Hardver programozás

C programozás nyelv:

Programozás: egymás alá írt utasítások egy szöveges fájlban (formázás nélkül!),

melyet a fordító program átalakít (átkonvertál) bináris számok sorozatára.

Algoritmus: gépiesen (egyéni döntések, gondolatok) nélkül végrehajtott

utasítások sorozata

A program elemei:

elemi tevékenységek: a=5; a+=1 műveletek a változóval

szekvencia: utasítások sorozata

döntés/elágazás

ciklus (többször hajtom végre az utasítást, elemi műveletet)

szubrutin hívás (függvények)

Adat: ami a programozó számára információt jelent.

Az adatnak két alapvető tulajdonsága van:

* értéke (mekkora)
* típusa (memóriában elfoglalt méretét, ábrázolását határozza meg)

A hétköznapi életben csak az értékével foglalkozunk.

A memória/tárolás, továbbá milyen műveletek megengedettek vele

szempontjából fontos az adat típusa.

Változó: az adat tárolására szolgáló memória terület, melyre azonosítójával (egy értelmes, beszédes név) hivatkozunk.

A változó mindig meghatározott adat tárolására szolgál, ezért a változó létrehozásakor meghatározzuk (deklaráljuk) milyen típusú adatokat fogunk benne tárolni.

Típusok az adott programozási nyelv által kínált típus lehet.

Pl.: double nincs Pythonban, C-ben van

struct, tömb nincs Pythonban, C-ben nincs lista, szótár

C-ben használt típusok:

alaptípusok:

char (karakter, 1 byte)

int (egész, 2 vagy 4 byte)

float(lebegőpontos, 4 byte)

double(lebegőpontos 8 byte)

enum(felsorolás típus)

struct(összetett adatszerkezet)

union

void („üres”), ha nem adok értéket

Az alap típusok által meghatározott memória egységeket a típusmódosítókkal módosíthatjuk:

short, long: tárolási hosszúság

signed, unsigned (előjeles, előjel nélküli). pl.: int esetén max. 2^16 pozitív számig tudunk elmenni

unsigned int esetén max. 2^32 pozitív számig.

Változó deklarálása:

típus változónév;

Érték is adható deklaráláskor:

típus változónév=érték;

pl: int a=5;

C nyelvben a printf függvényt használjuk a kiíratásra, ahol meg kell adni a kiíratandó változó esetén a típus formátumát. Ha csak értéket írok ki akkor nem kell típus megadás, ”” között írom ki.

int %d

char %c

float %f

double %lf

sztring %s

egész 16 számrendszerben %x

printf(”Szép napot!”);

Legegyszerűbb C program: nem csinál semmit

void main() //program végrehajtás kezdete, belépési pontja. Pythonban ilyen nem volt

{

} programkód vége

Egysoros megjegyzés: //

Többsoros megjegyzés:/\*\*/

# szolgál arra, hogy a fordítónak közvetlenül utasításokat adjak. Ez nem a program logikájához kell, hanem a „működés”.

include: egy filet emelek be a programba, olyan, mintha odamásoltam volna az elejére.

A file-névre (amit be akarok includolni) a < > között hivatkozok rá.

.h kiterjesztésű az a file ami deklarációkat tartalmaz.

Nem egy végrehajtható kód, nem tartalmaz main függvényt.

#include <stdio.h>

void main() //program végrehajtás kezdete, belépési pontja. Pythonban ilyen nem volt

{

printf("Szép napot!"); Az utasításokat ;-el zárom le, vagy {} ha több utasítás egybe tartozik

} programkód vége

---------------------------------------------------

#include<stdio.h>

void main()

{

int a=11;

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

}

A standard kimenetre írás a printf() utasítással történik, mely formázott megjelenítést tesz lehetővé. A beolvasásra több lehetőség is van. Formázott bevitel az scanf() utasítással történik.

Szintaktikája: scanf(”%formátum karakter, a típusnak megfelelő”,&változó neve), ahol az & a címét jelenti, tehát az adott változó címére olvassa be.

**Elágazó utasítás**: a feltétel teljesülése vagy nem teljesülése esetén történik a program utasításainak végrehajtása. A program végrehajtása melyik ágon folytatódik tovább.

Szintaktikája:

if(feltétel)

{ utasítások;}

else

{ utasítások;}

Folyamatábra: Feltétel

hamis (nem teljesül)

igaz

Többszörös feltétel vizsgálatra szolgál a **switch utasítás**. If-el is megoldható, de így olvashatóbb a kód.

Szintaktikája:

switch(feltétel)

{

case érték1:

utasítás;

break;

case érték2:

utasítás;

break;

case érték2:

utasítás;

break;

default:

utasítás;

}

A feltétel kiértékelésekor kapott eredmény amelyik case ág értékével egyezik meg, azt az utasítást vagy utasításokat hajtja végre.

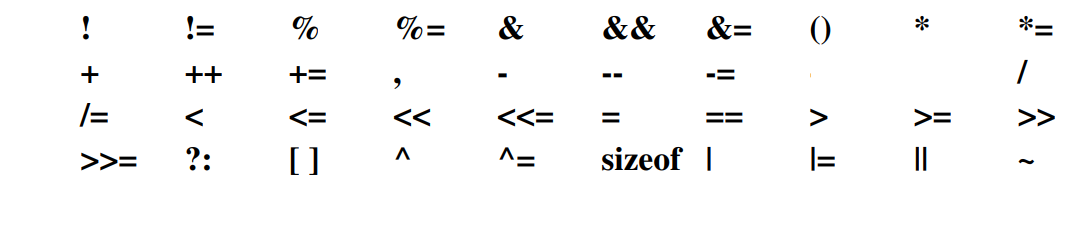
Ha egyik értékkel sem egyezik, akkor a default utáni utasításokat hajtja végre.

Ha nem szerepel break a case ágban, akkor a következő case ágat vagy ágak utasításait is végrehajtja.

Nem kötelező a default, el is hagyható.

szamologep\_switch.c

**Operátorok:**



Relációs operátorok:

<,<=,>>=,==,!=

Logikai operátorok:

! not

&& and

|| or

Értékadó operátor:

= , +=,-=,/=,%=

Példa: ezek a kifejezést rövidítik le.

a=a+2 a+=2

a=a-2 a-=2

a=a/2 a/=2

Matematikai operátor:

/ osztás

% maradékos osztás

\*

+

-

Bitműveletek operátorai:

&,<<,>>,<⬄>=,^,^=,|,|=,~

Léptető operátor: ++,--

i++;

i--;

++i;

--i;

kifejezés amit vizsgálok?ha igaz:, akkor azt hajtja végre ami itt áll:nem igaz esetén ezt hajtja végre

eredmeny=a<0?-a:a;

abszolutertek.c

abszolutertek2.c példa ?: operátorra

Feladat

kisebbnagyobb.c Bekér a felhasználótól két számot és meghatározza, hogy melyik a nagyobb.

negativpozitiv.c Bekér a felhasználótól egy számot és meghatározza, hogy pozitív vagy negatív.

parosptlan.c Bekér a felhasználótól egy számot és meghatározza, hogy páros vagy páratlan.

**Ciklusok, ciklus utasítások: ismételten végrehajtandó műveletekre.**

for(ciklus változó kezdeti érték beállítása; **végérték, kilépési feltétel**; léptetés)

{utasítások}

Akkor használom, amikor ismerem a ciklus futásának lépés számát.

ciklus változó definiálása és kezdeti értékének beállítása

while(kifejezes)

{utasítások; **ciklusváltozó növelése/csökkentése**;}

Csak akkor lép bele a ciklusba, ha a feltétel igaz, teljesül.

do

{ utasítások

**ciklusváltozó növelése/csökkentése;**

}while(feltétel);

Egyszer mindenképpen lefut.

Mind a while, mind a do-while futása a feltétel hamissá válásáig fut, azaz nem feltétlenül tudjuk,

hogy ez hány lépésszámból történik meg.

Végtelen ciklusok:

for(;;;) {}

while(1) {}

do{

}while(1);

**Tömbök:**

Összetett adatszerkezet. Egy változóban több elemet tárolok el.

Az összes elem azonos típusú.

**Deklarációja, definiálása:**

**típus tömb\_neve[méret];** Ilyenkor statikusan lefoglaljuk a tömb számára a méretet.

A típussal határozzuk meg, milyen elemeket tárolunk a tömbben.

pl.: int ertekek[30];

Statikus tömbfoglalás: //statikus szó azt jelenti, hogy állandó. Tehát nem változik a tömb lefoglalt mérete a futás során.

Mindig nagyobb méretet kell lefoglalnom, mint amennyi adatot töltök bele

tipus tömb\_neve[méret]; // Ez statikusan lefoglalt egydimenziós tömb pl.:1,2,3,4,56

int ert[10][5];//Ez egy statikusan lefoglalt tömb, ami két dimenziós.ért[10][5]

10 sor,5 oszlop.

//oszlopok száma

//pl.: 6,4,6,5,7

3,7,8,9,0

2,5,5,0,4

6,6,6,7,7

9,8,7,6,2

Kezdeti értékeket adjunk meg a tömbnek, ebben az esetben nem szükséges meghatározni

a tömb méretét, mert a fordító automatikusan meghatározza a kezdeti értékek számából

a méreteket (dimenziókat).

float tt[]={3.14,2.72,9.81}

Ha pontosan tudni szeretnénk mennyi elem van a tömbben a sizeof() operátorral kérdezhetjük le.

sizeof() operátor, ami meghatározza a tömb méretét

meret=sizeof(tömb változó neve); Ez a teljes tömb mérete.

Azt szeretném meghatározni, hogy hány eleme van, akkor

darab=sizeof(tt)/sizeof(tt[0]);

példa: 4byte\*3/4byte=3

#define N 34 //Az előfeldolgozónak szól, ami nem csinál mást, mint mindenhol ahol

//megtalálja az N-t, oda 34-et helyettesít be.

Bejárása ciklussal történik.

A tömb elemeire indexekkel hivatkozok.

Az indexelés 0-tól indul.

Meg kell adnom a tömb méretét az inicializáláskor. Ha nem adom meg, fordítási hibát kapok.

Kivéve kezdeti értékadás a deklarációkor, mert akkor annyi területet foglal le a fordító, amennyi a kezdeti értékek száma.

int ertekek[]={2,45,67,3,2,1,5,26};