Oinarrizko Programazioa I

Gaien Aurkibidea

Erreferentziak

Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language. 2nd Edition.* Prentice Hall, 2nd edition, April 1998.

Inaki Goirizelaia. *Programazioaren Oinarriak.* EUSKAL HERRIKO UNIVERTSITATEA, 1999.

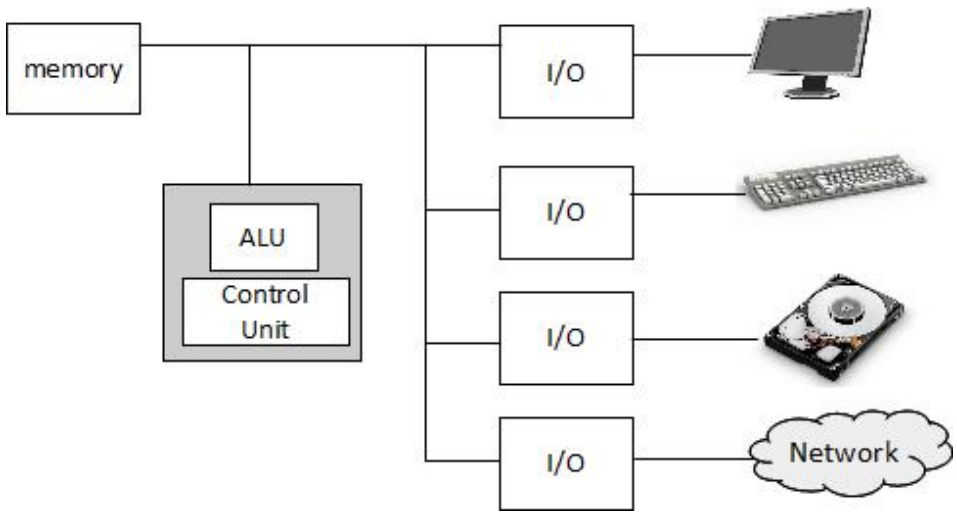
Michael Barr. *Embedded C Coding Standard.* Createspace Independent Pub, 2009.

1 Sarrera

1.1 Helburua

- **Helburua:** programazioaren oinarrizko elementuak ezagutu eta hauek erabiliz programatzea. Hau da, algoritmoak gauzatzen ikastea.
- Horretarako zenbait tresna beharko ditugu baina ez dira ikasgaiaren helburu; tresnak baino.
  - Programazio lengoai bat: gure kasuan, C.
  - Programak garatzeko tresnak: gure kasuan Visual C++.
- Beraz, algoritmoak helburutzat hartuta, C lengoaian programatzea ere lortuko da.
  - C lengoaia eta Visual C++ tresnari dagokienez, beharrezko elementuak ikusiko ditugu soilik.
  - Hauei dagozkien elementuak, algoritmoen garapenak eskatzen duten heinean ikusiko ditugu.

1.2 Konputagailuen Arkitektura

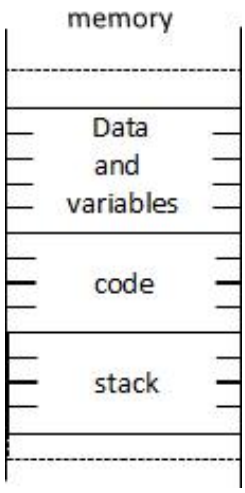


Irudia 1: Konputagailu baten oinarrizko arkitektura

1.3 Programa

- **Zer da Programa bat?** Aldagai multzo bat gehi instrukzio sekuentzia bat.
- Konputagailu arrunt batetan:
  1. Programa bat diskoko fitxategi batetan egoten da (.exe atzizkia duen batetan adibidez).
  2. Programa hori exekutatzeko, sistema eragileak memoriara ekarriko du. Zona batetan aldagaiak egongo dira, beste batetan instrukzioak (2 irudia).
  3. Programa exekutatzeak zera esan bahi du: kontrol unitatea aipatutako instrukzio zerrendak diona egitea.
  4. Instrukzio hauek funtsean zera egin dezakete:
     (a) Aldagaiek gordeta daukaten datua UAL-ra ekarri.
     (b) UAL-n kalkulu batzuk egin:

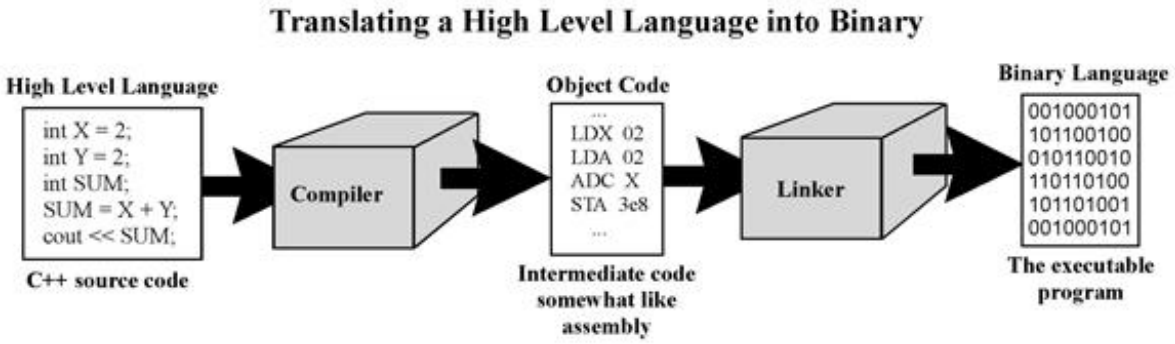
(c) UAL-tik aldagai batetara datu bat eraman.
(d) Gailu baten kontrolatzailetik datu bat UAL-era ekarri
(e) UAL-etik gailu kontrolatzaile batetara datu bat eraman.
(f) .....



Irudia 2: Programa bat memoriara kargatu ondoren.

1.4 Goi Mailako Lengoaiak

- Prozesadore batek uler dezakeen instrukzio zerrendari (eta formatua noski) ***makina lengoaia*** deritzo.
- Programazioa errazteko helburuarekin, goi mailako lengoaiak daude. Hauen bitartez idatziko ditugu programak eta ondoren makina kodera transformatu.



Irudia 3: Goi mailako lengoaiatik makina lengoaiarako transformazioa

2 C Lengoaian Programatzeko Oinarrizko Sententzia eta ...

- **Sententzia**-gatik, prozesadoreak exekutatu behar duen agindu banaezin bat ulertuko dugu.
- C lengoaian sententzia guztiak ';' karaktereak amaitu behar ditu,
- Sententzia konposatuak eratu daitezke. Horretarako '{' '}' karaktere artean zerrendatuko ditugu sententzia konposatu hori osatzen duen sententzi zerrenda.

2.1 Sarrera/Irteera

- Programa gehienek erabiltzailearengandik informazioa lortu behar dute edo honi informazioa bidali.
- Eginkizun honetarako era asko daude baina guk jarraituko duguna 1. listatukoa da.

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(int argc, char* argv[])
{
  int i;
  float f;
  char str[128];

  printf("Emaidazu zenbaki oso bat: ");
  fgets(str, 128, stdin);
  sscanf(str, "%d", &i);
  printf("Eta orain zenbaki erreal bat: ");
  fgets(str, 128, stdin);
  sscanf(str, "%f", &f);
  printf("%d eta %f zenbakiak eman dizkidazu\n", i, f);
  printf("Sakatu \"return\" amaitzeko .....");
  fgets(str, 128, stdin);
  return 0;
}
```

Listing 1: 01sarreraIrteeraMain.c fitxategia

2.2 Adierazpenak eta Esleipena

- Adierazpen batek kalkulu bat espezifikatzen du. Bai eta (gehienetan) datuak nondik hartu eta emaitza non gorde.

```
int main(int argc, char* argv[])
{
  int zatikizun, zatitzaile, ondar, zatidura;

  .....
  zatikizun=24;
  zatitzaile=3;
  ondar=zatikizun % zatitzaile;
  zatidura=zatikizun/zatitzaile;
  .........
}
```

Listing 2: esleipen eta adierazpen adibideak

- 1 taulan agertzen dira adierazpen batetan ager daitezkeen C lengoaiaren eragile guztiak.

operator	precedence	associativity
$() \; [] \; . \; \rightarrow$	higher	left to right
$! \quad - \; ++ \; -- \; \& \; * \; (tipo) \; sizeof$	|	right to left
$* \; / \; \%$	|	left to right
$+ \; -$	|	left to right
$<< \; >>$	|	left to right
$== \; !=$	|	left to right
$\&$	|	left to right
$\wedge$	|	left to right
$\vert$	|	left to right
$\&\&$	|	left to right
$\vert\vert$	|	right to left
$? :$	|	right to left
$= \; += \; -= \; *= \; etc-$	lower	left to right

Taula 1: C lengoaiaren eragileen zerrenda

- **Aldagaia**: balio bat gorde dezakeen memoriako gune bat. Aldagaiak mota batetakoa izan behar du; hau da mota batetako balioak gordetzeko balio du soilik.

mota	deskripzioa
`void`	ezer, ...
`char`	8 bit-etako zenbaki osoa (8 bits), karakterea
`int`	zenbaki osoa
`float`	zenbaki erreala
`double`	prezisio gehiagoko zenbaki erreala

Taula 2: Oinarrizko datu motak.

- `char` e `int` motek 3. taulako modifikatzaileak onartzen dituzte:

modifikatzaile	deskripzioa
`signed`	zenbaki osoa. defektuzko aukera
`unsigned`	zenbaki arrunta

Taula 3: Mota integralen modifikatzaileak

- **Esleipena**: `'='` eragilearen bitartez aldagai batetan balio konkretu bat gordetzeko agindu dezakegu.

```
int n,m;
float f1, f2, f3;

n=23;
m=4
f1=23;
f2=8.5;
n=n/m+8+n%3*4;  //n aldagaian 21 gordeko da
f3=f1/f2;       //f3 aldagaian 2,7058... gordeko da
n=f1/f2;        //n aldagaian 2 gordeko da
f3=23/m;        //f3 aldagaian 5 gordeko da
f3=23/(float)m; //f3 aldagaian 5,75 gordeko da
```

Listing 3: esleipen eta adierazpen adibideak

  - *Kalkuluak egiten direnean, aldagai edo konstanteen motaren arabera egiten dira.* Adibidez, `n/m` adierazpenean, `n` eta `m` `int` badira, zenbaki osoei dagokien zatiketa egingo da eta `float` balira, zenbaki errealei dagokiena.
  - Edozein kalkulutan, eragigai guztiak `int` bezala tratatzen dira gutxienez; eta behar izanez gero `float` edo `double` bilaka daitezke.

- C lengoaiak, idazketa errazteagatik eta kode efizientziarrekiko laguntza gisa, zenbait eragile eta adierazteko ahalmen aukera ditu:

```
int n,m;
float f1, f2, f3;

n=0;
n=n+2;       // n  <--  2
n+=2;        // n  <--  4
n++;         // n  <--  5
++n;         // n  <--  6
m=3+(n++);   // m  <--  9  eta n <-- 7
m=3+(++n);   // m  <--  11  eta n <-- 8
n=m=m+7;     // m  <--  18  eta n <-- 18
```

Listing 4: zenbait eragile *"labur"* eta adierazpen gaitasun

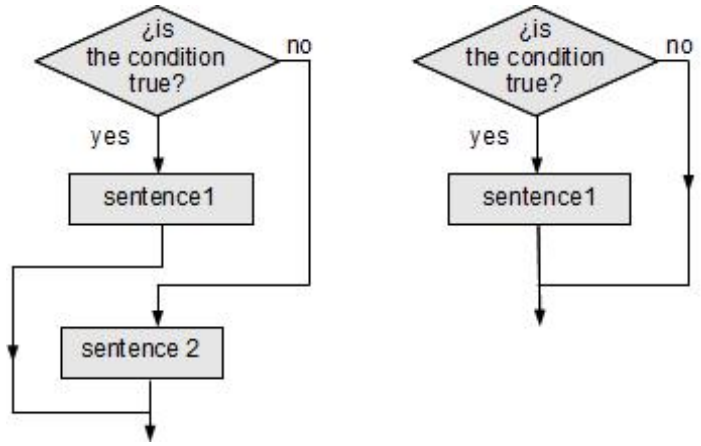
- **Adierazpen boolearrak**: `{0,1}`, `{false, true}`, `{egia,gezurra}`
  - C lengoaian ez dago boolear motarik.
  - Balio boolear bezala interpretatu behar den adierazpenetan edozein datu mota erabil dezakegu: hau 0 bada `false` moduan hartuko da eta $0 \neq$ bada, `true` moduan.

- C lengoaian *"ulertezin"* bilaka daitezkeen adierazpen konplexuak idatz daitezke. Argitasuna gomendatzen da *"idaketa ekonomiaren"* aurretik.

2.3 IF sententzia

- Sintaxia eta eginkizuna:

```
if(condition)
  sentence1;
[else
  sentence2; ]
```



- Adibidea:

```
/*-------------------------------------------------------------------
Erabiltzeari bi zenbaki eskatu eta txikitik haundira ordenatuta
pantailaratuko dizkion programa
-------------------------------------------------------------------*/
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(int argc, char* argv[])
{
  int x, y, tmp;
  char str[128];

  printf("Emaizkidazu 2 zenbaki oso: ");
  fgets(str,128,stdin);
  sscanf(str, "%d %d", &x, &y);
  if (x >= y)
  {
    tmp = x;
    x = y;
    y = tmp;
  }
  printf("%d <= %d \n", x, y);
  printf("Sakatu \"return\" amaitzeko .....");
  fgets(str, 128, stdin);
  return 0;
}
```

Listing 5: 02ifMain.c fitxategia

- Zenbait eragile boolear, konparaziorako eragile eta baldintza adibide:

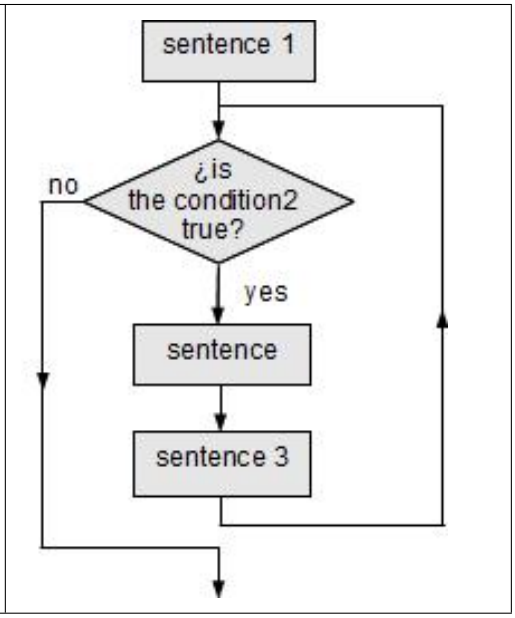
```
if((n>3) && (n<=8)) .....
if((n!=2) && (n % m ==7)).....
if((n > 0) || !(n>8)) .....
```

Listing 6: adierazpen-boolear adibideak

2.4 FOR sententzia

- Sintaxia eta eginkizuna:

```
for(sentence1;condition2;sentence3)
  sentence;
```



- Adibideak:

```
/*-------------------------------------------------------------------
Erabiltzeari zenbaki arrunt bat eskatu horrenbeste *-tako ilara
marraztuko dion programa
-------------------------------------------------------------------*/
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(int argc, char* argv[])
{
  int n, i;
  char str[128];

  printf("Emaidazu ilararen luzera: ");
  fgets(str, 128, stdin);
  sscanf(str, "%d", &n);
  for (i = 0; i < n; i++)
  {
    printf("*");
  }
  printf("\n");
  printf("Sakatu \"return\" amaitzeko.....");
  fgets(str, 128, stdin);
  return 0;
}
```

Listing 7: 05forMain.c fitxategia

```
/*-------------------------------------------------------------------
     Erabiltzeari n zenbaki arrunt bat eskatu [1..n] eta tarteko
   zenbaki gutien batura kalkulatuko dion programa
-------------------------------------------------------------------  */
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(int argc, char* argv[])
{
  int n, i, batura;
  char str[128];

  printf("Zenbaterainoko zenbakiak batu nahi dituzu? ");
  gets_s(str, 128);
  sscanf_s(str, "%d", &n);
  batura = 0;
  for (i = 1; i <= n; i++)
  {
    batura = batura + i;
  }
  printf("%d-rainoko zenbakien batura %d da.\n", n, batura);
  printf("Sakatu \"return\" amaitzeko .....");
  gets_s(str, 128);
}
```

Listing 8: 07forMain.c fitxategia

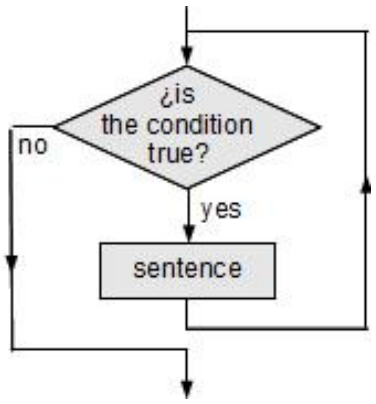
- Bere erabilerari buruzko oharra. For sententziaren sintaxiak aukera ugari ematen diten arren,
  - bere erabilera nagusia sententzia bat aldi jakin batez errepikatzea da.
  - horretarako kontagailu lanak egingo dituen aldagai bat erabiliko dugu:

```
for(i=3;i<8;i++) .....
for(i=0;i<n;i+=2) .....
for(i=100;i>=10; i--) .....
```

2.5 WHILE sententzia

- Sintaxia eta eginkizuna:

```
while(condition)
  sentence;
```



- Adibideak:

```c
/*-------------------------------------------------------------------
   Erabiltzeari n zenbaki arrunt bat eskatu zenbat digituz osatuta
dagoen esango dion programa
-------------------------------------------------------------------    */
#include <stdio.h>

int main(int argc, char* argv[])
{
  int zenbatDig, n, gainBorna;
  char str[128];

  printf("Emaidazu zenbaki oso bat: ");
  fgets(str,128,stdin);
  sscanf(str, "%d", &n);
  zenbatDig = 1;
  gainBorna = 10;
  while (gainBorna<=n)
  {
    zenbatDig++;
    gainBorna *= 10;
  }
  printf("%d zenbakia adierazteko %d digitu behar dira\n", n, zenbatDig);
  printf("Sakatu \"return\" amaitzeko .....");
  fgets(str,128,stdin);
  return 0;
}
```

Listing 9: 10whileMain.c fitxategia

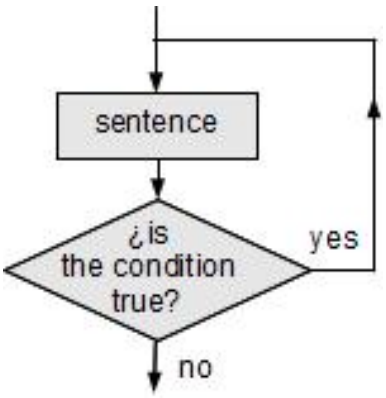
```c
/*-------------------------------------------------------------------
Erabiltzeari n eta m zenbaki arruntak bat eskatu eta n m-gatik
zenbat aldiz zatitu daitekeen esango digun programa.
n=k(m**b) edierazpenetik b kalkulatuko du.
-------------------------------------------------------------------*/

#include <stdio.h>

int main(int argc, char* argv[])
{
  int n, m, kont;
  char str[128];

  printf("Emaizkidazu bi zenbaki arrunt: ");
  fgets(str,128,stdin);
  sscanf(str, "%d %d", &n, &m);
  kont = 0;
  while (n%m == 0)
  {
    n = n / m;
    kont++;
  }
  printf("%d aldiz zatitu daiteke %d-gatik\n", m, kont);
  printf("Sakatu \"return\" amaitzeko .....");
  fgets(str,128,stdin);
  return 0;
}
```

Listing 10: 11whileMain.c fitxategia

2.6 DO-WHILE sententzia

- Sintaxia eta eginkizuna:

```
do
  sentence
while(condition);
```



- Adibideak:

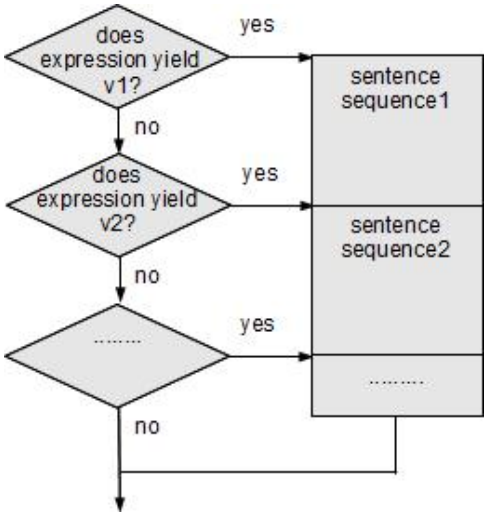
```
/*-------------------------------------------------------------------
Erabiltzeari n zenbaki arrunt bat eskatu zenbat digituz osatuta
dagoen esango dion programa
-------------------------------------------------------------------*/
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(int argc, char* argv[])
{
  int zenbatDig = 0, n, gainBorna = 1;
  char str[128];

  printf("Emaidazu zenbaki oso bat: ");
  fgets(str, 128, stdin);
  sscanf(str, "%d", &n);
  do
  {
    zenbatDig++;
    gainBorna *= 10;
  } while (gainBorna <= n);
  printf("%d zenbakia adierazteko %d digitu behar dira\n", n, zenbatDig);
  printf("Sakatu \"return\" amaitzeko .....");
  fgets(str, 128, stdin);
  return 0;
}
```

Listing 11: 12doWhileMain.c fitxategia

2.7 SWITCH sententzia

- Sintaxia eta eginkizuna:

```
switch(expression)
{
  case value1: <sentence-sequence1>
  case value2: <sentence-sequence2>
  case value3: <sentence-sequence3>
  ......
  [default:     <sentence-sequenceN>]
}
```



3 Funtzioak eta Algoritmoen eta Kodearen Deskonposaketa

3.1 Funtzioen Motibazioa

- Biz n elementu m-naka konbinatzeko dauden aukera kopurua kalkulatzen duen 12 programa. Hau da $C_n^m = \frac{n!}{m!(n-m)!}$.

```
/*-------------------------------------------------------------------
Erabiltzeari n eta m zenbaki arruntak eskatu eta n elementu m-naka
konbinatzeko zenbat aukera ezberdin dauden esango dion programa
-------------------------------------------------------------------*/
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(int argc, char* argv[])
{
  int i, n, m, nFakt, mFakt, nmFakt, emaitz;
  char str[128];

  printf("Zenbat elementu nahi zenituzke konbinatu: ");
  fgets(str, 128, stdin);
  sscanf(str, "%d", &n);
  printf("zenbatnaka: ");
  fgets(str, 128, stdin);
  sscanf(str, "%d", &m);
  nFakt = 1;
  for (i = 2; i <= n; i++) nFakt *= i;
  mFakt = 1;
  for (i = 2; i <= m; i++) mFakt *= i;
  nmFakt = 1;
  for (i = 2; i <= n - m; i++) nmFakt *= i;
  emaitz = nFakt / mFakt / nmFakt;
  printf("%d elementu %d naka konbinatzeko %d aukera daude\n", n, m, emaitz);
  printf("Sakatu \"return\" amaitzeko .....");
  fgets(str, 128, stdin);
  return 0;
}
```

Listing 12: 20combinationMain.c fitxategia

- 12 kodea laburra den arren, zenbait *"mekanismoren"* beharra ikus genezake.

  1. Faktoriala kalkulatzen duen kodigoa hirukoiztuta dago. Ezin al genezake behin bakarrik idatzi?
  2. Faktorialaren kalkulua beste programatzaile bati eskatuko bagenio, nola txertatuko genuke gure programan? kopiatuz eta behar diren aldagaiak gehituz/aldatuz?
  3. Eta kodigo hori aldatzea nahi bagenu, beste algoritmo efizienteago bat edo akats bat aurkitu dugulako adibidez, erabili dugun leku guztietan egin beharko genituzke aldaketak?
  4. Arazo hauetan laguntzeko daude funtzioak. Begiratu 13 kodea.

```c
/*-------------------------------------------------------------------
Erabiltzeari n eta m zenbaki arruntak eskatu eta n elementu m-naka
konbinatzeko zenbat aukera ezberdin dauden esango dion programa
-------------------------------------------------------------------*/
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(int argc, char* argv[])
{
  int  n, m, emaitz;
  char str[128];

  printf("Zenbat elementu nahi zenituzke konbinatu: ");
  fgets(str, 128, stdin);
  sscanf(str, "%d", &n);
  printf("zenbatnaka: ");
  fgets(str, 128, stdin);
  sscanf(str, "%d", &m);
  emaitz = faktorial(n) / faktorial(m) / faktorial(n - m);
  printf("%d elementu %d naka konbinatzeko %d aukera daude\n", n, m, emaitz);
  printf("Sakatu \"return\" amaitzeko .....");
  fgets(str, 128, stdin);
  return 0;
}

int faktorial(int n)
{
  int i, emaitz=1;

  for (i = 2; i <= n; i++) emaitz *= i;
  return emaitz;
}
```

Listing 13: 21combinationMain.c fitxategia

3.2 Aldagai eta Parametroak

- Noiz sortzen/deuseztatzeen dira aldagaiak? Nola erabili daitezke? C lengoaian aukera ugari daude baina ikasgai honetan erabilitako motak mugatu egingo ditugu:
  - **Aldagai globalak**: Inongo funtzioren barruan ez dagoena.
    - Programa hasi aurretik sortzen dira.
    - Programa amaitzerakoan deuseztatu.
    - Edozein tokitatik erabili daitezke.
  - **Aldagai lokalak**: funtzio barruan definitzen direnak.
    - Funtzioari deitu aurretik sortzen dira.
    - Funtzioa amaitzerakoan deuseztatu.
    - Funtzio barruan soilik dira erabilgarri.
- Parametroak ia aldagai lokalak bezelakoak dira. Ezberdintasun bakarra, funtzioa hasten denean deitu dionak dagoeneko balio bat jarri duela da.

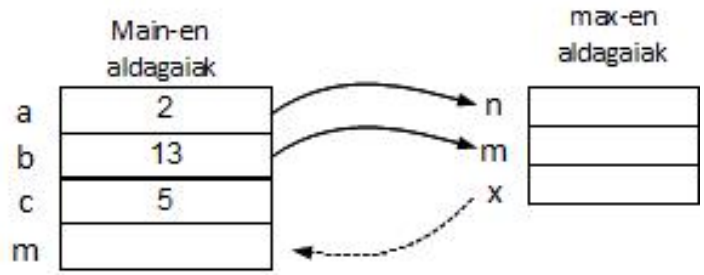
- Erabil dezagun adibide bat kontzeptu hauek argitzeko.

```
int main(int argc, char* argv[])
{
  int a,b,c,m;

  .....
  //a,b,c aldagaien maximoa m-n utzi
  m=max(a,b);
  m=max(m,c);
  .........
}
```

```
int max(int n, int m)
{
  int x;

  if(n>m) x=n;
  else x=m;
  return x;
}
```

Listing 14: main max-ri deitzen

1. `main` funtzioa hasterakoan bere `argc`, `argv` `a`, `b`, `c` eta `m` parametro eta aldagaiak sortuko dira.
2. ...
3. `max` funtzioari lehenengo aldiz deitzerakoan
   (a) Bere `n`, `m` eta `x` parametero eta aldagaiak sortuko dira
   (b) `a` eta `b`-ren edukinak `n` eta `m`-ra kopiatuko dira hurrenez hurren
4. `max` funtzioa amaitzerakoan bere parametro eta aldagaiak desagertu egingo dira. Itzultzen duen balio geratuko da *"nonbaiten"* (konpiladoreak sortzen duen aldagai tenporal batetan).
5. Itzuli duen balioa `m`-n gordeko da.
6. `max` funtzioari bigarren aldiz deitzerakoan
   (a) Bere `n`, `m` eta `x` parametero eta aldagaiak sortuko dira
   (b) `m` eta `c`-ren edukinak `n` eta `m`-ra kopiatuko dira hurrenez hurren.
7. `max` funtzioa amaitzerakoan bere parametro eta aldagaiak desagertu egingo dira eta ...
8. Itzuli duen balioa `m`-n gordeko da.
9. ...
10. `main` funtzioa amaitzerakoan bere `argc`, `argv` `a`, `b`, `c` eta `m` parametro eta aldagaiak deusezatu egingo dira.

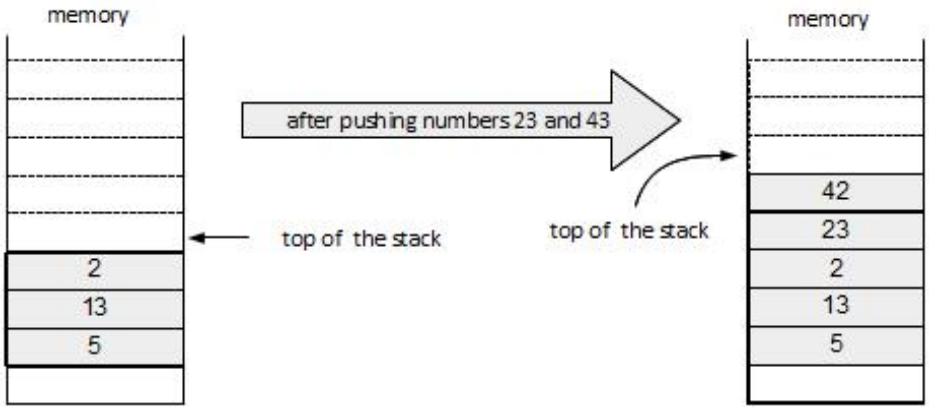


Irudia 4: 14 kodigoko aldagai lokal eta parametroak

3.3 Aldagai eta Parametroen Metatzea

- Funtzio bati deitu aurretik bere aldagai eta parametroak sortu egiten direla ikusi dugu aldez aurretik.
- Baina, aldagai bat sortzeak memoriako zati bat erreserbatzea esan nahi du. Nun eta nola egiten da erreserba hori?

- Exekuzioan dagon programa orok ***meta*** bat du batez ere aldagaiak pilatzeko edo ***metatzeko***. Begiratu 5 irudia.

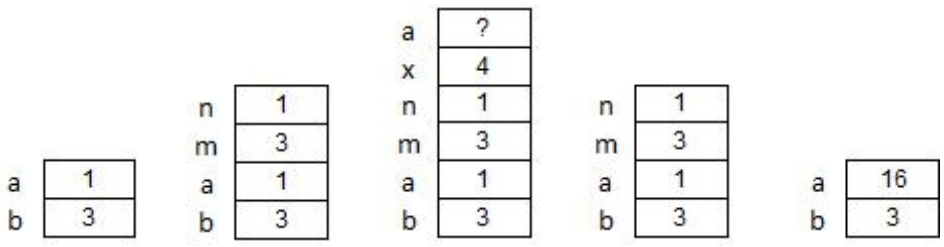


Irudia 5: Meta (Pila, Stack): Gainean elementu bat jarri edo gaineko kendu baino ezin da egin

- Har dezagun adibide gisa 15 kodea. `main` funtzioaren lehendabiziko sententziatik hasten bagara, 6 irudian agertzen da funtzio bakoitzari deitu aurreko eta ondorengo metaren egoera.

```
int main(int argc, char* argv[])
{
  int a,b;

  a=f(a,b);
}

int f(int n, int m)
{
  return g(n+m);
}

int g(int x)
{
  int a;

  a=x*x
  return a;
}
```

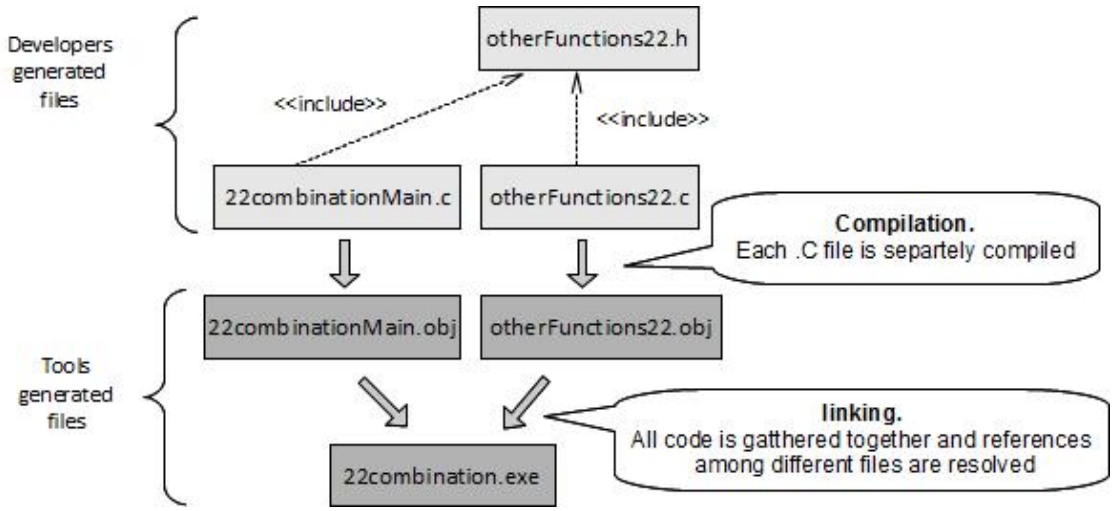
Listing 15: Metaren egoera azaltzeko adibidea



Irudia 6: 15 kodigoko metaren egoera funtzio bakoitzari egindako deiaren aurretik eta ondoren

3.4 Programen Garapenean Sor Daitezkeen Fitxategien Aukerak

- Programa baten tamaina handitzen doan heinean, fitxategi bakarrean idatzi ordez kode iturria fitxategi ugaritan banatu ahal izatea lagungarri litzateke:
  - Fitxategi bakoitzean irizpide baten arabera sailkatutako elementuak sar ditzakegu soilik. Adibidez funtzio matematikoak, funtzio grafikoak, datu baseak erabiltzeko funtzioak, ...
  - Garapen taldea pertsona ugariz osatuta badago, bakoitzaren lana fitxategi ezberdinetan legoke.
  - ...
- Azal dezagun adibide erraz baten bidez. 13. listatuko kodea fitxategi bakarrean dago idatzita. 7. irudiak erakusten du geuk idatziko ditugun fitxategiak, bai eta tresna ezberdinek sortuko dituztenak.



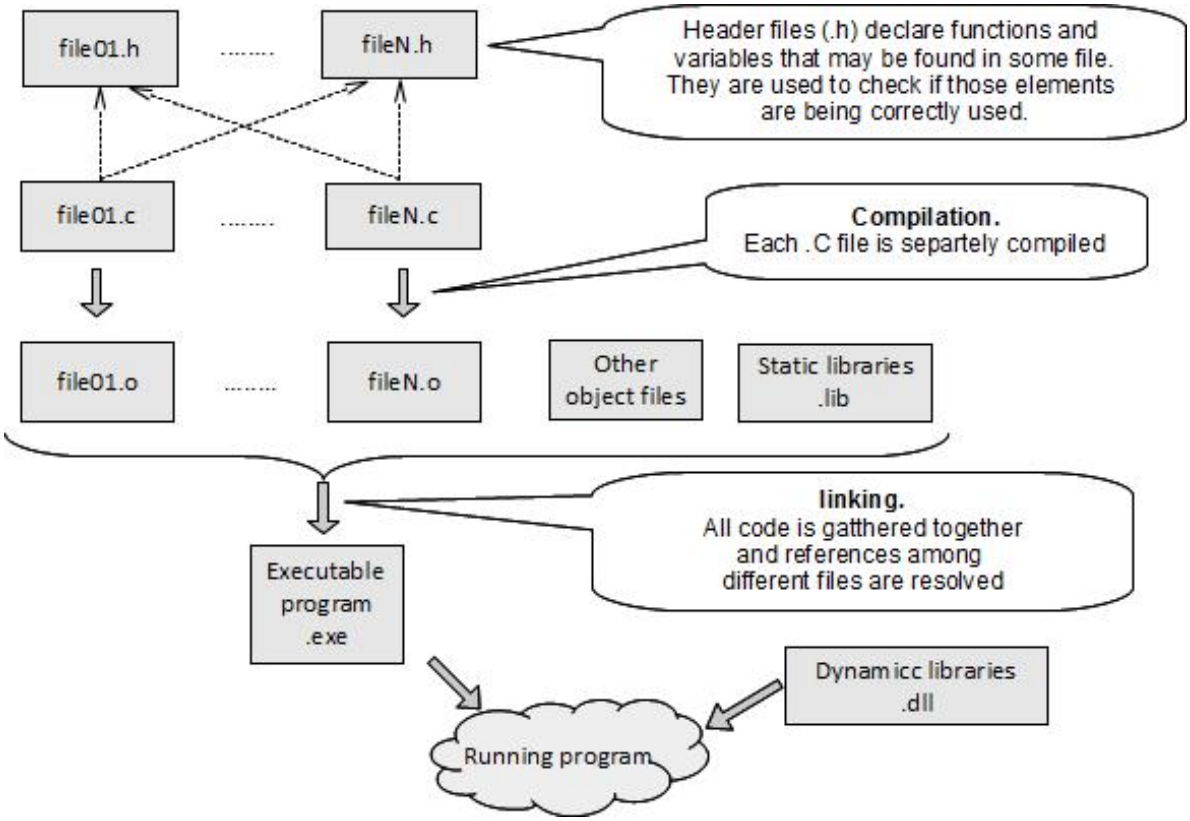
Irudia 7: Garapenean eta exekuzioan ager daitezkeen fitxategi mota ezberdinak

1. `otherFunctions.c` fitxategian `faktorial` funtzioa idatziko dugu.
2. `22combinationMain.c` fitxategian `main` funtzioa.
3. `otherFunctions.h` fitxategia ere sortuko dugu. `#include` direktibaren bitartez aurreko bietan sartuko dugu. Honen helburua bikoitza da:
   (a) funtzio honen deklarazioa eta definizioa bat datozela zirtatzea.
   (b) ongi erabiltzen dela ziurtatzea.

```
#ifndef OTHERFUNCTIONS22_H
#define OTHERFUNCTIONS22_H

int faktorial(int n);

#endif
```

Listing 16: otherFunctions22.h fitxategia

- 8 irudian agertzen da orokorrean C lengoaiaren inguruko tresnek eskaintzen diguten aukera.



Irudia 8: Garapenean eta exekuzioan ager daitezkeen fitxategi mota ezberdinak

4 Array-ak

- Demagun programa zati batetan 100 balio ezberdin erabili behar direla. 100 aldagai desberdin definitu behar al ditugu?
  - $10x10$ dimentsiotako matrizea badugu, 100 aldagai beharko genituzke?
  - 200 elementutako bektore edo zenbaki zerrenda bat erabili beharko bagenu, 200 aldagai?
- Array-ak aldagai ugari taldekatzeko mekanismo bat baino ez da. Horrela, programatzaileak 100 aldagaitako *"multzoa"* oso era errazean defini dezake. Has gaitezen 17. listatuko adibidearekin

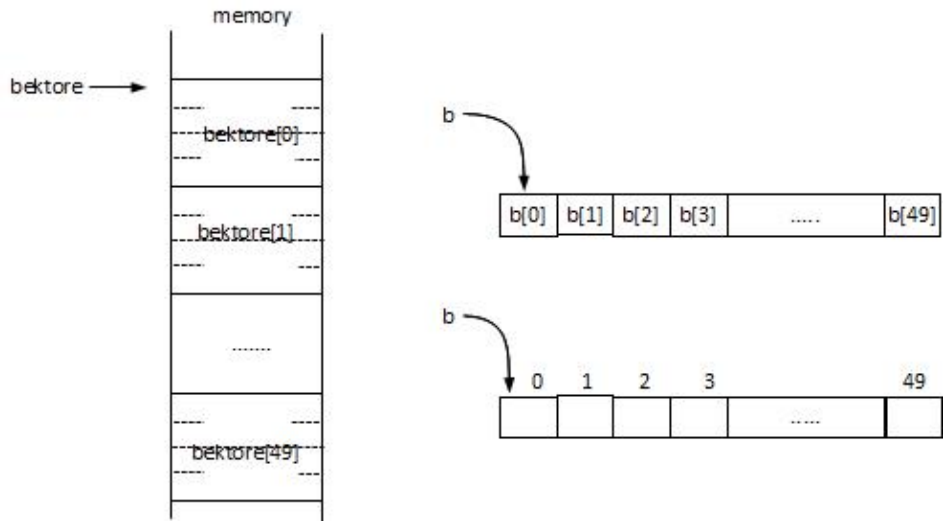
```
/*-------------------------------------------------------------------------
0 zenbakiaz amaituriko zenbaki zerrenda teklatutik irakurri eta zerrenda
berbera alderantziz pantailan idatziko duen programa. 0 - a zerrendaren
amaierako markatzat hartu eta ez zerrendako zenbakitzat. Bestalde, suposa
ezazu ez direla 50 zenbaki baino gehiago etorriko.
-------------------------------------------------------------------------*/
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int main(int argc, char* argv[])
{
  int i, n, zenbat;
  int bektore[50];
  char str[128];

  zenbat = 0;
  printf("Hasi zenbakiak sartzen, bat hilarako\n");
  do
  {
    fgets(str, 128, stdin);
    sscanf(str, "%d", &n);
    if (n != 0)
    {
      bektore[zenbat] = n;
      zenbat++;
    }
  } while (n != 0);
  printf("Zuk emandako zerrenda honako hau da:\n");
  for (i = 0; i < zenbat; i++) printf("%d ", bektore[i]);
  printf("\n\nTori orain zerrenda berbera baina alderantziz:\n");
  for (i = zenbat - 1; i >= 0; i--) printf("%d ", bektore[i]);
  printf("\n");
  printf("Sakatu \"return\" amaitzeko .....");
  fgets(str, 128, stdin);
  return 0;
}
```

Listing 17: 30bektoreMain.c fitxategia

4.1 Array baten definizioa eta Erabilera

- Array bat definitzerakoan elementuen mota eta osagai kopurua espezifikatu behar da. Sintaxia hurrengoa dugu:

```
int bektore[50];
```



Irudia 9: Array baten irudikapen ezberdinak

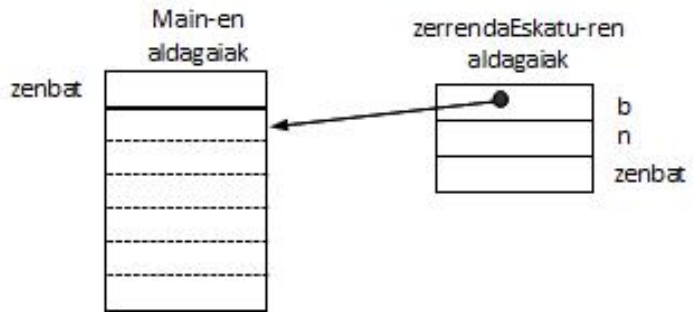
Definizio horren ondoren, `int` motako 50 aldagai gordetzeko lekua erreserbatuko da eta `bektore`, leku horren hasierako helbidea izango da (ikusi 9 irudia). Eredu errazago bat nahi badugu, aldagai-multzo osoaren izena dela pentsa genezake.

- Array baten osagaiak ordenatuta daude eta bakoitzak bere indizea du, indize hau 0-tik zenbatzen hasten delarik. Beraz, bektore batetako osagai bakoitza aldagai arrunt bat da zeinek izen berezi bat baino ez duen. Berau erabiltzeko array-aren izena eta indizea espezifikatu behar ditugu.

```
bektore[4]=bektore[5]+7;
for(i=0;i<50;i++) bektore[i]++;
```

4.2 Array-ak Parametro Gisa

- Funtzio bati array bat pasatzea posible da; baina:
  - Array-a bera pasa ordez bere hasierako helbidea pasatzen da (ikusi 10. irudia)
  - kontuz zeren eta funtzioak array-a aldatzen badu, deitu dionari ere aldatu dio zeren eta array berbera erabiltzen dute (batzuetan hau da nahi duguna).



Irudia 10: Parametroak array-ak direnean ez da array-a bera kopiatzen; bere hasierako helbidea baizik

- 17. listatuko adibidea berridatzi dezagun array-ekin egin dugun aktibitate bakoitzeko funtzio bat definituz:

```
int main(int argc, char* argv[])
{
  int zenbat;
  int bektore[50];
  char str[128];

  zenbat = zerrendaEskatu(bektore);
  printf("Zuk emandako emandako honako hau da:\n");
  zerrendaPantailaratu(bektore, zenbat);
  printf("\n\nTori orain zerrenda berbera baina alderantziz:\n");
  zerrendaIrauli(bektore, zenbat);
  zerrendaPantailaratu(bektore, zenbat);
  printf("Sakatu \"return\" amaitzeko .....");
  fgets(str, 128, stdin);
  return 0;
}

int zerrendaEskatu(int b[])
{
  int n, zenbat = 0;
  char str[128];

  printf("Hasi zenbakiak sartzen, bat hilarako\n");
  do
  {
    fgets(str, 128, stdin);
    sscanf(str, "%d", &n);
    if (n != 0)
    {
      b[zenbat] = n;
      zenbat++;
    }
  } while (n != 0);
  return zenbat;
}

void zerrendaIrauli(int b[], int dim)
{
  int i, tmp;

  for (i = 0; i < dim / 2; i++)
  {
    tmp = b[i];
    b[i] = b[dim - 1 - i];
    b[dim - 1 - i] = tmp;
  }
}

void zerrendaPantailaratu(int b[], int dim)
{
  int i;

  for (i = 0; i < dim; i++) printf("%d ", b[i]);
  printf("\n");
}
```

Listing 18: 31bektoreMain.c fitxategia

5 Karatereak eta Karaktere-Kateak (String)

5.1 Karaktereak

- Karaktereak, idazterakoan erabiltzen ditugun *"garabato"*, *"marrazki"* edo *sinboloak* baino ez dira. Baina, ordenagailu barruan nola kudeatzen dira?
  - Estandar ugari daude karaktereak klasifikatu eta zerrendatzeko. Horietako bat ASCII kodea dugu

dec	hex	symbol	dec	hex	symbol	dec	hex	symbol
32	20h	' '	48	30h	'0'	64	40h	'@'
33	21h	'!'	49	31h	'1'	65	41h	'A'
34	22h	'"'	50	32h	'2'	66	42h	'B'
35	23h	'#'	51	33h	'3'	67	43h	'C'
36	24h	'$'	52	34h	'4'	68	44h	'D'
37	25h	' %'	53	35h	'5'	69	45h	'E'
38	26h	'&'	54	36h	'6'	70	46h	'F'
39	27h	"'"	55	37h	'7'	71	47h	'G'
40	28h	'('	56	38h	'8'	72	48h	'H'
41	29h	')'	57	39h	'9'	73	49h	'I'
42	2Ah	'*'	58	3Ah	':'	74	50h	'J'
43	2Bh	'+'	59	3Bh	';'	75	51h	'k'
...			...			...		

Taula 4: ASCII taularen zati bat

  - ASCII edo beste edozein kodek zera ziurtatzen digute:
    - Digituek kode jarraiak dituztela '0'-tik hasita '9'-raino.
    - Letra larriek kode jarraiak dituztela 'A'-tik hasita 'Z'-raino.
    - Letra xeheek kode jarraiak dituztela 'a'-tik hasita 'z'-raino.
    - ...
- Ordenagailu barruan kodeak edo zenbakiak erabiltzen dira soilik. Kode bati dagokion sinboloa pantailan edo beste gailu batetan idazteko (marrazteko), orduan erabiliko da sinbolo edo marrazki hori soilik.
- C lengoaian karaktereak gordetzeko `char` mota dugu.

```
char c1, c2;

c1='A';              //c1 aldagaian 'a' letrari dagokion ASCII kodea gordeko du
c2='\66';            //c2 aldagaian 66 (54, 8 oinarrian adierazita dago) zenbakia gord
printf("%c␣%c", c1, c2);  //'A' eta '6' karaktereak pantailaratuko da
c1++;                // c1-en 'B' karakterearen ASCII kodea gordeko da
c2=c1+9;             // c2-en 'B' karakterea baino 9 posizio aurrerago dagoen
                     // karakterearen ASCII kodea gordeko da. Hau da, 'K' letrarena.
if(c1 < c2) ....     //c1 eta c2-n gordeta dauden ASCII kodeak konparatuko dira.
```

  - `char` mota 8 bitez adierazitako zenbaki osoak osatzen dute ([-128,127]).
  - `char` bati 1 gehitzerakoan ASCII taulako hurrengo karakterearen kodea lortuko da.
  - Bi `char` konparatzerakoan beraien ASCII kodeak konparatzen dira.

- Adibidea: letra xeheak hutsune batez bereizturik idazten duen kodea:

```
char c;

for(c='a';c<='z';c++) printf("␣%c␣",c);
```

- Adibidea: karaktere bat hizki xehea den hala ez esango digun kodea:

```
char c;
...
if((c>='a')&&(c<='z')) .....
```

5.2 Karaktere-Sekuentziak. String-ak

- C lengoaiak berak ez dauka ***karaktere-sekuentzia*** kontzepturik; baina bada hurrengo hitzarmena:
  - Karaktere-sekuentziak gordetzeko array-ak erabiltzea.
  - Karaktere sekuentzia horren amaiera `'\0'` karaktereak adieraztea. `'\0'` karakterea, 0 zenbakia dugu.
  - Beraz, 8 karakteretako sekuentzi bat gordetzeko 9 elementutako karaktere array-a behar dugu gutxienez.
- Hitzarmen hau ez dago erabili beharrik; baina string-ekin lan egiteko C lengoaiaren liburutegiko funtzio guztiek erabiltzen dute. Beraz, hauek aprobetxatzeko aukera galduko genuke.
- 19. listatuko adibidea erabiliko dugu oinarrizko zenbait komentario egiteko

```
/*-------------------------------------------------------
string.ak azaltzeko lehendabiziko adibidea
-------------------------------------------------------*/
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>
#include <string.h>

int main(int argc, char* argv[])
{
  char str1[128], str3[256];
  char str2[] = "abcdefghijk";
  unsigned int n, i;

  printf("Idatzi␣esaldi␣bat:␣");
  fgets(str1, 128, stdin);
  str1[strlen(str1)-1] = '\0';
  n = 0;
  for (i = 0; i<strlen(str1); i++)
    if (str1[i] == 'a') n++;
  printf("\n\"%s\"␣esaldian␣%d␣aldiz␣dago␣'a'␣letra\n\n", str1, n);
  n = 0;
  i = 0;
  while (str2[i] != '\0')
  {
    if (str2[i] == 'a') n++;
    i++;
  }
  printf("\""); printf(str2); printf("\"");
  printf("␣esaldian␣%d␣aldiz␣dago␣'a'␣letra\n\n", n);
```

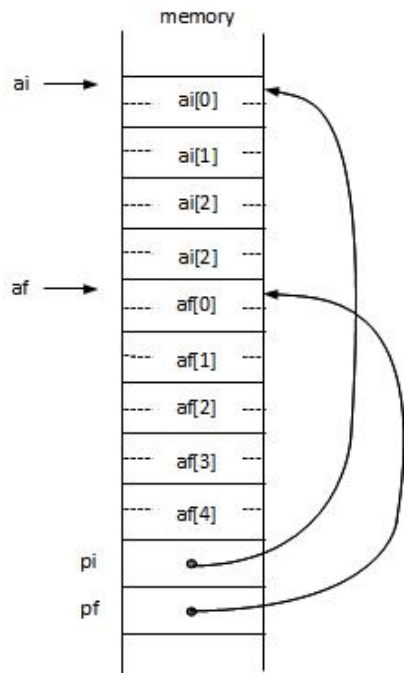
```
    strcpy(str3, str1);
    strcat(str3,str2);
    printf("erabilitako esaldiak elkartuz gero:\n\t\"%s\"\n\n", str3);
    printf("Sakatu \"return\" amaitzeko .....");
    fgets(str1, 128, stdin);
    return 0;
}
```

Listing 19: 40stringsMain.c fitxategia

- C lengoaiarekin batera edozein konpiladorek funtzioen liburutegi bat eskaintzen du: C lengoaiaren liburutegi estandarra deritzona. String-ekin lan egiten dutenen prototipoak `string.h` fitxategian daude eta hona hemen horietako zenbaitzuk:
  - `size_t strlen ( const char * str );`
    string bat pasatuz hau osatzen duten karaktere kopurua itzuliko du .
  - `char * strcpy ( char * destination, const char * source );`
    `destination` array-an `source`-n dagoen string-a kopiatuko du
  - `char * strcat ( char * destination, const char * source );`
    `destination` array-an dagoen string-ari `source`-n dagoena atsekituko dio
  - `int strcmp ( const char * str1, const char * str2 );`
    `str1` eta `str2` arrayetan dauden string-ak konparatuko ditu eta alfabetikoki str1 str2-ren aurretik badago, berdina bada edo aldabetikoki ondoren badago < 0 den balioa, 0 edo > 0 den balioa itzuliko du hurrenez hurren.

5.3 Erakusleei Buruzko Ohar Batzuk

- *Ez da ikasgai honen helburu erakusleak bere sakontasun osoan lantzea; baina C-ren liburutegia erabiltzerakoan sarritan agertuko zaizkigunez, horiek ondo ulertzeko behar duguna azalduko dugu.*
- ***Erakusleak*** memoriako helbide bat gordetzen duten aldagaiak dira.




```
int ai[4];
int *pi;
float af[5];
float *pf;

pi=ai;
pf=af;
pi[2]=5;              //<==>  ai[2]=5;
pf[4]=3;              //<==>  af[4]=3;
for(i=0;i<4;i++) ai[i]=i;
for(i=0;i<4;i++) pi[i]=i;
```

- `ai` jarraian dauden 4 aldagaitako array baten hasierako helbidea da.
- `af` jarraian dauden 5 aldagaitako array baten hasierako helbidea da.
- `pi` aldagaiak `int` motako aldagaien helbidean gordeko ditu.
- `pf` aldagaiak `float` motako aldagaien helbidean gordeko ditu.
- `ai[2]` edo `pi[2]`-k esanahi bera dute: `ai` helbidetik edo `pi` aldagaiak duen helbidetik hasita 2. `int`-ari egiten diote erreferentzia.
- `af[4]` edo `pf[4]`-kin gauza bera: `af` helbidetik edo `pf` aldagaiak duen helbidetik hasita 4. `float`-ari egiten diote erreferentzia.

- Funtzio bati array bat pasatzerakoan, ikusi dugu jada ez dela array-aren kopia bat egiten baizik eta array-aren hasierako helbidea pasatzen zaiola.

```
int zerrendalrakurri(int bektore[]);
void zerrendaPantailaratu(int bektore[], int dim);
```

  - `bektore` parametroa, izatez, erakusle bat baino ez da.
  - Beste era honetan ere idatz genezake;

```
int zerrendalrakuuri(int *bektore);
void zerrendaPantailaratu(int *bektore, int dim);
```

  - Idazteko moduak ez badu eraginik, zertarako bi modu? `[]` erabiltzen dugunean, parametroa array baten hasierako helbidea dela azpimarratzen dugu.
  - Beraz, C-re liburutegiko funtzioetan ez gaitezen arritu string.ak pasatzerakoan `[]` ordez erakusleak ikusteaz.

```
size_t strlen ( const char * str );
char * strcat ( char * destination, const char * source);
```

- Erakusleekin apur bat jolastuz:

```
int i,n, *pi;

i=3;
n=55;
pi=&i;          //pi-n i aldagaiaren helbidea gorde
*pi=(*pi)+4     //i=i+4;  efektu bera izango luke
pi=&n;          //pi-n n aldagaiaren helbidea gorde
(*pi)++;        //n++; sententziak efektu bera izango luke
```

5.4 `main` Funtzioaren Parametroak

- Orain artean, programa `main` izeneko funtzioa hasten dela esan dugu baina, zertarako bere parametro horiek?

```
int main(int argc, char * argv[]);
```

- Programa bat exekutatzeko hurrengo bi erak daude gutxienez:
  - Esploratzailearen bitartez programa hori aurkitu eta klik bikoitz egin bere gainean.
  - Konsola bat ireki, programa hori dagoen tokira joan, programaren izena idatzi , eta return sakatu
- komando bidez exekutatzerakoan, programaren izenaz gain parametroak ere pasa diezazkiokegu. Har dezagun adibide bezala 20 listatuko adibidea. Programa honek komando lerroa idaztitako hiru zenbakiren zkh kalkulatzen du.

```c
/*-------------------------------------------------------------
 komando lerro bidez 3 zenbaki pasa eta gauen zkh
kalkulatzen duen programa
-------------------------------------------------------------*/
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int zkh(int n, int m);

int main(int argc, char* argv[])
{
  int n1, n2, n3, x;
  char str[32];

  if (argc != 4)
  {
    printf("gaizki deitu nauzu. Niri deitzeko formatua hurreungoa da:\n\n");
    printf("\t%s <zenbaki> <zenbaki> <zenbaki>\n\n", argv[0]);
    return 1;
  }
  sscanf(argv[1], "%d", &n1);
  sscanf(argv[2], "%d", &n2);
  sscanf(argv[3], "%d", &n3);
  x = zkh(n1,n2);
  x = zkh(n3, x);
  printf("zkh(%d,%d,%d)=%d\n",n1,n2,n3,x);
  printf("Sakatu \"return\" amaitzeko .....");
  fgets(str, 32, stdin);
  return 0;
}

int zkh(int n, int m)
{
  int tmp;

  while (n%m != 0)
  {
    tmp = n%m;
    n = m;
    m = tmp;
  }
  return m;
}
```

Listing 20: 41mainsParams.c fitxategia

- `argc` parametroak komando lerroan izatzitako *"hitz"* adierazten du.
- `argv` parametroa string-en array bat da. Zehatzago esanda, array horretako elementuak komando lerroan idatzitako parametro bakoitzarekiko erakusteak dira.
- Adibidez, `41mainsParams 1024 56 1536` komandoa idatziz deituz gero `41mainsParams.exe` programari
  - `argc` 4 izango da
  - `argv[0]` "41mainsParams"
  - `argv[1]` "1024"
  - `argv[2]` "56"
  - `argv[3]` "1526"

6 Datu Mota Gehiago

6.1 Datu Motentzako Izen Berriak

- `typedef` eragilearen bitartez, datu motei izen berriak eman diezazkiekegu:

```
typedef unsigned char BYTE;

BYTE byteBat;
```

  Hemendik aurrera `BYTE` hitzak konpiladoreak ezagutzen duen datu mota berri bat izango da

- Ez dezagun pentsa `#define` moduko delarik:

```
#define BYTE unsigned char

BYTE byteBat;
```

- Nahiz eta aurreko bi aukerek antzekoa diruditen:
  - `#define` erabiltzen badugu, hau preprozesadorearen kontua da eta konpilatzen hasi aurretik `BYTE` hitza `unsigned char`-gatik ordezkatuko da. Hau da, edizio erraztasun bat baino ez da.
  - `typedef` erabilioz ordea, konpiladoreak berak ezagutzen duen datu mota bat sortu dugu.

6.2 Mota Zerrendatuak

- Demagun programa batetan hurrengo kontzeptuak adierazi nahi ditugula:
  - `{FALSE, TRUE}`
  - `{ASTELEHENA, ASTEARTEA, ..., IGANDEA}`

  Hauek gauzatzeko `{0, 1}` eta `{0, 1, ... 6}` zenbaki osoak erabili genitzake baina ...

- C lengoaian posible da mota zerrendatuak definitzea:

```
typedef enum E_ASTEKO_EGUNA{ASTELEHENA, ASTEARTEA, ..., IGANDEA} ASTEKO_EGUNA;
enum E_BOOLEAN{FALSE, TRUE};

ASTEKO_EGUNA eguna;
enum E_BOOLEAN boolearBat;
```

- Sortu berri ditugun datu moten izenak `enum E_ASTEKO_EGUNA` eta `enum E_BOOLEAN` dira. Dena den, `typedef` erabiliz, izen berri eta egokiago bat ere eman geniezaieke.

- Mota zerrendatuak `int` bidez gauzatzen ditu konpiladoreak; baina ez genuke honetan oinarritu behar. Gure adibidean `ASTELEHENA` 0 da, `ASTEARTEA` 1, etb.

6.3 Egiturak

- Ikusi dugu dagoeneko array-ak aldagai ugari taldekatzeko mekanismo bat dela. Baina kasu honetan aldagaiok mota berekoan izan behar dute. Egia esan, kasu askotan egokiak dira. Adibidez, bektoreak, zenbaki zerrendak, matrizeak, string-ak gordetzeko.

- Baina aldagai horiek mota ezberdinetakoa direnean? Adibidez, pertsona bati buruzko informazioa gordetzeko bere izen, abizen, tfno eta jaiotze data izan nahi ditugu. Kasu hauetarako C lengoaiak egiturak eskaintzen ditu:

```
typedef struct S_DATA
{
  unsigned char egun;
  unsigned char hile;
  unsigned int urte;
}DATA;

typede struct S_PERTSONA
{
  char izen[10];
  char abizen[20];
  int tfno;
  DATA jaiotzeEgun;
}PERTSONA;

   ......
  PERTSONA p;
  struct S_PERTSONA talde[5];

  strcpy(p.izen,"Martin");
  strcpy(p.abizen,"Berastegi");
  p.tfno=444444444;
  p.jaiotzeEgun.egun=4;
  p.jaiotzeEgun.hile=4;
  p.jaiotzeEgun.urte=1944;
  .......
  talde[0]=p;
  ........
  for(i=0;i<5;i++)
  {
    printf("␣%s␣%s␣tfno:%d␣jaiotze␣data:␣%d/%d/␣%d
␣␣␣␣␣␣␣␣␣␣␣␣talde[i].izen,␣talde[i].abizen,␣talde[i].tfno,␣talde[i].jaiotzeEgun.egun,
␣␣␣␣␣␣␣␣␣␣␣␣talde[i].jaiotzeEgun.hile,␣talde[i].jaiotzeEgun.urte\n");
  }
```

Listing 21: egitura adibideak

6.4 Bi Dimentsiotako Array-ak

- Orain arte dimentsio bakarreko array-ak erabili baditugu ere, posible da 2, 3 edo dimentsio gehiagokoak ere erabiltzea. Bi dimentsiokoak adibidez, egokiak dira matrizeak erabiltzeko.
- Erabil dezagun hurrengo adibidean:

```
    double m[3][4];     //3x4 double elementutako matrizea

    m[0][1]=8;
    m[2][1]=3*m[0][1];
    ........
    matrizealdatzi(m,3,4);
    .....

  void matrizealdatzi(double m[][4],int ilara, int zutabe)
  {
    int i,j;

    for(i=0;i<ilara;i++)
    {
      for(j=0;j<zutabe;j++) printf("␣%lf␣",m[i][j]);
      printf("\n");
  }
```

- Matrizeen kasuan, aldagai bat definitzekoan bi dimentsio adierazi behar dira.
- Matrizearen izena ez da aldagai osoa baizik eta matrizearen hasierako helbidea
- Funtzioei pasatzerakoan
  - ez da matrizea kopiatzen baizik eta matrizearen hasierako helbidea. Beraz, funtzio batek matrize bat aldatzen badu, pasa diogun matrizea aldatuko du.
  - Bere dimentsio guztiak espezifikatu behar dira lehena ezik.

A Kodifikazio-Arauak

```
int lehenaAlDa(int n)
{
  int zatitzaile;
  int lehenaDa=1;

  zatitzaile = 2;
  if (n < 2) lehenaDa=0;
  else
  {
    while ((n >= zatitzaile*zatitzaile) && (n % zatitzaile != 0))
    {
      zatitzaile++;
    }
    lehenaDa=(n < zatitzaile*zatitzaile);
  }
  return lehenaDa;
}
```

- Ahal dela C99 estandarraren araberako konpiladorea hautatuko da.
- Ilara batek ezin ditu 80 karaktere baino gehiago izan.
- Kode bloke bat zabaldu eta irekitzen duten {} giltzak elkarren parean agertuko dira eta blokea behar duen sententziaren azpian: `if`, `while`,...
- Adierazpen konplexuak daudenean ez oinarritu lehentasun arauetan; erabili parentesiak.
- `goto` eta `continue` ezin dira erabili. `break` ordea, `switch` sententzietan soilik.
- Indentatzeko 4 espazio (edo 2) erabiliko dira. Eta tabuladoreak ez ditugu inoiz erabiliko.
- Funtzioek orrialde bateko luzera izan dezakete asko jota (edo pantaila bat).
- Funtzio guztiek irteera puntu bakarra izango dute. Hau da, asko jota `return` bakarra, berau funtzioaren amaieran joango delarik. .