not destroys the elements

v.resize(5);

// Shrinks to the size

// till which elements are

// destroys the elements after 5

v.shrink\_to\_fit();

**Linked List**

**A linked list is a list constructed using pointers.** it is not fixed in size and can grow and shrink while the program is running. it’s fast when inserting new element but slow when searching elements in the list. Also requires more memory space for the pointer in the struct/class.

Dynamic variables that are either structs or classes normally have one or more member variables that are pointer variables which connect them to other dynamic variables.

**Nodes**

A structure consists of data and pointers like shown below.

struct ListNode

{

int count;

string item;

ListNode\* next;

};

typedef ListNode\* ListNodePtr;

and it needs space that we can allocate by using new operator

ListNodePtr head;

head = new ListNode;

/\* There are two ways of modifying the data in the node.

We can treat it like a pointer that points to the

variable in the struct. \*/

(\*head).count = 10;

/\* Or we can use the -> arrow operator, which combines the

actions of a dereferencing operator \* and a dot operator

to specify a member of a dynamic struct or object that is

pointed to by a given pointer. \*/

head->count = 12;

head->item = "egg";

head->next = NULL;

There are 4 steps for inserting a node at the Head of the list:

// 1. Create new temp\_ptr and new node.

ListNodePtr temp\_ptr;

temp\_ptr = new ListNode;

// 2. Add data in

cout << "Enter Item Name: ";

cin >> temp\_ptr->item;

cout << "Enter Item Quantity: ";

cin >> temp\_ptr->count;

// 3. New node points to the previous head node.

temp\_ptr->next = head;

// 4. Head pointer points to the new node.

head = temp\_ptr;

lengthList++;