Read and WRITE Operation using pointers

#include <stdio.h>

int main()

{

int a = 10;

float b = 3.14;

char c = 'a';

float n;

int \* pint = &a;

float \* pflt=&b;

char \* pchr = &c;

\*pflt = \*pflt + 10.0; // write operation performed

n = \*pflt; // Read operation

printf("n= %f \n",n);

return 0;

}

1. Write a C program that declares an integer pointer, initializes it to point to an integer variable, and prints the value of the variable using the pointer.

#include <stdio.h>

int main()

{

int a = 10;

int n;

int \* pint = &a;

n=\*pint;

printf("a = %d",n);

return 0;

}

1. Write a C program that declares an integer pointer, initializes it to point to an integer variable, and prints the value of the variable using the pointer.

#include<stdio.h>

int main()

{

float a = 3.14 , n;

float \*pflt = &a;

printf(" a = %f \n",a);

\*pflt = \*pflt + 12.0;

n= \*pflt;

printf("changed value is %f",n);

return 0;

}

1. Given an array of integers, write a function that takes a pointer to the array and its size as arguments. Use pointer arithmetic to calculate and return the sum of all elements in the array.

#include<stdio.h>

int main()

{

int size;

printf("enter the size of array :");

scanf("%d",&size);

int array[size];

printf("enter the array elements:");

for(int i=0;i<size;i++){

printf("elemnt %d ",i+1);

scanf("%d",&array[i]);

int sum = calculatesum(array,size);

printf("sum = %d",sum);

}

return 0;

}

int calculatesum(int \*array, int size){

int sum= 0;

for(int i=0;i<size;i++){

sum+=\*(array + i);

}

return sum;

}

1. Write a program that demonstrates the use of a null pointer. Declare a pointer, assign it a null value, and check if it is null before attempting to dereference it.

#include <stdio.h>

int main() {

int \*ptr = NULL;

if (ptr == NULL) {

printf("The pointer is null.\n");

} else {

printf("The value pointed to by the pointer is: %d\n", \*ptr);

}

return 0;

}

1. Create an example that illustrates what happens when you attempt to dereference a wild pointer (a pointer that has not been initialized). Document the output and explain why this leads to undefined behavior.

#include <stdio.h>

int main() {

int \*ptr;

printf(" wild pointer: %d\n", \*ptr);

return 0;

}

1. Implement a C program that uses a pointer to a pointer. Initialize an integer variable, create a pointer that points to it, and then create another pointer that points to the first pointer. Print the value using both levels of indirection.

#include <stdio.h>

int main() {

int a=10;

int \*pint1 = &a;

int \*pint2 = &\*pint1;

printf("001 a = %d\n",\*pint1);

printf("002 a = %d",\*pint2);

return 0;

}

1. Write a program that dynamically allocates memory for an array of integers using malloc. Populate the array with values, print them using pointers, and then free the allocated memory.

#include <stdio.h>

#include <stdlib.h>

int main() {

int \*arr;

int n, i;

printf("Enter the number of elements: ");

scanf("%d", &n);

arr = (int \*)malloc(n \* sizeof(int));

if (arr == NULL) {

printf("Memory allocation failed!\n");

return 1;

}

for (i = 0; i < n; i++) {

arr[i] = i \* 5;

}

printf("Array values: ");

for (i = 0; i < n; i++) {

printf("%d ", \*(arr + i));

}

printf("\n");

free(arr);

return 0;

}

1. Define a function that takes two integers as parameters and returns their sum. Then, create a function pointer that points to this function and use it to call the function with different integer values.

#include <stdio.h>

int sum(int a, int b) {

return a + b;

}

int main() {

int (\*func\_ptr)(int, int) = sum;

int result1, result2;

result1 = func\_ptr(5, 10);

result2 = func\_ptr(15, 25);

printf("The sum of 5 and 10 is: %d\n", result1);

printf("The sum of 15 and 25 is: %d\n", result2);

return 0;

}

1. Create two examples: one demonstrating a constant pointer (where you cannot change what it points to) and another demonstrating a pointer to constant data (where you cannot change the data being pointed to). Document your findings.

#include <stdio.h>

int main() {

int var1 = 10;

int var2 = 20;

int \*const ptr = &var1;

\*ptr = 30;

printf("Value of var1 after modification: %d\n", var1);

// Attempting to change the address the pointer points to will result in error

//ptr = &var2; // Uncommenting this line will cause a compilation error

return 0;

}

1. Write a program that compares two pointers pointing to different variables of the same type. Use relational operators to determine if one pointer points to an address greater than or less than another and print the results.

#include <stdio.h>

int main() {

int a = 10;

int b = 20;

int \*ptr1 = &a;

int \*ptr2 = &b;

printf("Address of a: %p\n", (void \*)ptr1);

printf("Address of b: %p\n", (void \*)ptr2);

if (ptr1 > ptr2) {

printf("ptr1 points to a higher address than ptr2.\n");

} else if (ptr1 < ptr2) {

printf("ptr1 points to a lower address than ptr2.\n");

} else {

printf("ptr1 and ptr2 point to the same address.\n");

}

return 0;

}

1. Write a program that declares a constant pointer to an integer. Initialize it with the address of an integer variable and demonstrate that you can change the value of the integer but cannot reassign the pointer to point to another variable.

#include<stdio.h>

int main(){

int a =10;

int b =20;

int \*const ptr = &a;

printf("001 a = %d\n",a);

\*ptr = 30;

printf("002 a =%d",a);

// ptr = &b;

return 0;

}

1. Create a program that defines a pointer to a constant integer. Attempt to modify the value pointed to by this pointer and observe the compiler's response.

#include<stdio.h>

int main(){

int a = 10;

int const \*ptr = &a;

printf("a=%d",\*ptr);

\*ptr = 30;

return 0;

}

1. Implement a program that declares a constant pointer to a constant integer. Show that neither the address stored in the pointer nor the value it points to can be changed.

#include<stdio.h>

int main(){

int a = 10;

int b = 20;

int const \*const ptr = &a;

printf("a=%d",\*ptr);

// \*ptr = 30;

// ptr = &b;

return 0;

}

1. Develop a program that uses a constant pointer to iterate over multiple integers stored in separate variables. Show how you can modify their values through dereferencing while keeping the pointer itself constant

#include <stdio.h>

int main() {

int a = 5, b = 10, c = 15;

int \*const ptr = &a; // ptr is a constant pointer, its address can't change

printf("001 a = %d, b = %d, c = %d\n", a, b, c);

\*ptr = 20;

b = 30;

c = 40;

printf("\nAfter modification:\n");

printf("a = %d, b = %d, c = %d\n", a, b, c);

return 0;

}

1. Implement a program that uses pointers and decision-making statements to check if two constant integers are equal or not, printing an appropriate message based on the comparison.

#include<stdio.h>

int main()

{

int const a =10;

int const b =20;

int const \*ptr1=&a;

int const \*ptr2=&b;

if(\*ptr1==\*ptr2){

printf("the integers are equal");

} else {

printf("the integers are not equal");

}

return 0;

}

1. Create a program that uses conditional statements to determine if a constant pointer is pointing to a specific value, printing messages based on whether it matches or not.

#include<stdio.h>

int main()

{

int a =10;

int \*const ptr = &a;

printf("001 a =%d\n",a);

if(\*ptr==10){

printf("pointing to 10");

} else {

printf(" Not pointing to 10");

}

return 0;

}

1. Write a program that declares two constant pointers pointing to different integer variables. Compare their addresses using relational operators and print whether one points to a higher or lower address than the other.

#include<stdio.h>

int main()

{

int a =10;

int b = 20;

int \*const ptr1 = &a;

int \*const ptr2 = &b;

printf("001 address of a =%p\n",ptr1);

printf("001 address of b =%p\n",ptr2);

if(ptr1>ptr2)

{

printf("address of a has higer addresss \n");

}

else if(ptr1<ptr2)

{

printf("address of b has higer addresss \n");

}else {

printf("address of a and b are same \n");

}

return 0;

}

Implement a program that uses a constant pointer within loops to iterate through multiple variables (not stored in arrays) and print their values.

#include<stdio.h>

int main()

{

int a =10;

int b = 20;

int c = 30;

int \*const ptr = &a;

printf(" a =%d\n",\*ptr);

printf(" b =%d\n",\*(ptr+1));

printf(" c =%d\n",\*(ptr+2));

return 0;

}

Develop a program that uses a constant pointer to iterate over several integer variables (not in an array) using pointer arithmetic while keeping the pointer itself constant.

#include<stdio.h>

int sum(int \*, int \*);

int main(){

int a = 10;

int b = 20;

int sumValue = 0;

printf("addresss of a = %p \n",&a);

printf("addresss of b = %p \n",&b);

sumValue = sum(&a,&b);

printf("sumvalue = %d",sumValue);

return 0;

}

int sum(int \*p, int \*q){

printf("p=%p \n",p);

printf("q=%p \n",q);

return 1;

}

**1. Machine Efficiency Calculation**

**Requirements:**

* Input: Machine's input power and output power as floats.
* Output: Efficiency as a float.
* Function: Accepts pointers to input power and output power, calculates efficiency, and updates the result via a pointer.
* Constraints: Efficiency = (Output Power / Input Power) \* 100.

#include <stdio.h>

void calculateEfficiency( float\*inputPower,float\*outputPower,float\*efficiency)

{

if(\*inputPower!=0){

\*efficiency = (\*outputPower/ \*inputPower) \* 100;

}else {

printf("input power cannot be zero");

}

}

int main()

{

float inputPower,outputPower,efficiency;

printf("enter the machines input poweer(in watts) :");

scanf("%f",&inputPower);

printf("enter the machines output poweer(in watts) :");

scanf("%f",&outputPower);

calculateEfficiency(&inputPower,&outputPower,&efficiency);

if(inputPower!=0){

printf("effiency = %.2f",efficiency);

}

return 0;

}

**2. Conveyor Belt Speed Adjustment**

**Requirements:**

* Input: Current speed (float) and adjustment value (float).
* Output: Updated speed.
* Function: Uses pointers to adjust the speed dynamically.
* Constraints: Ensure speed remains within the allowable range (0 to 100 units).

#include <stdio.h>

// Function to adjust the speed

void adjustSpeed(float \*currentSpeed, float adjustment) {

\*currentSpeed += adjustment;

if (\*currentSpeed < 0) {

\*currentSpeed = 0;

} else if (\*currentSpeed > 100) {

\*currentSpeed = 100;

}

}

int main() {

float currentSpeed, adjustment;

// Input the current speed and adjustment value

printf("Enter the current speed of the conveyor belt (0 to 100 units): ");

scanf("%f", &currentSpeed);

printf("Enter the adjustment value: ");

scanf("%f", &adjustment);

// Adjust the speed

adjustSpeed(&currentSpeed, adjustment);

printf("The updated speed of the conveyor belt is: %.2f units\n", currentSpeed);

return 0;

}

**3. Inventory Management**

**Requirements:**

* Input: Current inventory levels of raw materials (array of integers).
* Output: Updated inventory levels.
* Function: Accepts a pointer to the inventory array and modifies values based on production or consumption.
* Constraints: No inventory level should drop below zero

#include <stdio.h>

void updateInventory(int \*inventory, int size, int \*changes) {

for (int i = 0; i < size; i++) {

inventory[i] += changes[i];

// Ensure inventory level does not drop below zero

if (inventory[i] < 0) {

inventory[i] = 0;

}

}

}

int main() {

int size;

// Input the size of the inventory array

printf("Enter the number of raw materials: ");

scanf("%d", &size);

int inventory[size];

int changes[size];

// Input the current inventory levels

printf("Enter the current inventory levels:\n");

for (int i = 0; i < size; i++) {

printf("Raw material %d: ", i + 1);

scanf("%d", &inventory[i]);

}

// Input the changes in inventory (positive for production, negative for consumption)

printf("Enter the changes in inventory (positive for production, negative for consumption):\n");

for (int i = 0; i < size; i++) {

printf("Change for raw material %d: ", i + 1);

scanf("%d", &changes[i]);

}

// Update the inventory levels

updateInventory(inventory, size, changes);

// Print the updated inventory levels

printf("Updated inventory levels:\n");

for (int i = 0; i < size; i++) {

printf("Raw material %d: %d\n", i + 1, inventory[i]);

}

return 0;

}

**4. Robotic Arm Positioning**

**Requirements:**

* Input: Current x, y, z coordinates (integers) and movement delta values.
* Output: Updated coordinates.
* Function: Takes pointers to x, y, z and updates them based on delta values.
* Constraints: Validate that the coordinates stay within the workspace boundaries.

#include <stdio.h>

// Function to update coordinates based on delta values and workspace constraints

void update\_coordinates(int \*x, int \*y, int \*z, int dx, int dy, int dz, int min, int max) {

// Update x, y, z based on deltas

\*x += dx;

\*y += dy;

\*z += dz;

// Validate and constrain coordinates within the workspace boundaries

if (\*x < min) \*x = min;

if (\*x > max) \*x = max;

if (\*y < min) \*y = min;

if (\*y > max) \*y = max;

if (\*z < min) \*z = min;

if (\*z > max) \*z = max;

}

int main() {

// Declare variables for current coordinates and deltas

int x, y, z;

int dx, dy, dz;

// Workspace boundaries

const int MIN\_BOUNDARY = 0;

const int MAX\_BOUNDARY = 100;

// Input current coordinates

printf("Enter current x, y, z coordinates (separated by spaces): ");

scanf("%d %d %d", &x, &y, &z);

// Input movement deltas

printf("Enter movement deltas (dx, dy, dz separated by spaces): ");

scanf("%d %d %d", &dx, &dy, &dz);

// Update coordinates using the function

update\_coordinates(&x, &y, &z, dx, dy, dz, MIN\_BOUNDARY, MAX\_BOUNDARY);

// Output updated coordinates

printf("Updated coordinates: x = %d, y = %d, z = %d\n", x, y, z);

return 0;

}

**5. Temperature Control in Furnace**

**Requirements:**

* Input: Current temperature (float) and desired range.
* Output: Adjusted temperature.
* Function: Uses pointers to adjust temperature within the range.
* Constraints: Temperature adjustments must not exceed safety limits.

#include <stdio.h>

// Function to adjust temperature

void adjustTemperature(float \*currentTemperature, float minTemperature, float maxTemperature) {

if (\*currentTemperature < minTemperature) {

\*currentTemperature = minTemperature;

} else if (\*currentTemperature > maxTemperature) {

\*currentTemperature = maxTemperature;

}

}

int main() {

float currentTemperature;

float minTemperature, maxTemperature;

// Input the current temperature

printf("Enter the current temperature: ");

scanf("%f", &currentTemperature);

// Input the desired temperature range

printf("Enter the minimum desired temperature: ");

scanf("%f", &minTemperature);

printf("Enter the maximum desired temperature: ");

scanf("%f", &maxTemperature);

// Adjust the temperature

adjustTemperature(&currentTemperature, minTemperature, maxTemperature);

printf("The adjusted temperature of the furnace is: %.2f degrees\n", currentTemperature);

return 0;

}

**6. Tool Life Tracker**

**Requirements:**

* Input: Current tool usage hours (integer) and maximum life span.
* Output: Updated remaining life (integer).
* Function: Updates remaining life using pointers.
* Constraints: Remaining life cannot go below zero.

#include <stdio.h>

// Function to update remaining life

void updateRemainingLife(int \*remainingLife, int usageHours, int maxLifeSpan) {

\*remainingLife = maxLifeSpan - usageHours;

// Ensure remaining life does not go below zero

if (\*remainingLife < 0) {

\*remainingLife = 0;

}

}

int main() {

int usageHours, maxLifeSpan, remainingLife;

printf("Enter the current tool usage hours: ");

scanf("%d", &usageHours);

printf("Enter the maximum life span of the tool: ");

scanf("%d", &maxLifeSpan);

// Update the remaining life

updateRemainingLife(&remainingLife, usageHours, maxLifeSpan);

printf("The updated remaining life of the tool is: %d hours\n", remainingLife);

return 0;

}

**7. Material Weight Calculator**

**Requirements:**

* Input: Weights of materials (array of floats).
* Output: Total weight (float).
* Function: Accepts a pointer to the array and calculates the sum of weights.
* Constraints: Ensure no negative weights are input.

#include <stdio.h>

// Function to calculate the sum of weights

float calculateTotalWeight(float \*weights, int size) {

float totalWeight = 0.0;

for (int i = 0; i < size; i++) {

if (weights[i] < 0) {

printf("Error: Negative weight detected. Please enter non-negative weights.\n");

return -1;

}

totalWeight += weights[i];

}

return totalWeight;

}

int main() {

int size;

printf("Enter the number of materials: ");

scanf("%d", &size);

float weights[size];

printf("Enter the weights of the materials:\n");

for (int i = 0; i < size; i++) {

printf("Weight of material %d: ", i + 1);

scanf("%f", &weights[i]);

}

float totalWeight = calculateTotalWeight(weights, size);

if (totalWeight != -1) {

printf("The total weight of the materials is: %.2f units\n", totalWeight);

}

return 0;

}

**8. Welding Machine Configuration**

**Requirements:**

* Input: Voltage (float) and current (float).
* Output: Updated machine configuration.
* Function: Accepts pointers to voltage and current and modifies their values.
* Constraints: Validate that voltage and current stay within specified operating ranges.

#include <stdio.h>

// Function to configure welding machine parameters

void configure\_welding\_machine(float \*voltage, float \*current, float min\_voltage, float max\_voltage, float min\_current, float max\_current) {

// Validate and constrain voltage

if (\*voltage < min\_voltage) {

\*voltage = min\_voltage;

} else if (\*voltage > max\_voltage) {

\*voltage = max\_voltage;

}

// Validate and constrain current

if (\*current < min\_current) {

\*current = min\_current;

} else if (\*current > max\_current) {

\*current = max\_current;

}

}

int main() {

// Variables for voltage and current

float voltage, current;

// Operating ranges

const float MIN\_VOLTAGE = 10.0;

const float MAX\_VOLTAGE = 50.0;

const float MIN\_CURRENT = 20.0;

const float MAX\_CURRENT = 300.0;

// Input voltage and current

printf("Enter welding machine voltage (V): ");

scanf("%f", &voltage);

printf("Enter welding machine current (A): ");

scanf("%f", &current);

// Configure welding machine parameters

configure\_welding\_machine(&voltage, &current, MIN\_VOLTAGE, MAX\_VOLTAGE, MIN\_CURRENT, MAX\_CURRENT);

// Output updated configuration

printf("Updated Welding Machine Configuration:\n");

printf("Voltage: %.2f V\n", voltage);

printf("Current: %.2f A\n", current);

return 0;

}

**9. Defect Rate Analyzer**

**Requirements:**

* Input: Total products and defective products (integers).
* Output: Defect rate (float).
* Function: Uses pointers to calculate defect rate = (Defective / Total) \* 100.
* Constraints: Ensure total products > defective products.

#include <stdio.h>

// Function to calculate defect rate

void calculateDefectRate(int totalProducts, int defectiveProducts, float \*defectRate) {

if (totalProducts <= defectiveProducts) {

printf("Error: Total products must be greater than defective products.\n");

\*defectRate = -1.0;

return;

}

\*defectRate = ((float)defectiveProducts / totalProducts) \* 100;

}

int main() {

int totalProducts, defectiveProducts;

float defectRate;

printf("Enter the total number of products: ");

scanf("%d", &totalProducts);

printf("Enter the number of defective products: ");

scanf("%d", &defectiveProducts);

calculateDefectRate(totalProducts, defectiveProducts, &defectRate);

if (defectRate != -1.0) {

printf("The defect rate is: %.2f%%\n", defectRate);

}

return 0;

}

**10. Assembly Line Optimization**

**Requirements:**

* Input: Timing intervals between stations (array of floats).
* Output: Adjusted timing intervals.
* Function: Modifies the array values using pointers.
* Constraints: Timing intervals must remain positive

#include <stdio.h>

// Function to adjust timing intervals

void adjustTimingIntervals(float \*intervals, int size) {

for (int i = 0; i < size; i++) {

if (intervals[i] < 0) {

printf("Error: Negative timing interval detected at index %d. Adjusting to 0.\n", i);

intervals[i] = 0;

}

}

}

int main() {

int size;

printf("Enter the number of timing intervals: ");

scanf("%d", &size);

float intervals[size];

printf("Enter the timing intervals between stations (in seconds):\n");

for (int i = 0; i < size; i++) {

printf("Timing interval %d: ", i + 1);

scanf("%f", &intervals[i]);

}

adjustTimingIntervals(intervals, size);

printf("Adjusted timing intervals:\n");

for (int i = 0; i < size; i++) {

printf("Timing interval %d: %.2f seconds\n", i + 1, intervals[i]);

}

return 0;

}

**11. CNC Machine Coordinates**

**Requirements:**

* Input: Current x, y, z coordinates (floats).
* Output: Updated coordinates.
* Function: Accepts pointers to x, y, z values and updates them.
* Constraints: Ensure updated coordinates remain within machine limits

#include <stdio.h>

// Function to update CNC machine coordinates

void update\_coordinates(float \*x, float \*y, float \*z, float dx, float dy, float dz, float min\_limit, float max\_limit) {

\*x += dx;

\*y += dy;

\*z += dz;

if (\*x < min\_limit) \*x = min\_limit;

if (\*x > max\_limit) \*x = max\_limit;

if (\*y < min\_limit) \*y = min\_limit;

if (\*y > max\_limit) \*y = max\_limit;

if (\*z < min\_limit) \*z = min\_limit;

if (\*z > max\_limit) \*z = max\_limit;

}

int main() {

float x, y, z;

float dx, dy, dz;

const float MIN\_LIMIT = 0.0;

const float MAX\_LIMIT = 500.0;

printf("Enter current x, y, z coordinates (separated by spaces): ");

scanf("%f %f %f", &x, &y, &z);

printf("Enter movement deltas (dx, dy, dz separated by spaces): ");

scanf("%f %f %f", &dx, &dy, &dz);

update\_coordinates(&x, &y, &z, dx, dy, dz, MIN\_LIMIT, MAX\_LIMIT);

printf("Updated CNC Machine Coordinates:\n");

printf("x = %.2f, y = %.2f, z = %.2f\n", x, y, z);

return 0;

}

**12. Energy Consumption Tracker**

**Requirements:**

* Input: Energy usage data for machines (array of floats).
* Output: Total energy consumed (float).
* Function: Calculates and updates total energy using pointers.
* Constraints: Validate that no energy usage value is negative.

#include <stdio.h>

// Function to calculate total energy consumed

float calculateTotalEnergy(float \*energyUsage, int size) {

float totalEnergy = 0.0;

for (int i = 0; i < size; i++) {

if (energyUsage[i] < 0) {

printf("Error: Negative energy usage value detected at index %d. Please enter non-negative values.\n", i);

return -1.0; // error

}

totalEnergy += energyUsage[i];

}

return totalEnergy;

}

int main() {

int size;

printf("Enter the number of machines: ");

scanf("%d", &size);

float energyUsage[size];

printf("Enter the energy usage data for each machine (in kilowatt-hours):\n");

for (int i = 0; i < size; i++) {

printf("Energy usage for machine %d: ", i + 1);

scanf("%f", &energyUsage[i]);

}

// Calculate the total energy consumed

float totalEnergy = calculateTotalEnergy(energyUsage, size);

if (totalEnergy != -1.0) {

printf("The total energy consumed by all machines is: %.2f kWh\n", totalEnergy);

}

return 0;

}

**13. Production Rate Monitor**

**Requirements:**

* Input: Current production rate (integer) and adjustment factor.
* Output: Updated production rate.
* Function: Modifies the production rate via a pointer.
* Constraints: Production rate must be within permissible limits.

#include <stdio.h>

#define MIN\_PRODUCTION\_RATE 0

#define MAX\_PRODUCTION\_RATE 1000

void updateProductionRate(int \*productionRate, int adjustment) {

\*productionRate += adjustment;

if (\*productionRate < MIN\_PRODUCTION\_RATE) {

\*productionRate = MIN\_PRODUCTION\_RATE;

} else if (\*productionRate > MAX\_PRODUCTION\_RATE) {

\*productionRate = MAX\_PRODUCTION\_RATE;

}

}

int main() {

int productionRate, adjustment;

printf("Enter the current production rate: ");

scanf("%d", &productionRate);

printf("Enter the adjustment factor: ");

scanf("%d", &adjustment);

updateProductionRate(&productionRate, adjustment);

printf("The updated production rate is: %d units\n", productionRate);

return 0;

}

**14. Maintenance Schedule Update**

**Requirements:**

* Input: Current and next maintenance dates (string).
* Output: Updated maintenance schedule.
* Function: Accepts pointers to the dates and modifies them.
* Constraints: Ensure next maintenance date is always later than the current date.

#include <stdio.h>

#include <string.h>

// Function to update maintenance schedule

void updateMaintenanceSchedule(char \*currentDate, char \*nextDate, const char \*newNextDate) {

if (strcmp(newNextDate, currentDate) > 0) {

strcpy(nextDate, newNextDate);

printf("Next maintenance date updated successfully.\n");

} else {

printf("Error: The new next maintenance date must be later than the current date.\n");

}

}

int main() {

char currentDate[11], nextDate[11], newNextDate[11];

printf("Enter the current maintenance date (YYYY-MM-DD): ");

scanf("%10s", currentDate);

printf("Enter the next maintenance date (YYYY-MM-DD): ");

scanf("%10s", nextDate);

printf("Enter the new next maintenance date (YYYY-MM-DD): ");

scanf("%10s", newNextDate);

updateMaintenanceSchedule(currentDate, nextDate, newNextDate);

printf("Current maintenance date: %s\n", currentDate);

printf("Next maintenance date: %s\n", nextDate);

return 0;

}

**15. Product Quality Inspection**

**Requirements:**

* Input: Quality score (integer) for each product in a batch.
* Output: Updated quality metrics.
* Function: Updates quality metrics using pointers.
* Constraints: Ensure quality scores remain within 0-100.

#include <stdio.h>

// Function to update quality metrics

void updateQualityMetrics(int \*qualityScores, int size) {

for (int i = 0; i < size; i++) {

if (qualityScores[i] < 0) {

qualityScores[i] = 0;

} else if (qualityScores[i] > 100) {

qualityScores[i] = 100;

}

}

}

int main() {

int size;

// Input the number of products in the batch

printf("Enter the number of products in the batch: ");

scanf("%d", &size);

int qualityScores[size];

// Input the quality score for each product

printf("Enter the quality scores for each product (0-100):\n");

for (int i = 0; i < size; i++) {

printf("Quality score for product %d: ", i + 1);

scanf("%d", &qualityScores[i]);

}

updateQualityMetrics(qualityScores, size);

printf("Updated quality scores:\n");

for (int i = 0; i < size; i++) {

printf("Product %d: %d\n", i + 1, qualityScores[i]);

}

return 0;

}

**16. Warehouse Space Allocation**

**Requirements:**

* Input: Space used for each section (array of integers).
* Output: Updated space allocation.
* Function: Adjusts space allocation using pointers.
* Constraints: Ensure total space used does not exceed warehouse capacity.

#include <stdio.h>

#define MAX\_CAPACITY 1000

void adjustSpaceAllocation(int \*spaces, int size, int adjustment) {

int totalSpace = 0;

for (int i = 0; i < size; i++) {

spaces[i] += adjustment;

if (spaces[i] < 0) {

spaces[i] = 0;

}

totalSpace += spaces[i];

}

// Check if total space used exceeds warehouse capacity

if (totalSpace > MAX\_CAPACITY) {

printf("Error: Total space used exceeds warehouse capacity.\n");

for (int i = 0; i < size; i++) {

spaces[i] = (spaces[i] \* MAX\_CAPACITY) / totalSpace;

}

}

}

int main() {

int size, adjustment;

printf("Enter the number of sections in the warehouse: ");

scanf("%d", &size);

int spaces[size];

printf("Enter the space used for each section (in units):\n");

for (int i = 0; i < size; i++) {

printf("Space used for section %d: ", i + 1);

scanf("%d", &spaces[i]);

}

printf("Enter the adjustment value: ");

scanf("%d", &adjustment);

adjustSpaceAllocation(spaces, size, adjustment);

printf("Updated space allocation:\n");

for (int i = 0; i < size; i++) {

printf("Section %d: %d units\n", i + 1, spaces[i]);

}

return 0;

}

**17. Packaging Machine Settings**

**Requirements:**

* Input: Machine settings like speed (float) and wrap tension (float).
* Output: Updated settings.
* Function: Modifies settings via pointers.
* Constraints: Validate settings remain within safe operating limits.

#include <stdio.h>

#define MIN\_SPEED 0.0

#define MAX\_SPEED 100.0

#define MIN\_WRAP\_TENSION 0.0

#define MAX\_WRAP\_TENSION 50.0

void updateSettings(float \*speed, float \*wrapTension, float newSpeed, float newWrapTension) {

\*speed = newSpeed;

if (\*speed < MIN\_SPEED) \*speed = MIN\_SPEED;

if (\*speed > MAX\_SPEED) \*speed = MAX\_SPEED;

\*wrapTension = newWrapTension;

if (\*wrapTension < MIN\_WRAP\_TENSION) \*wrapTension = MIN\_WRAP\_TENSION;

if (\*wrapTension > MAX\_WRAP\_TENSION) \*wrapTension = MAX\_WRAP\_TENSION;

}

int main() {

float speed, wrapTension;

float newSpeed, newWrapTension;

printf("Enter the current speed: ");

scanf("%f", &speed);

printf("Enter the current wrap tension: ");

scanf("%f", &wrapTension);

printf("Enter the new speed: ");

scanf("%f", &newSpeed);

printf("Enter the new wrap tension: ");

scanf("%f", &newWrapTension);

updateSettings(&speed, &wrapTension, newSpeed, newWrapTension);

printf("Updated machine settings:\n");

printf("Speed: %.2f\n", speed);

printf("Wrap Tension: %.2f\n", wrapTension);

return 0;

}

**18. Process Temperature Control**

* **Requirements:**
* Input: Current temperature (float).
* Output: Adjusted temperature.
* Function: Adjusts temperature using pointers.
* Constraints: Temperature must stay within a specified range.

#include <stdio.h>

#define MIN\_TEMPERATURE 10.0

#define MAX\_TEMPERATURE 100.0

void adjustTemperature(float \*temperature, float adjustment) {

\*temperature += adjustment;

if (\*temperature < MIN\_TEMPERATURE) {

\*temperature = MIN\_TEMPERATURE;

} else if (\*temperature > MAX\_TEMPERATURE) {

\*temperature = MAX\_TEMPERATURE;

}

}

int main() {

float temperature, adjustment;

printf("Enter the current temperature: ");

scanf("%f", &temperature);

printf("Enter the adjustment value: ");

scanf("%f", &adjustment);

adjustTemperature(&temperature, adjustment);

printf("The adjusted temperature is: %.2f degrees\n", temperature);

return 0;

}

**19. Scrap Material Management**

* **Requirements:**
* Input: Scrap count for different materials (array of integers).
* Output: Updated scrap count.
* Function: Modifies the scrap count via pointers.
* Constraints: Ensure scrap count remains non-negative.

#include <stdio.h>

// Function to update scrap count

void updateScrapCount(int \*scrapCounts, int size) {

for (int i = 0; i < size; i++) {

if (scrapCounts[i] < 0) {

scrapCounts[i] = 0;

}

}

}

int main() {

int size;

printf("Enter the number of different materials: ");

scanf("%d", &size);

int scrapCounts[size];

printf("Enter the scrap count for each material:\n");

for (int i = 0; i < size; i++) {

printf("Scrap count for material %d: ", i + 1);

scanf("%d", &scrapCounts[i]);

}

updateScrapCount(scrapCounts, size);

printf("Updated scrap count:\n");

for (int i = 0; i < size; i++) {

printf("Material %d: %d\n", i + 1, scrapCounts[i]);

}

return 0;

}

**20. Shift Performance Analysis**

* **Requirements:**
* Input: Production data for each shift (array of integers).
* Output: Updated performance metrics.
* Function: Calculates and updates overall performance using pointers.
* Constraints: Validate data inputs before calculations.

#include <stdio.h>

// Function to validate data inputs

int validateData(int \*data, int size) {

for (int i = 0; i < size; i++) {

if (data[i] < 0) {

return 0;

}

}

return 1;

}

void calculatePerformance(int \*data, int size, float \*average, int \*total) {

\*total = 0;

for (int i = 0; i < size; i++) {

\*total += data[i];

}

\*average = (float)(\*total) / size;

}

int main() {

int size;

printf("Enter the number of shifts: ");

scanf("%d", &size);

int data[size];

printf("Enter the production data for each shift:\n");

for (int i = 0; i < size; i++) {

printf("Production data for shift %d: ", i + 1);

scanf("%d", &data[i]);

}

if (!validateData(data, size)) {

printf("Error: Invalid production data. All values must be non-negative.\n");

return 1;

}

float average;

int total;

calculatePerformance(data, size, &average, &total);

printf("Total production: %d units\n", total);

printf("Average production per shift: %.2f units\n", average);

return 0;

}

POINTERS----------------------------

/\*#include <stdio.h>

int main()

{

int a = 10;

int b = 20;

int \*ptr= &a;

\*ptr+= 25;

ptr=&b;

\*ptr += 25;

printf("a= %d and b = %d",a,b);

return 0;

}

\*/

/\*

#include <stdio.h>

int main()

{

int value;

int \*ptr=&value;

printf("enter number ");

scanf("%d",ptr);

printf("value is %d",value);

return 0;

}

\*/

/\*

#include<stdio.h>

int main(){

int \*ptr;

\*ptr = 5;// alaways avoid this kind of scenario

}

\*/

/\*

constant pointer

#include <stdio.h>

int main()

{

int a = 10;

int const \*ptr = &a;

\*ptr = 20;

return 0;

}

\*/

//try tp modifyng a constant value

#include <stdio.h>

int main()

{

int a = 10;

int b=20;

//int const \*ptr = &a;// dataa will be constant

int \*const ptr = &a;

//printf("001 value of ptr = %p\n",ptr);

ptr = &b;

//printf("001 value of ptr = %p",ptr);

\*ptr = 30;

printf("a = %d",a);

//\*ptr = 30;

return 0;

}