**VARIABLES SCANNING & CHANGES**

**🧩 Challenge 1: Scan an integer**

**Goal:** Read a single integer and print it.

#include <stdio.h>

int main() {

int x;

// TODO: Scan an integer and print it

return 0;

}

**🧩 Challenge 2: Scan a world**

**Goal :** Read a single word (no space) and print it.

#include <stdio.h>

int main() {

char word[100];

// TODO: Read a word and print it

return 0;

}

**🧩 Challenge 3: Scan a word and change it**

**Goal 2:** Read a single word (no space), and add “The Best” in front of it

int main() {

char word[100];

// TODO: Read a word (ex banana)

// TODO: Change the word to add The best at the beginning

// (ex The best banana)

// TODO: Print the result

return 0;

}

[HELP ? Check the LIB !](https://www.programiz.com/c-programming/library-function/string.h/strcat)

**🧩 Challenge 4: Scan many integers (fixed size)**

**Goal:** Read 5 integers into an array and print them.

#include <stdio.h>

int main() {

int nums[5];

// TODO: Read 5 integers into nums and print them

return 0;

}

**🧩 Challenge 5: Scan multiple words**

**Goal:** Read a line of text (up to 100 characters, including spaces).

#include <stdio.h>

int main() {

char line[101];

// TODO: Use fgets to read a line and print it

return 0;

}

**🧩 Challenge 6: Scan a Struct and change it**

**Goal :**  Read a Person with an int age and a name (no space)

After scanning the person, change the name to “CADT”

#include <stdio.h>

typedef struct {

int age;

char name[50];

} Person;

int main() {

Person p;

// TODO: Scan p.age and p.name

// TODO: Change the person name to CADT

// TODO: Print the result

return 0;

}

**🧩 Challenge 7: Scan a List of Structs (Fixed Size)**

**Goal:** Read n persons (name and age) and print them.

#include <stdio.h>

typedef struct {

int age;

char name[50];

} Person;

int main() {

int n;

Person people[100];

// TODO:

// 1. Read n

// 2. Read n people (name + age)

// 3. Print all people

return 0;

}

**🧩 Challenge 8: Scan a Dynamically Allocated Array of Integers**

**Goal:** Allocate memory for n integers, scan them, print them.

#include <stdio.h>

#include <stdlib.h>

int main() {

int n;

int \*arr;

// TODO:

// 1. Read n

// 2. Dynamically allocate memory for n integers

// 3. Scan values and print them

// 4. Free memory

return 0;

}

**🧩 Challenge 9: Scan a Dynamically Allocated Array of Structs**

**Goal:** Dynamically read n persons (name + age), print them, and free memory.

typedef struct {

int age;

char name[50];

} Person;

int main() {

int n;

Person \*people;

// TODO:

// 1. Read n

// 2. Allocate memory for n persons

// 3. Scan and print each

// 4. Free memory

return 0;

}

**🧩 Challenge 10: Nested Struct with List Inside**

**Goal:** A Classroom has a name and a dynamic list of Students.

typedef struct {

int id;

char name[50];

} Student;

typedef struct {

char name[50];

Student \*students;

int count;

} Classroom;

int main() {

Classroom c;

// TODO:

// 1. Read classroom name

// 2. Read number of students

// 3. Allocate and scan students

// 4. Print everything

// 5. Free memory

return 0;

}

**🧩 Challenge 11: Clone a Person to a Best Person**

**Goal:** Create a function to clone a person and The Best to its name.

typedef struct {

int age;

char name[50];

} Person;

Person getBestPerson(Person person) {

// TODO: Create a new version of the given person

// having its name ending with The Best

}

int main() {

Person normalPerson;

printf("Enter age and name:\n");

scanf("%d %s", &normalPerson.age, normalPerson.name);

// Clone the person as a best person

Person bestPerson = getBestPerson(normalPerson);

printf("Normal person name: %s\n", normalPerson.name);

printf("Best person name: %s\n", bestPerson.name);

return 0;

}

**🧩 Challenge 12: Update a Person to a Best Person**

**Goal:** Create a function to update a person to add The Best to its name.

typedef struct {

int age;

char name[50];

} Person;

void updateBestPerson(Person\* person) {

// TODO: Modify the given person

// to have its name ending with The Best

}

int main() {

Person person;

printf("Enter age and name:\n");

scanf("%d %s", &person.age, person.name);

// Modify the person as a best person

updateBestPerson(person);

printf("person name: %s\n", person.name);

return 0;

}

**🧩 Challenge 13: Implement a growable list of people**

**Goal:**

Build a system that:

Initialize the list.

Add a person. If the list is full, it should automatically resize using realloc.

Print all people.

(Bonus) Removing or updating a person by name or index

**Start Code**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct {

int age;

char name[50];

} Person;

typedef struct {

Person \*persons;

int capacity;

int size;

} PersonList;

// Global variable

// We use the global variable theList here to make it accessible from all functions without needing to pass a pointer each time.

PersonList theList;

void initList() {

theList.capacity = 2;

theList.size = 0;

theList.persons = malloc(theList.capacity \* sizeof(Person));

}

// ✅ Task: Implement this function

void add(Person p) {

}

// ✅ Task: Implement this function

void printList() {

}

void freeList() {

free(theList.persons);

}

int main() {

initList();

Person p1 = {20, "p1"};

add(p1);

Person p2 = {20, "p2"};

add(p2);

Person p3 = {20, "p3"};

add(p3);

printList();

freeList();

return 0;

}

**Output**

The program should output:

List capacity reallocated to 4

Personn p1 - Age= 20

Personn p2 - Age= 20

Personn p3 - Age= 20