Pentru a putea programa in C avem nevoie pe masina de un compilator de C

Cum verificam daca avem un compilator ?

**gcc –version**

Nu avem compilator ? Sa instalam unul

**sudo apt-get install gcc**

Sa scriem primul nostrum program in C

… facem un fisier cu extensia c din hello world din lab

**Cum il rulam**

O sa incercam cu modul classic in care rulam un sh (evident ca nu va functiona dar trebuie sa ne convingem 😊 ). Facem fisierul executabil si incercam cu ./hello\_world.c

Nu merge asa cum ne asteptam. Hai sa il compilam. Inainte de al compila o sa ii facem remove la permisiunea de executie pentru fisierul in c

**chmod -x hello\_world.c**

executam comanda

**gcc hello\_world.c -o hello\_world**

Ne uitam la fisiere din nou cu ls -ls sa vedem ce avem acum

In urma compilarii am observat ca avem un now fisierier executabil denumit asa cum am specificat in comanda gcc

Sa vedem ce se intampla cand avem erori in cod. Stricam putin codul si sa vedem ce se intampla la compilare

Acum as vrea sa tin toate erorile mele intr-un fisier si sa fac append la fiecare compilare. Cum fac asta (aici astept un raspuns de la grupa inainte de a le da comanda)

gcc hello\_world.c -o hello\_world 2>>compilation\_errors.txt

**int main –** acel int reprezinta status-ul de iesire din program. Zero indica faptul ca programul a fost executat cu success. In rest este un octet si poate fi definit oricum si de obicei o valoare diferite de zero indica faptul ca programul a intimpinat probleme

Ne intoarcem in pdf si povestim putin despre reserved keywords:

* **auto** : În programarea în limbajul C, cuvântul cheie "auto" este un specificator de tip (type specifier) folosit pentru a declara o variabilă locală într-o funcție. Acesta permite compilatorului să deducă automat tipul de date al variabilei în funcție de expresia de inițializare folosită la declarație. Totusi este de dorit sa nu fie folosit putand duce la probleme de intelegere a codului
* **break** - break este o instrucțiune în C care este utilizată pentru a ieși dintr-un bloc de instrucțiuni de control, cum ar fi un ciclu for sau un switch. De exemplu, o instrucțiune break într-un ciclu for va opri ciclul și va continua execuția programului de la următoarea instrucțiune din afara ciclului.
* **case** - case este o clauză în instrucțiunea switch în C, care indică ce instrucțiuni trebuie executate atunci când valoarea unei expresii se potrivește cu o anumită valoare. De exemplu, într-un bloc switch, o clauză case 1: indică faptul că urmează să fie executate instrucțiuni atunci când valoarea expresiei este 1.
* **char** - char este un tip de date în C care reprezintă un singur caracter. Variabilele de tip char pot reține un singur caracter, cum ar fi 'a', 'X', '$', etc.
* **const** - const este un calificator în C care indică faptul că o variabilă sau o valoare nu poate fi modificată. Variabilele declarate ca fiind const trebuie să fie inițializate la declarare și nu pot fi modificate ulterior în program.
* **default** - default este o clauză în instrucțiunea switch în C, care se execută atunci când nicio clauză case nu se potrivește cu valoarea expresiei. De obicei, clauza default este utilizată pentru a gestiona cazurile de eroare sau de valori neașteptate.
* **do** - do este o instrucțiune în C care inițiază un ciclu do-while. Corpul ciclului este executat cel puțin o dată, apoi se testează condiția de ieșire, iar ciclul se va executa atâta timp cât condiția este adevărată.
* **double** - double este un tip de date în C care reprezintă un număr cu virgulă mobilă cu precizie dublă față de tipul float. Variabilele de tip double pot reține numere reale cu o precizie de aproximativ 15-16 cifre semnificative.
* **else** - else este o clauză într-un bloc if în C, care se execută atunci când condiția din instrucțiunea if este falsă. De obicei, clauza else este utilizată pentru a gestiona cazurile alternative la condiția din blocul if.
* **enum** - enum este o structură de date în C care definește o enumerare de constante întregi. O enumerare poate fi definită pentru a oferi un set de nume semnificative pentru un grup de constante întregi asociate. De exemplu, un set de constante numerice pentru zilele săptămânii poate fi definit cu ajutorul unui tip enum.
* **extern** - extern este un calificator în C care se referă la o variabilă sau la o funcție care este definită într-un alt fișier sau în altă parte în program. De obicei, acest calificator este utilizat pentru a indica faptul că variabila sau funcția respectivă trebuie să fie căutată și definită în altă parte în program.
* **float** - float este un tip de date în C care reprezintă un număr cu virgulă mobilă cu precizie relativă. Variabilele de tip float pot reține numere reale cu o precizie de aproximativ 6-7 cifre semnificative.
* **for** - for este o instrucțiune de control în C care inițiază un ciclu de tipul for. Ciclul for permite executarea unui bloc de instrucțiuni de un număr finit de ori, specificat prin intermediul unei expresii de testare și a unei expresii de modificare.
* **goto** - goto este o instrucțiune în C care permite transferul controlului la o altă locație din program, identificată prin intermediul unei etichete. Utilizarea instrucțiunii goto poate face programul mai greu de urmărit și de înțeles și poate fi evitată prin utilizarea altor instrucțiuni de control în C.
* **if** - if este o instrucțiune de control în C care permite executarea unui bloc de instrucțiuni în funcție de o condiție. Dacă condiția este adevărată, blocul de instrucțiuni specificat în corpul instrucțiunii if este executat; dacă condiția este falsă, blocul de instrucțiuni este trecut.
* **int** - int este un tip de date în C care reprezintă un întreg cu semn. Variabilele de tip int pot reține numere întregi cu semn, cum ar fi -10, 0, 15 etc.
* **long** - long este un tip de date în C care reprezintă un întreg cu semn cu o dimensiune mai mare decât int. Variabilele de tip long pot reține numere întregi cu semn mai mari decât cele reținute de int, cum ar fi -2147483648 până la 2147483647.
* **register** - register este un calificator în C care poate fi aplicat unei variabile și indică faptul că variabila trebuie să fie stocată în registrul procesorului pentru o accesare mai rapidă. Utilizarea calificatorului register nu garantează faptul că variabila va fi stocată în registrul procesorului, ci este o sugestie către compilator.
* **return** - return este o instrucțiune în C care indică faptul că o funcție se termină și returnează o valoare. Valoarea returnată poate fi orice tip de date, inclusiv întregi, caractere, float-uri sau pointeri.
* **short** - short este un tip de date în C care reprezintă un întreg cu semn cu o dimensiune mai mică decât int. Variabilele de tip short pot reține numere întregi cu semn mai mici decât cele reținute de int, cum ar fi -32768 până la 32767.
* **signed** - signed este un calificator în C care poate fi aplicat unei variabile și indică faptul că variabila poate să conțină numere întregi cu semn. Dacă nu se specifică niciun calificator, se consideră implicit că variabila este de tip signed.
* **sizeof** - sizeof este o funcție în C care calculează dimensiunea unui tip de date sau a unei variabile în octeți. De exemplu, sizeof(int) va returna dimensiunea tipului de date int în octeți.
* **static** - static este un calificator în C care poate fi aplicat variabilelor și funcțiilor. O variabilă statică declarată într-o funcție va fi persistentă pe parcursul rulării programului, iar o funcție statică va fi vizibilă doar în fișierul în care este definită.
* **struct** - struct este un tip de date compus în C care permite definirea unei structuri de date care poate conține mai multe tipuri de date diferite. O structură poate fi definită prin intermediul unui șir de variabile și poate fi utilizată pentru a stoca informații asociate cu o entitate sau un obiect.
* **switch** - switch este o instrucțiune de control în C care permite executarea diferitelor blocuri de instrucțiuni în funcție de valoarea unei expresii. Blocurile de instrucțiuni sunt definite pentru fiecare valoare posibilă a expresiei și se execută când expresia are valoarea respectivă.
* **typedef** - typedef este un cuvânt cheie în C folosit pentru a defini un alias pentru un tip de date existent. De exemplu, un tip de date definit ca typedef int myInt va permite utilizarea variabilei myInt în loc de int în program.
* **unsigned** - unsigned este un calificator în C care poate fi aplicat unei variabile și indică faptul că variabila poate să conțină numere întregi fără semn. Dacă nu se specifică niciun calificator, se consideră implicit că variabila este de tip signed.
* **union** - union este un tip de date în C care permite definirea unei structuri de date care poate conține mai multe tipuri de date diferite, dar care utilizează aceeași zonă de memorie. Astfel, toți membrii unei uniuni împart aceeași zonă de memorie și numai unul dintre membri poate fi utilizat la un moment dat.
* **void** - void este un tip de date în C care indică faptul că o funcție nu returnează nicio valoare. Variabilele de tip void nu pot fi declarate, dar tipul de date void este util în definiția funcțiilor care nu returnează o valoare.
* **volatile** - volatile este un calificator în C care poate fi aplicat unei variabile și indică faptul că variabila poate fi modificată de către o altă parte a programului în afară de codul curent. Astfel, compilatorul nu va optimiza sau elimina codul care utilizează această variabilă, deoarece valoarea acesteia poate fi schimbată într-un alt moment.
* **while** - while este o instrucțiune de control în C care inițiază un ciclu while. Corpul ciclului este executat atâta timp cât condiția de testare este adevărată. Dacă condiția este falsă, executarea continuă cu următoarea instrucțiune după corpul ciclului.

**Sa ne uitam cum se defineste o variablila** (vom decomenta liniile din hello world)

Pointer

Un pointer în C este o variabilă specială care stochează adresa de memorie a unei alte variabile. Adresa de memorie este o valoare numerică care identifică locul unde este stocată variabila în memorie. Pointerul poate fi folosit pentru a accesa valoarea stocată în variabila respectivă prin intermediul adresei de memorie.

În C, declarația unui pointer se face prin specificarea tipului de date al variabilei căreia îi este asociat pointerul, urmat de operatorul \*, care indică faptul că variabila este un pointer. De exemplu, o variabilă de tip int poate fi declarată ca un pointer prin intermediul următoarei sintaxe:

int a = 10;

int \*ptr = &a; // atribuirea adresei variabilei 'a' pointerului 'ptr'

printf("%d", \*ptr); // afișarea valorii stocate la adresa specificată de pointer

Exemplu pointeri explicatie

**printf("Valoarea lui num: %d, Adresa lui num: %p\n", num, &num);**

Această instrucțiune afișează valoarea și adresa variabilei num. Șirul de formatare %d indică că valoarea va fi afișată ca un număr întreg, în timp ce %p afișează adresa variabilei.

**printf("Valoarea pointerului ptr\_int: %p, Valoarea la adresa pointerului ptr\_int: %d\n", ptr\_int, \*ptr\_int);**

Această instrucțiune afișează valoarea și adresa pointerului ptr\_int. Șirul de formatare %p indică că adresa va fi afișată în format hexadecimal, în timp ce %d afișează valoarea stocată la adresa indicată de pointer. Utilizarea operatorului \* pentru pointer (\*ptr\_int) permite accederea valorii stocate la adresa specificată de pointer

**Hai sa vedem argumentele**

Pentru asta o sa ne aducem lab2\_2.c

Explicatia

**#include <stdio.h>**

Această linie include fișierul de antet standard stdio.h, care conține definiții de funcții utilizate pentru citirea și scrierea datelor de intrare și ieșire în C.

**int main(int argc, char \*argv[], char \*env[])**

{

Această linie definește funcția principală a programului, main(), care primește trei argumente: argc - numărul de argumente din linia de comandă,

* argv - vectorul argumentelor de la linia de comandă și
* env - variabilele de mediu.

**if (argc == 1)**

**{**

**printf("Error: Insufficient number of arguments.\n");**

**printf("Usage: lab2\_2 arg\_1 arg\_2 ...\n");**

**return 1;**

**}**

Această instrucțiune verifică dacă numărul de argumente din linia de comandă este mai mic decât 2. Dacă acesta este cazul, se afișează un mesaj de eroare și se afișează modul de utilizare corectă a programului. Funcția return 1; indică că programul a fost încheiat cu o eroare.

**printf("Number of arguments (argc): %d\n", argc);**

Această instrucțiune afișează numărul de argumente primite în linia de comandă, utilizând variabila argc. %d este un șir de formatare care indică că o valoare întreagă va fi afișată în locul șirului de formatare.

**for (int i=0; i < argc; i++)**

**{**

**printf("Argument argv[%d]: %s\n", i, argv[i]);**

**}**

Această instrucțiune utilizează un ciclu for pentru a parcurge toate argumentele primite în linia de comandă, prin intermediul vectorului argv. Fiecare argument este afișat într-o linie separată, împreună cu poziția sa în vector. %s este un șir de formatare care indică că o valoare de tip șir de caractere va fi afișată în locul șirului de formatare.

**return 0;**

Această instrucțiune indică că programul s-a încheiat cu succes și se returnează valoarea 0.

**Sa vedem env variables**

Pentru asta o sa ne aducem lab2\_3.c

**int main(int argc, char \*argv[], char \*env[])**

**{**

Această linie definește funcția principală a programului, main(), care primește trei argumente: argc - numărul de argumente din linia de comandă, argv - vectorul argumentelor de la linia de comandă și env - variabilele de mediu.

**int i = 0;**

Această linie declară o variabilă de tip întreg numită i și o inițializează cu valoarea 0.

**printf("Environment variables:\n");**

Această instrucțiune afișează un mesaj de început pentru a indica că lista de variabile de mediu urmează să fie afișată. Caracterul \n este folosit pentru a introduce un caracter de linie nouă și a muta cursorul pe o nouă linie.

**while (env[i])**

**{**

**printf("env[%d]: %s\n", i, env[i]);**

**i++;**

**}**

Această instrucțiune folosește un ciclu while pentru a parcurge toate variabilele de mediu disponibile, prin intermediul vectorului env. Variabila i este utilizată ca index în vector. Ciclul este executat până când se ajunge la o variabilă de mediu care are valoarea NULL, ceea ce indică sfârșitul listei. Fiecare variabilă de mediu este afișată împreună cu poziția sa în vector, utilizând șiruri de formatare %d pentru a afișa indexul și %s pentru a afișa valoarea variabilei.

**return 0;**

Această instrucțiune indică că programul s-a încheiat cu succes și se returnează valoarea 0.

Pentru a arata functionaliteatea acestui program se ruleaza programul si apoi se excuta comanda **printenv**

**Utilizare array-uri unidimensionale**

**int main(int argc, char \*argv[])**

**{**

Această linie definește funcția principală a programului, main(), care primește doi argumente: argc - numărul de argumente din linia de comandă și argv - vectorul argumentelor de la linia de comandă.

**long t[10], \*p;**

Această instrucțiune declară un vector de 10 elemente de tip long numit t și un pointer la long numit p. Pointerul p este inițial neinițializat.

**int i;**

**for (i=0; i<10; t[i++]=i);**

Această instrucțiune initializează fiecare element al vectorului t cu valoarea indexului său utilizând un ciclu for. Variabila i este inițializată cu valoarea 0, iar ciclul se execută până când valoarea acesteia devine egală cu 10. Operatorul post-incrementare i++ este utilizat pentru a crește valoarea lui i cu 1 după fiecare iterație. De fiecare dată, valoarea indexului este atribuită elementului corespunzător din vectorul t.

**p = t;**

Această instrucțiune atribuie adresa primului element al vectorului t pointerului p.

**for (i=0; i<10; i++)**

**printf("%d %d %d %d\n", t[i], p[i], \*(p+1), \*(t+1));**

Această instrucțiune afișează valorile elementelor vectorului t, valorile corespunzătoare ale vectorului p, precum și valorile care sunt accesate prin pointerii p+1 și t+1. Fiecare element este afișat în format zecimal cu ajutorul șirului de formatare %d.

La fiecare iterație a ciclului for, se afișează valoarea elementului corespunzător din vectorul t și valoarea corespunzătoare din vectorul p prin intermediul expresiei p[i]. În plus, expresiile \*(p+1) și \*(t+1) accesează elementul de la o poziție dincolo de primul element al vectorului, astfel încât se afișează aceleași valori pentru toate cele 10 iterații ale ciclului.

**Important – un vector este reprezentat printr-un pointer la primul sau element. Atunci când declarați un vector, compilatorul aloca automat o zonă de memorie consecutivă pentru elementele vectorului, iar variabila asociată primei poziții în memorie este considerată implicit un pointer către primul element al vectorului.**

**return 0;**

Această instrucțiune indică că programul s-a încheiat cu succes și se returnează valoarea 0.

Lab2\_5.c

Codul are o eroare de sintaxă. În funcția scanf(), trebuie să pasați adresa variabilei n, nu valoarea sa. Codul corectat arată astfel:

#include <stdio.h>

int main(int argc, char \*argv[])

{

int n;

printf("Introduceti valoarea lui n: ");

scanf("%d", &n); // corectat pentru a trece adresa lui n

printf("Valoarea lui n este: %d\n", n);

return 0;

}

Dacă adresa variabilei n nu este trecută corect în funcția scanf(), se va produce o eroare de acces la memorie și programul va închide în mod neașteptat.

**Sa cititm continutul unui fisier**

**char c;**

**char buffer[50];**

**char filename[] = "test.txt";**

Aceste instrucțiuni definesc variabilele necesare pentru citirea fișierului. Variabila c va fi utilizată pentru citirea caracterelor din fișier, variabila buffer va fi utilizată pentru citirea fișierului folosind un buffer, iar variabila filename conține numele fișierului pe care îl vom citi.

**FILE \*file = fopen(filename, "r");**

**if (file == NULL)**

**{**

**perror("Error: ");**

**return 1;**

**}**

Această instrucțiune deschide fișierul cu numele specificat în modul "read" și returnează un pointer la structura FILE. Dacă fișierul nu poate fi deschis, se afișează un mesaj de eroare utilizând funcția perror() și se încheie programul cu valoarea 1.

**printf("Trying to read file char by char...\n");**

**while ((c = fgetc(file)) != EOF)**

**{**

**printf("%c", c);**

**}**

Această instrucțiune afișează un mesaj pentru a indica că se încearcă citirea fișierului caracter cu caracter, apoi se utilizează un ciclu while pentru a citi fiecare caracter din fișier utilizând funcția fgetc(). Acest ciclu se execută până la întâlnirea caracterului de final de fișier (EOF).

**rewind(file);**

Această instrucțiune reinițializează pointerul către începutul fișierului utilizând funcția rewind(). Această instrucțiune este opțională, dar necesară dacă doriți să citiți fișierul din nou.

**printf("Trying to read file using a buffer...\n");**

**while (fgets(buffer, sizeof(buffer), file) != NULL)**

**{**

**printf("%s", buffer);**

**}**

Această instrucțiune afișează un mesaj pentru a indica că se încearcă citirea fișierului utilizând un buffer, apoi se utilizează un ciclu while pentru a citi conținutul fișierului în blocuri utilizând funcția fgets(). Această funcție citeste cel mult sizeof(buffer) caractere și pune caracterul NULL la sfârșitul șirului citit. Ciclul se execută până la întâlnirea caracterului de final de fișier (EOF).

**fclose(file);**

Această instrucțiune închide fișierul utilizând funcția fclose(). Aceasta este o practică bună pentru a elibera resursele și pentru a evita coruperea fișierului. La final, funcția main() returnează valoarea 0, indicând că programul s-a încheiat cu succes.

**Lab2\_7**

Codul de mai jos are o mică eroare, dar funcționează în mod corect. Voi explica fiecare linie de cod în continuare:

#include <ctype.h>

#include <stdio.h>

#include <stdlib.h>

Aceste instrucțiuni includ trei fișiere de antet: ctype.h, stdio.h și stdlib.h.

void allocate\_array(int rows, int cols, int \*\*array)

{

array = (int \*\*)malloc(rows \* sizeof(int \*));

if (array == NULL)

{

perror("Error");

exit(EXIT\_FAILURE);

}

for (int i = 0; i < rows; i++)

{

array[i] = (int \*)malloc(cols \* sizeof(int));

if (array[i] == NULL)

{

perror("Error");

exit(EXIT\_FAILURE);

}

}

}

Această funcție alocă o matrice cu rows rânduri și cols coloane și returnează pointerul la matrice. Se alocă mai întâi un array bidimensional de pointeri cu rows elemente și apoi se alocă câte cols elemente pentru fiecare element din acest array bidimensional. Funcția este comentată, dar nu este utilizată în programul actual.

void print\_array(int rows, int cols, int \*\*array)

{

for (int i = 0; i < rows; i++)

{

printf ("( ");

for (int j = 0; j < cols; j++)

{

printf("%d ", array[i][j]);

}

printf(")\n");

}

}

Această funcție afișează matricea formată din rows rânduri și cols coloane utilizând o buclă for. Este afișată o paranteză deschisă, fiecare element din matrice este afișat, iar la sfârșit este afișată o paranteză închisă urmată de o nouă linie.

Acest cod citește din fișierul "matrix.txt" o matrice de dimensiuni predefinite (nrows=3 și ncols=4), fiecare element al matricei fiind de tip integer. După citire, matricea este afișată în consolă și memoria alocată este eliberată.