3. gaia: C PROGRAMAZIO LENGOAIA

HASIERAKO KONTZEPTUAK

C Programazio lengoaia 1972an sortu eta implementatu zen lehenengo aldiz. Helburua UNIX sistema eragilea idazteko lengoaia sortzea izan zen.

Mikrokonputagailuen garapenaren ondorioz C implementazio desberdin asko sortu ziren. Garraiogarritasuna galtzeko zorian zegoenez, ANSI-C estandarra sortu zen.

C lengoaia oso sinplea eta ahaltsua da. Gainera, konpilatu ostean C-n idatzitako programak oso eraginkorrak izaten dira (trinkoak -memoria zati txikia hartu- eta azkarrak).

Askotan entzungo dugu ere, erdi-mailako programazio lengoaia dela. Honen arrazoia da goi-mailako lengoaien elementuekin batera (datu motak etab.), behe-mailako lengoaia baten funtzionaltasuna eskaintzen digula: bit maneiua, erakusleak (memoriako helbideak)...

Programa baten egitura

C-n programak modulutan banatzen dira, bata modulu nagusia delarik. Moduluek funtzio desberdinak izan ditzakete, eta nagusia *main()* izena jasotzen du. Hasieran egingo ditugun programak, modulu bakarra eta funtzio bakarra izango dute, baina orokorrean ez da zertan horrela izan behar. Ondoren C programa baten egitura orokorra ikus daiteke: ("[" eta "]" artean dagoena hautazkoa da)

```
Liburutegiak
Konstanteak
Void main ( [parametroak])
{
        [Aldagai erazagupena]
        Gorputza /*Iruzkinak*/
}
```

Aldagaiak eta konstanteak programan zehar datuak gordetzeko erabiliko ditugun memoria zatiak baino ez dira; gorputza berriz agindu multzo bat da eta liburutegiak zer diren aurrerago ikusiko dugu.

Programan tabulazioak edo tarte hutsak sartzea ez da derrigorrezkoa baina bai komenigarria ulergarritasuna handitzeko, eta erroreak ditugunean azkarrago zuzendu ahal izateko.

Oinarrizko osagai horietaz gain, iruzkinak sar ditzakegu idazten ditugun programetan. Iruzkinak programatzailearentzako idazten diren oharrak dira, konpilatzaileak albo batera uzten ditu baina hau ere, tabulazioak bezala, garrantzitsua da programen ulergarritasuna handitzeko.

Iruzkinak Turbo-Cn egitura hau daukate:

```
/*nahi adina lerro*/
//lerro bakarra
```

Turbo C-n aginduak multzoka biltzeko giltzak ,{ }, erabiltzen dira. Honen bitartez, teorikoki agindu bakarra sar dezakegun tokietan agindu multzoak sar daitezkeelarik. Orokorrean Turbo C-ko agindu guztiak ;-rekin amaitu behar dute. Jarri behar ez direneko toki bakarra } ondoren da.

Identifikadoreak

Programa batek erabili ditzakeen objektuentzako izenak. Hauek hitz erreserbatuak, edo erabiltzaileak definitutakoak izan daitezke.

Hitz erreserbatuak konpiladorearentzat esanahi berezia duten hitzak dira, ondorengo taulan azaltzen dira:

ANSI estandarreko hitz erreserbatuak			
auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Turbo Cko hitz erreserbatuak								
Asm	_cs	_ds	_es	_AH	_AL	_AX	_BH	_BL
_ss	cdecl	far	huge	_BX	_CH	_CL	_CX	_DH
interrupt	near	pascal		_DL	_DX	_BP	_DI	_SI
				_SP	_DS	_CS	_ES	_SS

Erabiltzaileak definitutakoetan ondorengo arauak jarraitu behar dira:

- Identifikadoreak karaktere alfabetikoz eta digituz osatutako sekuentziak dira
- Gehienez 32 karaktere izan ditzakete identifikadoreek.
- Lehenengo osagaia letra bat edota _ izan behar du

Gainera, kontutan izan behar dugu ezin ditugula hitz erreserbatuak erabili, guk sortutako aldagaientzat, hauek esanahi berezia baitute eta esanahi horrekin soilik erabili daitezke. C-n maiuskulak eta minuskulak desberdintzen dira. Identifikadoreak aukeratzeko orduan zenbait arau jarraitzea komeni da:

- identifikadoreak esanguratsuak izan behar dute, horrela identifikadorea ikusi bezain laster jakingo dugu zer den bertan gordetzen dugun datua
- -Esanguratsuak bai baina ez luzeegiak bestela programa ulertzea zailagoa gertatuko zaigu.
- -Konstanteak maiuskulaz idatzi ohi dira eta aldagaiak minuskulaz
- -Aldagai eta konstanteentzat sustantiboak eta funtzioentzat aditzak.

Konstanteak

Konstanteak datuak dira, exekuzioaren zehar alda ezin daitezkeenak, hau da, behin balio bat hartzen dutenean exekuzio osoan zehar hori mantenduko dute. Hauek definitzeko funtzioa hasi baino lehen, hau da *void main* baino lehen, *#define izena balioa* motako sententzia bat jarri behar da. Izena identifikadore bat da eta beraz, berau erabakitzeko orduan identifikadoreak definitzeko arauak jarraitu behar ditugu.

Adibidez: *#define PI 3.1416*

Aldagaiak

Aldagaiak ere datuak dira, baina aurrekoekin alderatuz esan behar da hauek exekuzioan zehar alda daitezkeela. Aldagaiek hiru ezaugarri dauzkate:izena, mota eta balioa

- Izena: identifikadore bat
- Mota:balio multzoa eta haiekin egin daitezkeen eragiketak.
- Balioa:Une zehatz batean daukana.

Aldagaiak erabiltzeko zenbait pauso jarraitu behar ditugu:

- Erazagupena: Aldagaien erazagupenean zein aldagai (izena eta mota) erabiliko ditugun adieraziko dugu. Hau adierazteko *mota izena;* edo *mota izen1, izen2...izenn;* aginduak erabiliko ditugu. Agindu hauek behar adina aldiz jar daitezke.
- Hasieraketa: Erazagututako aldagaiei hasierako balio bat esleitzean datza. *mota izena=balioa;* edo *mota izen1, izen2...izenn=balioa;* egiturak erabiliz egin daiteke hau. Hasieraketa ez da derrigorrezkoa, aurreko fasearekin nahikoa da, baina guztiz gomendagarria da hasieraketa egitea ez-ustekorik ez izateko.

Ondoren, programan zehar nahi dugunean haien balioa aldatu edo atzitu dezakegu.

Datu-motak

Datu motek aldagai batek izan ditzakeen balio multzoa eta aldagai horrekin egin daitezkeen eragiketak definitzen ditu. Turbo-Cn dauzkagun oinarrizko motak, okupatzen duten memoria zatia eta izan ditzaketen balio multzoa ondorengo taulan azaltzen dira:

Mota	BYTE/BITak memorian	Balio multzoa
char	1 - 8	-128 ... 127
int	2 - 16	-32768 ... 32767
float	4 - 32	±3.4E-38 ... ±3.4E+38
double	8 - 64	±1.7E-308 ... ±1.7E+308
void	0 - 0	Baliorik gabea

Mota hauei *signed*, *unsigned*, *long* eta *short* modifikatzaileak gehi diezaizkiekegu. Bereziki erabilgarriak dira ondorengo konbinazioak

Mota	BYTE/BITak	Balio multzoa
Unsigned char	1 - 8	0 … 255
Unsigned int	2 - 16	0 .. 65535
long int	4 - 32	-2.147.483.648 … 2.147.483.647
Unsigned long int	4 - 32	0 … 4.294.967.295
long float ≅ double	8 - 64	±1.7E-308 ... ±1.7E+308

Mota batek okupatzen duen memoria zatia *sizeof (datua);* aginduaren bitartez lor dezakegu. Mota hauetako aldagai eta konstanteekin lan egiteko erabil ditzakegun eragileak multzoetan bana ditzakegu: aritmetikoak (datu numerikoekin lan egiteko), erlaziozkoak (zenbaki eta karaktereak konparatzeko), logikoak (espresio logikoak), bit-maneiukoak eta bereziak.

Kontutan izan behar da, oinarrizkoak ez diren datu motek beraien eragiketak izango dituztela, hauek behar ditugun heinean ikusiko ditugu.

Eragile bereziak

ERAGIKETA	ERAGILEA	FORMATUA	AZALPENA
Casting	(mota)	(mota)a	a aldagaia *mota*-ra pasatzen du
Esleipena	=	a=b	a-ri b-ren balioa esleitzen dio
Tamaina	sizeof()	sizeof(a)	a-k okupatzen dituen byte kopurua adierazten du

Eragile aritmetikoak:

ERAGIKETA	ERAGILEA	FORMATUA	AZALPENA
Ukapena	-	-X	X-en balioa zeinuz aldatu
Batuketa	+	X+Y	X gehi Y
Kenketa	-	X-Y	X ken Y
Biderkaketa	*	X*Y	X bider Y
Zatiketa	/	X/Y	X zati Y
Modulua	%	X%Y	X/Y -ren hondarra
Inkrementua	++	X++, ++X	X=X+1
Dekrementua	--	X--,--X	X=X-1

Inkremetuan eta dekrementuan eragileak aurretik edo atzetik jarrita ere eragiketa berdina egiten du; arazoa da zerbait konplexuagoaren barruan sartzen dugunean ikusten da. Adibidez, demagun x aldagaiaren balioa 6 dela; y=++x agindua burutzerakoan y aldagaiak 7 balioa hartuko du eta y=x++ jartzen badugu aldiz y aldagaiak 6 balioa hartuko du. Hau da, lehenengo kasuan inkrementua esleipena baino lehen egiten du eta bigarrenean alderantziz.

Zatiketa eragiketarekin, bi zenbakiak osokoak badira emango duen zatiketa osokoa izango da, zenbaki errealak ditugunean aldiz, zatiketaren emaitza erreala izango da. Modulu eragiketa osoko eragigaiekin baino ezin da erabili.

Eragile konbinatuak

Eragile hauek aurreko eragileak eta esleipena konbinatzen dituzte. Konbinatuen kasuan, emaitza eta lehen eragigaia beti aldagai berbera dira.

+= -= *= /= %=

x+=1 → x aldagaiak 5 balioa bazuen, eragiketa honen ostean 6 balioa hartuko du

Erlazio-eragileak eta eragile lojikoak

Erlazio edo logikako espresioak ebaluatzeko balio dute. FALSE balioa 0-koa da eta gainontzekoak TRUE.

ERAGIKETA	ERAGILEA	FORMATUA	AZALPENA
Handiago	>	a>b	Baldin a>b egiazko(1) bestela faltsua (0)
Txikiago	<	a<b	Baldin a<b 1, bestela 0
Berdin	= =	a= = b	Baldin a= =b 1, bestela 0
Handiago edo berdin	>=	a>=b	Baldin a>=b 1, bestela 0
Txikiago edo berdin	<=	a<=b	Baldin a<=b 1, bestela 0
Ezberdin	!=	a!=b	Baldin a ezberdin b 1, bestela 0
Eta	&&	e1&&e2	Baldin e1 eta e2 1, bestela 0
Edo	||	e1||e2	Baldin e1 edo e2 1, bestela 0
Ez	!	!e1	Baldin e1 0, bestela 1

Erlaziozko ez den espresio bat daukagunean baldintza moduan, espresioaren ebaluaketa 0 denean faltsutzat hartzen da eta gainerako kasuetan egiazkotzat.

Bit-maneiurako eragileak

C lengoaiaren ezaugarrien artean bitak atzitzeko gaitasuna ematen digula aipatu genuen, horretarako zenbait eragile eskaintzen dizkigularik. Horietako batzuk ondorengo taulan azaltzen dira:

ERAGIKETA	ERAGILEA	FORMATUA	AZALPENA
AND	&	X&Y	X AND Y (Bit mailako eragiketa)
OR	|	X|Y	X OR Y
XOR	^	X^Y	X XOR Y

Hauek datuen bitak atzitzen dituzte zuzenean, kontuz izan behar dugu ez okertzeko eta albo ondoriorik ez izateko.

Adierazpenak

Aldagai eta eragileak konbinaturik sortzen dituzte adierazpenak. Balio bat kalkulatzeko formulak bezala ikus ditzakegu. Amaieran ; jarririk aginduetan bilakatzen dira.

Eragiketa bat baino gehiago daukagunean, lehentasun taularen arabera ebaluatzen dira adierazpenak. Lehentasun handienetik txikienera joanez. Lehentasun taula ondoren azaltzen dena da:

Maila	**Eragiketak**	**Ebaluazioa**
1(altuena)	**( ) [ ] -> .**	⟵
2	**! ~ ++ -- (mota) * & sizeof( )**	⟶
3	*** / %**	⟵
4	**+ -**	⟵
5	**<< >>**	⟵
6	**< <= > >=**	⟵
7	**== !=**	⟵
8	**&**	⟵
9	^	⟵
10	|	⟵
11	**&&**	⟵
12	||	⟵
13	**?:**	⟶
14	**= += -= *= /=**	⟶

Bi datu mota desberdineko aldagaien artean eragiketa bat egin nahi dugunean, konpiladoreak automatikoki datu-mota aldaketak edo castingak egiten ditu, datu-mota aldaketa hau tamaina handienerantz egiten du beti, baina gero emaitza aldagai bati esleitzerakoan ez du kontrolik egiten eta informazioa galdu dezakegu. Programatzaileak gauza hauek kontutan izan behar ditu.

Mota	**char**	**int**	**float**	**double**
Tamaina	1	2	4	8
Konbertsio ona	---->------>----->----			
Konbertsio txarra*	----<------<-----<----			

* informazio galera gertatzen da

Konbertsioak inplizituak izan daitezke (konpiladoreak egindakoak) edota esplizituak guk egiteko adierazten badiogu. Casting-ak egiteko *(mota_izena) aldagaia;* agindua erabiltzen da eta honek, aldagaia guk adierazitako motara (mota_izena) bihurtuko du.

DATUEN SARRERA ETA IRTEERA

Algoritmoak ikusterakoan adierazi genuen konputagailuari dena esan behar diogula, eta gainera ez duela ezer ikasten. Suposatu orain teklatutik (edo sarrerako periferiko batetik) zerbait irakurri nahi dugula. Guk funtzio bat egin nahiko bagenu konputagailuaren hardwareari buruzko gauzak jakin beharko genituzke, lan guzti hori ekiditeko badauzkagu liburutegi estandarrak.

Liburutegi estandarra erabiltzeko prest dagoen funtzio multzo bat da, dagoeneko sortuta eta konpilatuta daudenak.

Hauek erabiltzeko konpilatzaileari adierazi behar diogu bai erabiliko ditugula eta baita non aurki ditzakeen, horretarako funtzio hasieran *#include <liburutegia.h>* motako agindua erabiliko dugu.

Liburutegi estandar ugari daude eta bakoitzak funtzionalitate bat eskaintzen digu. Adibide moduan sarrera eta irteerak kudeatzen dituena ikusiko dugu.

Sarrera-irteerako funtzioak

Sarrera eta irteerako funtzio aurredefinitu gehienak stdio.h izeneko liburutegian daude, beraz, erabiliko ditugula adierazteko, *#include <stdio.h>* jarri beharko dugu. Liburutegi honetan funtzio ugari daude, haien artean:

- Irteerako funtzioak
  - *printf ();*
  - *putchar ();*
  - *puts ();*
- Sarrerako funtzioa:
  - *scanf ();*
  - *getchar ();*
  - *getch ();*
  - *getche ();*
  - *gets ();*

Ondoren banan-banan azaltzen dira funtzio hauek

Putchar()

Funtzio honek parametro gisa pasatako karakterea, edo emaitza bezala karaktere bat itzultzen duen espresioak, idazten du. Ez da hurrengo lerrora pasatzen. Adibidez:

```
char A = 'k', B = 72;
putchar(A);          /* k karakterea idazten du */
putchar(B);          /* ASCII 72 (H) idazten du */
putchar('*');        /*  * bat idatziko du */
putchar('\n');       /* \n  idatziko du*/
putchar('a'+32);     /* A idatziko du*/
....;
```

Puts()

Parametro gisa pasatako karaktere katea idazten du, ondoren hurrengo lerrora pasatzen da . Adibidez:

```
char A[80], B[] = "kaixo denei.";
puts(B);             /* B karaktere katea eta CR bat idatziko ditu*/
puts("kaixo ...");   /* Pantailan kaixo ...eta CR bat idatziko ditu.*/
puts("");            /* Kate hutsa eta CR bat idatziko ditu.*/
puts("\n");          /* 2CR idatziko ditu */
```

Printf([“Kontrol_katea”] [,parametroak]*);
Kontrol katea, existitzen bada, karaktere testualak eta formatu aginduez osatuta dago. Karaktereak zuzenean idazten dira eta formatu aginduek parametroak kontrol katean nola sartuko diren adierazten dute.

Parametroak: aldagai, espresio eta konstanteen lista komaz banaturik. Formatu agindu eta parametro kopurua berdina izan behar dute eta ordenan bikotean bilduko dira.
Formatu aginduak *% [aldatzailea] agindua* motakoak izango dira:

Agindu posibleak:

c	karakterea	s	karaktere-katea
d	osokoa	f	erreala
e	idazkera zientifikoa		

Aldatzaileak:

n1: Parametroak okupatuko duen tokia adierazten duen modifikatzailea, tokirik sobran balego eskuinera justifikatuko luke.Toki gehiago behar badu moztuko du.

-n1: berdina baina justifikazioa ezkerraldera eginez.

0n1: Eskuinera justifikatuta eta toki hutsak badaude 0z beteko du ezkerraldeko tartea

n1.n2: zenbaki erreal eta kateentzako:

n1: eremu tamaina

n2: dezimal kopurua

Adibide batzuk:

```
int x=3;
float y = 2.56;
char A=’a’, s1[80]=”Kaixo ...”;
...
printf(“Testua bakarrik ...”);
printf(“\nTestua eta datuak: %10d \t %-15d”,x,x+10);
printf(“%d - %f - %10.3lf”,5,y,3.141695);
printf(“Var. A = %5c \nCad. s1 = %45.6s \n%s”,A,s1,”azkena”);
printf(s1);
```

Getchar();
Teklatuko bufferraren lehenengo karakterea irakurri eta itzultzen du.

```
char A;
...;
A = getchar();          /*return sakatzean, teklatuko lehenengo karakterea  irakurtzen du*/
```

Getch();
Karaktere bat irakurri eta itzultzen du. Funtzio hau conio.h izeneko liburutegian dago definituta

```
char a;
....;
a = getch();
```

Getche();
Aurrekoaren berdina baina pantailan oihartzuna egiten

Gets(parametroak);
Kate bat irakurtzen du eta bere parametroan gordetzen du.

```
char s1[80];
....;
gets(s1);
```

Scanf([“Kontrol_katea”] [,parametroak]*);
printf() funtzioa bezalakoa baina karaktereak irakurtzeko. Adibidez:

```
scanf (“%d,%d”, &altuera, &adina);
```

FLUXU-KONTROLA

Orain arte ikusitakoan aginduak bata bestearen ostean egikaritzen ditugu, honek egin ditzakegun gauza kopurua izugarri murrizten du.

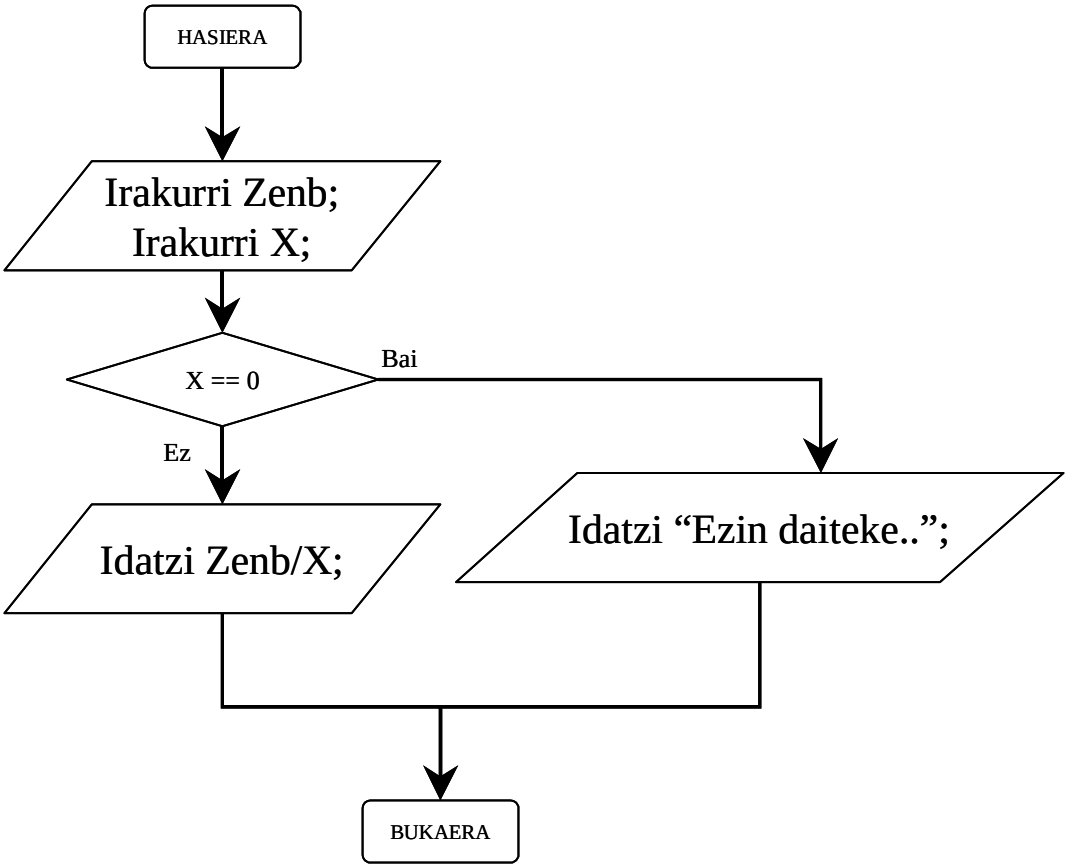
Pentsatu adibidez esaten digutela bi zenbaki, zenb eta X ditugula eta zenb/X emaitza itzuli behar dugula. X aldagaiak 0 balioa badu, errorea emango digu eta orain arte ikusitakoarekin ez dugu kontrolatzeko modurik; beraz, fluxuaren orden sekuentziala aldatuko duten aginduak behar ditugu hemen: Baldintzazkoak eta errepikakorrak.

Baldintzak

Aurreko arazoa algoritmo honek ebatziko luke:

```
HASIERA
IRAKURRI Zenb
IRAKURRI X;
BADA (X == 0)
        IDATZI ("ezin daiteke zatiketa egin")
BESTELA
        Emaitza = Zenb / X;
        IDATZI (Emaitza);
BUK_BADA
BUKAERA
```

Fluxu diagraman aldiz:

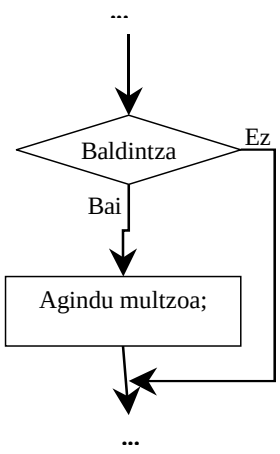


TurboC-k horrelako gauzak egiteko if eta switch aginduak eskaintzen dizkigu

If sententzia

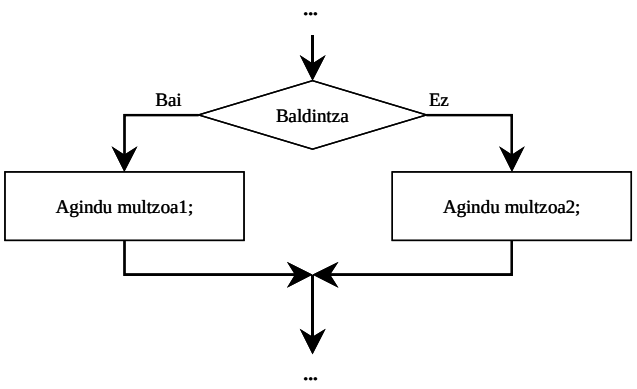
If sententzia modu desberdinetan erabili dezakegu: baldintza sinpleak edota baldintza bikoitzak inplementatzeko.

Egitura baldintzatua sinplea:

Fluxu diagrama	C kodea
	... if (baldintza) { Agindu multzoa; } ...

Baldintza betetzen denean *Agindu multzoa* egikarituko da eta bestela ez da ezer egingo. *Agindu multzoa* agindu sinple bat edo agindu anitz izan daiteke.

Egitura baldintzatua bikoitza:

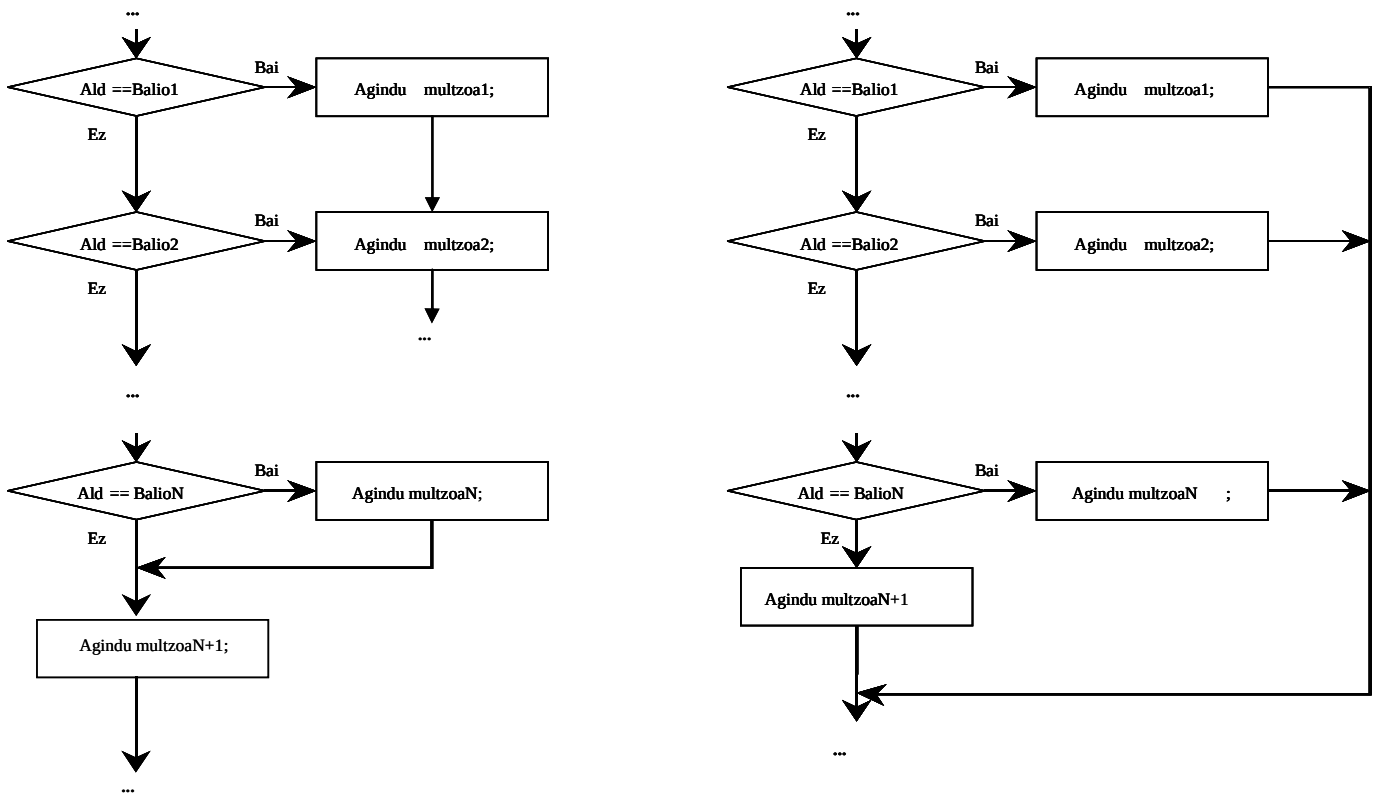
Fluxu diagrama	Cko kodea
	... if (baldintza) { Agindu multzoa1; } else { Agindu multzoa2; } ...

Baldintza betetzen denean *Agindu multzoa1* egikarituko da eta bestela *Agindu multzoa2*. *Agindu multzoa2* zatia hutsik dagoenean egitura baldintzatua sinplearen kasuan gaude. Gainera, egitura hau bi if sinpleen kateaketa bezala adierazi daiteke.

Switch sententzia

Aukera gehiagoko baldintzak lotzeko, if asko habiratu ditzakegu, baina errepikakorra gerta daiteke. Horregatik sortzen da switch sententzia. Kontutan izan hala ere switch-ekin egin daitekeen guztia habiratutako if-ekin egin dezakegula.

Fluxu diagrama



C kodea

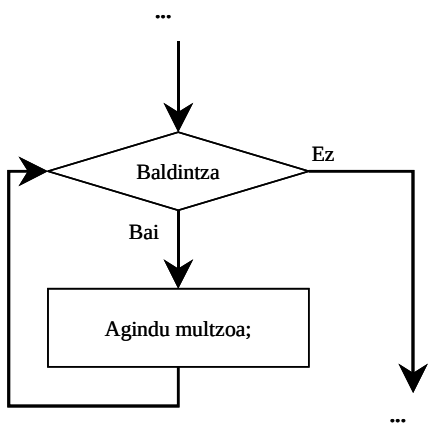
```
...
switch (aldagaia)
{
        case balio-1:
           { Agindu multzoa1;
            [break;]}
        case balio-2:
           { Agindu multzoa2;
            [break;]}
            .  .  .
        case balio-n:
            { Agindu multzoaN;
           [break;]}
        [default: {Agindu multzoaN+1;}]
}
...
```

Kontrol aldagaia *char* edo *int* motakoa baino ezin da izan. Baldintza betetzen den edo ez konprobatuz goaz, betetzen deneko kasu bat topatzen dugunean hori egikarituko du eta ondoren hurrengoak begiratzen jarraituko du ez baldin badiogu *break* agindu baten bitartez *switch*etik ateratzeko esaten. *Default* baldintza hautazkoa da eta gainontzeko baldintzarik betetzen ez denean sartuko da egikaritzapena zati honetan.

Egitura errepikakorrak

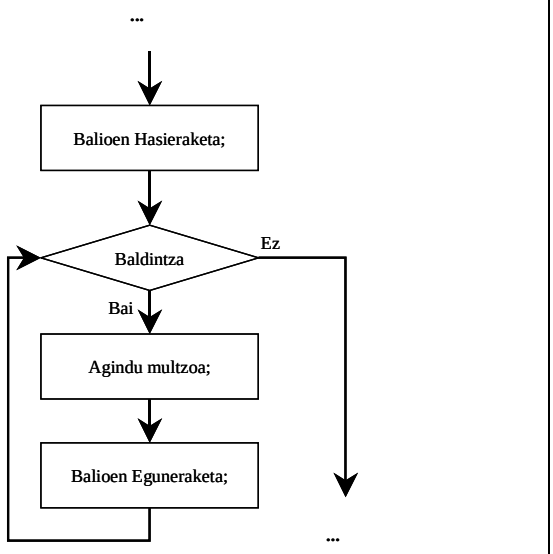
Batzutan baldintzekin ez zaigu nahiko fluxua kontrolatzeko. Imajinatu gauza bera askotan egin nahi dugula edota baldintza bat bete arte zerbait egin nahi dugula. Horretarako egitura errepikakorrak sortzen dira. Hauek zerbait egiten dute behin eta berriz baldintza bat bete arte edota baldintza betetzen den bitartean. Hemen ikusiko ditugun sententziak while, for eta do-while izango dira.

While sententzia

Fluxu diagrama	Cko kodea
	... while (baldintza) { *Agindu multzoa;* } ...

Baldintza betetzen den bitartean agindu batzuk egikaritzen ditugu. Baldintza hasieran konprobatzen denez gerta daiteke while-aren barruko aginduak inoiz ez egikaritzea. While barruan baldintzaren balioa aldatzen joango den agindu bat behar dugu begizta amaigabe batean sar ez dadin programa.

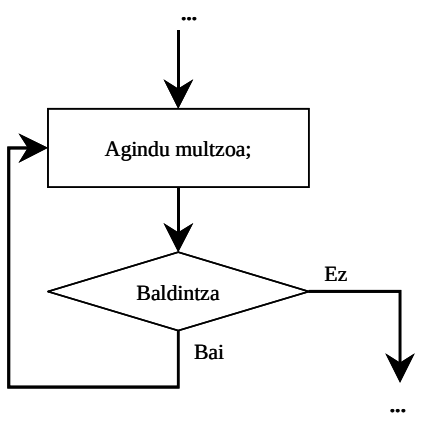
For sententzia

Fluxu diagrama	Cko kodea
	... for([sententzia1];[sententzia2];[sententzia3]) { *Agindu multzoa*; } ...

-Sententzia1: balioen hasieraketak, komaz banaturik. Balioen Hasieraketa agindu multzoaren parekoak.
-Sententzia2: Begiztan mantentzeko baldintza(k). Fluxu diagraman agertzen den Baldintza parekoa.

-Sententzia3: balioen eguneraketa, besteak beste baldintzan parte hartzen duten aldagaien batzuen balioak aldatuko dira. Balioen Eguneraketa agindu multzoaren parekoak dira komaz banaturik.

Do-while sententzia

Fluxu diagrama	Cko kodea
	... do { *Agindu multzoa;* } while (baldintza); ...

Aginduak egikarituko dira bukaeran azaltzen den baldintza betetzen den bitartean. Baldintza amaieran konprobatzen da, beraz, begiztaren barruko aginduak behintzat behin egikarituko dira. Begizta barruan, baldintzako aldagaia aldatzen joango den agindua behar dugu amaiera gabeko begizta batean sar ez gaitezen.

TAULAK

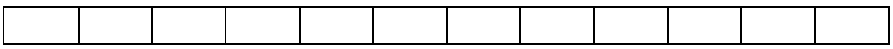
Taula bat mota konposatuko aldagai bat da, non osagai homogeneoak sar ditzakegun. Ideia mota berdineko osagai askoren tratamenduan datza. Taulen bitartez datuak modu erlazionatu batean sor ditzakegu eta horrek lana errazten du.

Orokorrean, bektore, matrize, array, katea eta string erabiltzen dira taularen sinonimo gisa.

Imajinatu urte bateko hilabete bakoitzeko euri jasa gorde nahi dugula. Orain artekoa ikusitakoarekin 12 aldagai sortu beharko genituzke, oso eraginkorra ez dena eragiketak egiteko orduan. Honi soluzio eraginkor eta apropos bat emateko sortzen dira taulak. Taula bat aldagai multzo bat bezala ikusi dezakegu. Taulak dimentsio desberdinetakoak izan daitezke.

Dimentsio bateko taulak

Dimentsio bakarreko taulak bektore batekin konpara dezakegu. Ondoren azaltzen den irudia hamabi osagaiko taula bat da.



Honekin, aldagai bakar batean urte bateko hilabete bakoitzean eroritako euria gorde dezakegu.

Orain arte bezala, horrelako aldagaiak erabiltzeko lehendabizi erazagutu behar ditugu. Taulen kasuan honela egiten da:

mota izena [kopurua];

Motak taularen osagai bakoitza zein motatakoa den adierazten du; kopuruak adierazten du zenbat osagai izan ditzakegun

Orain, hilabete bakoitzeko euri kopurua gordetzeko honelako aldagai bat definitu beharko dugu:

int euri_kantitatea [12];

Suposatu orain, taula definitzearekin batera taula hasieratu nahi dugula. Hori egiteko honelako zerbait egin genezake:

i*nt euri_kantitatea [12] ={2,3,4,5,6,7,8,9,0,10,11,12};*

Hilabete zehatz bateko datua atzitzeko (i. hilabetekoa) :*euri_kantitatea [i-1];* idatzi beharko dugu.

i-1 jarri dugu indize moduan, C-n indizeak 0-n hasten direlako eta osagai_kopura-1 arte iristen delako.

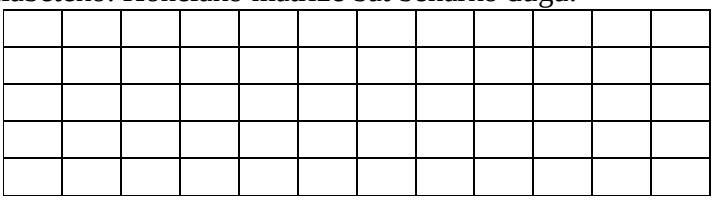
Hilabete zehatz bati (i. hilabeteari) balio bat emateko: *euri_kantitatea [i-1]=456*;

Taula baten tamaina konstantea denez, egokia da oso programazioan konstante bezala definitzea, Adibidez:

```
#define Hilabeteak 12
.....
int euri_kantitatea [Hilabeteak];
```

Bi dimentsiotako taulak

Bi dimentsioetako taulak matrizeekin konpara ditzakegu. Suposatu orain, azken bost urtetako euri jasa gorde nahi dugula, hilabeteko. Honelako matrize bat beharko dugu:



Hauek definitzeko honelako sententziak erabili beharko ditugu:

mota izena [lerroak] [zutabeak];

Gure kasuan*: int euria_azken_urteetan [5] [12];*

Hasieratzeko orduan honela egingo dugu:

int euria_azken_urteetan [5] [12]= {1,25,11,....,80};

Orain suposatu lehenengo urteko abuztuan eroritako euri kopurua jakin nahi dugula. Horretarako hondokoa idatzi beharko dugu:

euria_azken_urteetan [0] [7];

Ariketa: Sortu taula bat azken 5 urteetako euri jasa gordetzeko. Programak datu horiek irakurri eta ondorengo bilaketak eginbehar ditu: Lehenengo urteko hilabete guztietako tenperatura handiena izan duen hilabetea, Urte guztian abuztuan jasotako euri kantitatea pantailaratu.

```
#include <stdio.h>
#include <conio.h>
#define Urte 5
#define Hila 12
void main(){
int lerro, zutabe;
int handiena;
int euria_urte[Urte][Hila];
clrscr ();
//Balioen irakurketa
for (lerro=0;lerro<Urte;lerro++)
    for (zutabe=0;zutabe<Hila;zutabe++)
          scanf ("%d", &euria_urte[lerro][zutabe]);
//lehenengo urteko maximoa
handiena=euria_urte[0][0];
for (lerro=1;lerro<Hila;lerro++)
  if (euria_urte[0][lerro]>handiena)
    handiena=euria_urte[0][lerro];
 printf ("Handiena %d da  ", handiena);
 //urte guztietan abuztuan eroritako euria
 for (zutabe=0;zutabe<Urte;zutabe++)
   printf ("%d urtean %d litro",
                      zutabe,euria_urte[zutabe][7]);
}
```

Dimentsio anitzeko taulak

Nahi adina dimentsiotako taulak sor ditzakegu baina orokorrean ez dira erabiltzen bi dimentsio baino gehiago dauzkaten taulak.

L dimentsioetako taulak definitzeko honelako sententzia erabili beharko dugu.

Mota izena [d1] [d2]...[dl];

Asignatura honetan ez dugu taula mota hau landuko.

Karaktere kateak edo String-ak

Karaktereez osatutako dimentsio bateko taula da string bat; taula arrunt batekiko diferentzia amaieran ‘\0’ bat gordetzen duela da. Azken karaktere hori sartzen da taularen tamaina aldakorra denean amaiera detektatu ahal izateko.

Mota honetako aldagaien deklarazioa taula arruntena bezalakoa da, baita bere barne errepresentazioa ere.

char izenburua [20] = “Urteko salmenta” /*konpilatzaileak zuzenean gehitzen dio \0*/

Kontutan izan amaieran \0 daukagunez beti, X karakteretako kate batentzako *char izena [X+1];* jarri beharko dugula, hau da, X+1 toki beharko ditugu.

String-ak lantzeko funtzioak

Badago string.h izeneko liburutegi bat karaktere kateekin lan egiteko funtzio ugari dauzkana, horietako batzuk:

- strlen(s) : s-ren luzera idatziko du. (\n kontutan hartu gabe)
- strcpy (s1,s2); s1-en s2 kopiatuko du
- strcmp (s1, s2): s1 eta s2 berdinak badira (alfabetikoki) 0 itzuliko du; baldin s1<s2 bada orduan zenbaki negatiboa eta s1>s2 bada orduan zenbaki positiboa
- strcat (s1,s2): s1-en amaieran s2 kateatuko du eta s1-en gordetzen du emaitza
- gets(aldagaia):parametro bezala pasatako karaktere kate bat teklatutik irakurtzen du
- puts (aldagaia): parametro gisa pasatako katea idazten du pantailan.

ERAKUSLEAK

ERAKUSLEA: beste datu baten helbidea gordetzen duen elementua. Beraz, beste datua zeharka maneiatzeko erabiltzen da. Erakusle aldagaiak memoriaren helbide bat gordetzen du, memorian bi byte hartzen ditu. Era honetan erazagutzen da:
mota *izena;
mota *izena = helbidea;

Mota: Ze datu-mota gordetzen den erakusten den memoria-helbidean. Erazagupen honek (int *p1;) p1 aldagaiak izango duen helbidean osoko zenbaki bat gordetzen dela adierazten du.
Helbidea: Datu zehatz horrekin hasieratzen da erakusle aldagaia.
***:** Izartxoa ez da izenaren zati bat, erakusle bat dela adierazteko erabiltzen den marka baizik. ADI¡ Izartxoa hiru gauza desberdinetarako erabiltzen da:

1. Erakusle bat erazagutzeko: float *Erak = &ZenbErreala;
2. Erakusleak seinalatzen duen memoria-posizioa atzitzeko: *Erak = 4.5;
3. Biderkaketa eragiketa adierazteko: ZenbErreala = 1.2 * 3.8;

Scanf eragiketan erabili izan dugu &. Honek aldagai baten helbidea bueltatzen du. Beraz, &ZenbErreala, ZenbErreala aldagaiaren helbidea bueltatzen du.

ADIBIDEAK:
Suposatu dezagun memoriaren hitz-tamaina Byte batekoa dela. Programan aldagai erazagupen hau egingo bagenu:

```
int x= 5, y = 3, *p1, *p2 = &x;
char *p3, A= 'H';
float *p4, z1 = 3.14;
```

Memoria era honetan geratuko litzateke:

Memoria-helbidea	Memorian gordeta dauden datuak	Aldagaia
...		...
65000 65001	5	x
65002 65003	3	y
65004 65005	??	p1
65006 65007	65000	p2
65008 65009	???	p3
65010	'H'	A
65011 65012	????	p4
65013 65014 65015 65016	3.14	z1
...	...	...

Agindu hauek aldaketak sortzen dituzte memorian:

p1 = &y; ➔

Memoria-helbidea	Memorian gordeta dauden datuak	Aldagaia
...		...
65000	5	x
65001		
65002	3	y
65003		
65004	**65002**	p1
65005		
65006	65000	p2
65007		
65008	???	p3
65009		
65010	'H'	A
65011	????	p4
65012		
65013	3.14	z1
65014		
65015		
65016		

x= *p1 * *p2; ➔

Memoria-helbidea	Memorian gordeta dauden datuak	Aldagaia
...		...
65000	**15**	x
65001		
65002	3	y
65003		
65004	65002	p1
65005		
65006	65000	p2
65007		
65008	???	p3
65009		
65010	'H'	A
65011	????	p4
65012		
65013	3.14	z1
65014		
65015		
65016		

*p1= *p2; ➔

Memoria-helbidea	Memorian gordeta dauden datuak	Aldagaia
...		...
65000	15	x
65001		
65002	**15**	y
65003		
65004	65002	p1
65005		
65006	65000	p2
65007		
65008	???	p3
65009		
65010	'H'	A
65011	????	p4
65012		
65013	3.14	z1
65014		
65015		
65016		

*p2 = 10 + (y - *p1); ➔

Memoria-helbidea	Memorian gordeta dauden datuak	Aldagaia
...		...
65000 65001	**10**	x
65002 65003	15	y
65004 65005	65002	p1
65006 65007	65000	p2
65008 65009	???	p3
65010	'H'	A
65011 65012	????	p4
65013 65014 65015 65016	3.14	z1

p2 = p1; ➔

Memoria-helbidea	Memorian gordeta dauden datuak	Aldagaia
...		...
65000 65001	10	x
65002 65003	15	y
65004 65005	65002	p1
65006 65007	**65002**	p2
65008 65009	???	p3
65010	'H'	A
65011 65012	????	p4
65013 65014 65015 65016	3.14	z1

p3 = &A; ➔

Memoria-helbidea	Memorian gordeta dauden datuak	Aldagaia
...		...
65000 65001	10	x
65002 65003	15	y
65004 65005	65002	p1
65006 65007	65002	p2
65008 65009	**65010**	p3
65010	'H'	A
65011 65012	????	p4
65013 65014 65015 65016	3.14	z1

*p3 = ‘k’; ➔

Memoria-helbidea	Memorian gordeta dauden datuak	Aldagaia
...		...
65000	10	x
65001		
65002	15	y
65003		
65004	65002	p1
65005		
65006	65002	p2
65007		
65008	65010	p3
65009		
65010	**‘k’**	A
65011	????	p4
65012		
65013	3.14	z1
65014		
65015		
65016		

A = ‘w’; ➔

Memoria-helbidea	Memorian gordeta dauden datuak	Aldagaia
...		...
65000	10	x
65001		
65002	15	y
65003		
65004	65002	p1
65005		
65006	65002	p2
65007		
65008	65010	p3
65009		
65010	**‘w’**	A
65011	????	p4
65012		
65013	3.14	z1
65014		
65015		
65016		

p4 = &z1; ➔

Memoria-helbidea	Memorian gordeta dauden datuak	Aldagaia
...		...
65000	10	x
65001		
65002	15	y
65003		
65004	65002	p1
65005		
65006	65002	p2
65007		
65008	65010	p3
65009		
65010	‘w’	A
65011	**65013**	p4
65012		
65013	3.14	z1
65014		
65015		
65016		

*p4 = 2.81; ➔

Memoria-helbidea	Memorian gordeta dauden datuak	Aldagaia
...		...
65000	10	x
65001		
65002	15	y
65003		
65004	65002	p1
65005		
65006	65002	p2
65007		
65008	65010	p3
65009		
65010	'w'	A
65011	65013	p4
65012		
65013	**2.81**	z1
65014		
65015		
65016		

*p4 = *p4 * 10; ➔

Memoria-helbidea	Memorian gordeta dauden datuak	Aldagaia
...		...
65000	10	x
65001		
65002	15	y
65003		
65004	65002	p1
65005		
65006	65002	p2
65007		
65008	65010	p3
65009		
65010	'w'	A
65011	65013	p4
65012		
65013	**28.1**	z1
65014		
65015		
65016		

z1 = *p4 / 10.0; ➔

Memoria-helbidea	Memorian gordeta dauden datuak	Aldagaia
...		...
65000	10	x
65001		
65002	15	y
65003		
65004	65002	p1
65005		
65006	65002	p2
65007		
65008	65010	p3
65009		
65010	'w'	A
65011	65013	p4
65012		
65013	**2.81**	z1
65014		
65015		
65016		

Erakusle baten balioa pantailaratzeko, printf funtzioan %p erabiliko dugu: hamaseitar modura azalduko du.

Kontuz!!!! erakusleak erabiltzerakoan errore larriak gerta daitezke, erakusle batek Sistema Eragilearen memoria apunta dezakelako, bertan balio bat esleitzen bada ordenagailua izorra dezakegu. Erakusleak erabili baino lehen hasieratzea komeni da.

```
void main ()
{
    int A, *erak1;                    /* erak1 zenbaki oso bat erakusten du*/
    float R, *erak2 = &R;         /*erak2 R aldagi erreala erakusten du*/
    char *erak3 = &C, C='m';  /*erak3 gaizki hasieratuta. Konpilazio errorea: C erabilia erazagutu baino lehen */
    int *erak4 = &R;              /*erak4 gaizki hasieratuta. Konpilazio errorea: datu mota desberdinak */
    ...
}
```

ERAKUSLEAK ETA TAULAK:

Erakusle aldagaien balioak aldatu daitezke memoria korritzeko. Eragile aritmetikoak erabili daitezke (+ - ++ -- += -=) Erakusleen balioak aldatzea oso erabilgarria da taulen elementuak korritzeko.

```
void main ()
{
      int x, y[10], *erak1;                                    /* y[0] y[1] y[2]... y[9] memorian jarraian*/
      ...
      for (erak1= &y[0]; erak1 <&y[0]+9; erak1++)   /*erak1 erabiliz taula korritu*/
         {printf ("%p :  %5d", erak1,  *erak1);}
      ...
      for (erak1=y, x=0; erak1 < &y[10]; erak1++)          /*y, lehen elementuaren erakuslea da. (&y[0] = y)*/
         {if (*erak1 == 0)
                  {x++;}
            }
      printf ("%d zero ditu taulak", x);
      ...
}
```

Taula baten aldagaia berez erakusle konstantea da, lehen posizioa (zerogarrena) erakusten du. Erakusle aldagai bati zuzenean erakusle konstantea esleitu daiteke, baina soilik dimentsio bakarreko taula bada. Adi errore hauekin, erakusle konstanteak ezin dira aldatu:

```
void main ()
{
      int x, y[10], *erak1;
      ...
      erak1 = &y[0]++;                                    /*Gaizki. y[0]-ren helbidea konstantea da */
      erak1 = y++;                                        /*Gaizki. y, zerogarren elementuaren helbide
      konstantea da */
      erak1 = y + 1;                                      /*Ondo. */
      y += 3;                                             /*Gaizki. y helbide konstantea da*/
      erak1 += 3;                                         /*Ondo.  */
      ...
}
```

Erakusleek ere string edo karaktere kateak apunta ditzakete.

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

void main ()
{
      char es[100], *erak = es;
      int kontbokalak = 0, kontkonsonanteak=0;
      clrscr();
      puts ("Idatzi esaldi bat: ");
      gets (es);
      strlwr(es);

      for ( ; *erak!='\0'; erak++)
         {
         switch (*erak)
                  {
                  case 'a':
                  case 'e':
                  case 'i':
                  case 'o':
                  case 'u': {kontbokalak++;
                           break;}
                  default: {kontkonsonanteak++;}
                  }  //switch bukaera
         }  //for bukaera
      printf ("\n\t%s\n\t%d bokal eta %d konsonante ditu.",es, kontbokalak, kontkonsonanteak);

      puts ("\nSakatu tekla bat");
      getch();
}
```

ERAKUSLEEN TAULAK

Tauletan erakusleak ere gorde daitezke.

```
void main ()
{
int *erakTaula1 [10];                    /*zenbaki osoak apuntatzeko hamar elementuen taula*/
float *erakTaula2[5] = {1.1, 2.2, 3.3, 4.4, 5.5}; /*zenbaki errealak seinalatzeko bost elementuen taula. Hasieratuta*/
char *erakTaula3 [6]={'a','b','c','d','e', 'f'};  /*karaktereei apuntatzen duten erakusleen taulak*/
char *erakTaula4 [3]={"Lehen esaldia",  "Bigarrena", "Hirugarrena"};
...
}
```

*erakTaula2 [0]: 1.1 zenbakia apuntatzen du.
*erakTaula3 [5]: f karakterea apuntatzen du.
*erakTaula4 [0]: L karakterea apuntatzen du.
*erakTaula4 [1]: B karakterea apuntatzen du.
*erakTaula4 [2]: H karakterea apuntatzen du.

Helbidea	Edukia	Aldagaia
65000	**?**	erakTaula1 (erakTaula1[0])
65001		
65002	**?**	(erakTaula1[1])
65003		
65004	**?**	(erakTaula1[2])
65005		
65006	**?**	(erakTaula1[3])
65007		
65008	**?**	(erakTaula1[4])
65009		
65010	**?**	(erakTaula1[5])
65011		
65012	**?**	(erakTaula1[6])
65013		
65014	**?**	(erakTaula1[7])
65015		
65016	**?**	(erakTaula1[8])
65017		
65018	**?**	(erakTaula1[9])
65019		
65020	**65030**	erakTaula2 (erakTaula2[0])
65021		
65022	**65034**	(erakTaula2[1])
65023		
65024	**65038**	(erakTaula2[2])
65025		
65026	**65042**	(erakTaula2[3])
65027		
65028	**65046**	(erakTaula2[4])
65029		
65030		
65031	**1.1**	
65032		
65033		
65034		
65035	**2.2**	
65036		
65037		
65038		
65039	**3.3**	
65040		
65041		
65042		
65043	**4.4**	
65044		
65045		

65046		
65047	**5.5**	
65048		
65049		
65050	**65062**	ErakTaula3 (ErakTaula3[0])
65051		
65052	**65063**	(ErakTaula3[1])
65053		
65054	**65064**	(ErakTaula3[2])
65055		
65056	**65065**	(ErakTaula3[3])
65057		
65058	**65066**	(ErakTaula3[4])
65059		
65060	**65067**	(ErakTaula3[5])
65061		
65062	**'a'**	
65063	**'b'**	
65064	**'c'**	
65065	**'d'**	
65066	**'e'**	
65067	**'f'**	
65068	**65074**	ErakTaula4 (ErakTaula4[0])
65069		
65070	**65088**	(ErakTaula4[1])
65071		
65072	**65098**	(ErakTaula4[2])
65073		
65074	**'L'**	
65075	**'e'**	
65076	**'h'**	
65077	**'e'**	
65078	**'n'**	
65079	**' '**	
65080	**'e'**	
65081	**'s'**	
65082	**'a'**	
65083	**'l'**	
65084	**'d'**	
65085	**'i'**	
65086	**'a'**	
65087	**'\0'**	
65088	**'B'**	
65089	**'i'**	
65090	**'g'**	
65091	**'a'**	
65092	**'r'**	
65093	**'r'**	
65094	**'e'**	
65095	**'n'**	
65096	**'a'**	
65097	**'\0'**	
65098	**'H'**	
65099	**'i'**	
65100	**'r'**	
65101	**'u'**	
65102	**'g'**	
65103	**'a'**	
65104	**'r'**	
65105	**'r'**	
65106	**'e'**	
65107	**'n'**	
65108	**'a'**	
65109	**'\0'**	
65110		

ERAKUSLEI ERAKUSLEAK

Erakusleak apunta daitezke. Erakusle bat seinalatzen duen erakuslea sortu daiteke, alegia.

```
void main ()
{
      float A, *erak1, **erak2, ***erak3;
      ...
      erak1 =  &A;
      erak2 =  &erak1;
      erak3 =  &erak2;
      ...
      A = 5;                    /* A aldagaiak 5 balio du*/
      *erak1 = 6;               /* A aldagaiak 6 balio du*/
      **erak2 = 7;              /* A aldagaiak 7 balio du*/
      ***erak3 = 8;             /* A aldagaiak 8 balio du*/
      ...
}
```

Erazagupena ondoren: **float A, *erak1, **erak2, ***erak3;**

Helbidea	**Edukia**	**Aldagaia**
32000	**?**	A
32001		
32002		
32003		
32004	**?**	erak1
32005		
32006	**?**	erak2
32007		
32008	**?**	erak3
32009		

Lehen agindu ondoren: **erak1 = &A;**

Helbidea	**Edukia**	**Aldagaia**
32000	**?**	A
32001		
32002		
32003		
32004	**32000**	erak1
32005		
32006	**?**	erak2
32007		
32008	**?**	erak3
32009		

Bigarren agindu ondoren: **erak2 = &erak1;**

Helbidea	**Edukia**	**Aldagaia**
32000	**?**	A
32001		
32002		
32003		
32004	**32000**	erak1
32005		
32006	**32004**	erak2
32007		
32008	**?**	erak3
32009		

Hirugarren agindu ondoren: **erak3 = &erak2;**

Helbidea	**Edukia**	**Aldagaia**
32000	**?**	A
32001		
32002		
32003		
32004	**32000**	erak1
32005		
32006	**32004**	erak2
32007		
32008	**32006**	erak3
32009		

Laugarren agindu ondoren: **A = 5;**

Helbidea	**Edukia**	**Aldagaia**
32000	**5.0**	A
32001		
32002		
32003		
32004	**32000**	erak1
32005		
32006	**32004**	erak2
32007		
32008	**32006**	erak3
32009		

Bostgarren agindu ondoren: ***erak1 = 6;**

Helbidea	**Edukia**	**Aldagaia**
32000	**6.0**	A
32001		
32002		
32003		
32004	**32000**	erak1
32005		
32006	**32004**	erak2
32007		
32008	**32006**	erak3
32009		

Seigarren agindu ondoren: ****erak2 = 7;**

Helbidea	**Edukia**	**Aldagaia**
32000	**7.0**	A
32001		
32002		
32003		
32004	**32000**	erak1
32005		
32006	**32004**	erak2
32007		
32008	**32006**	erak3
32009		

Zazpigarren agindu ondoren: *****erak3 = 8;**

Helbidea	**Edukia**	**Aldagaia**
32000	**8.0**	A
32001		
32002		
32003		
32004	**32000**	erak1
32005		
32006	**32004**	erak2
32007		
32008	**32006**	erak3
32009		

OHIKO ERROREAK

```
void main ()
{
	int a=3, *erak1, b=2;
	float *erak2, c= 5.5, *erak3;
	char d='H', *erak4;
	int e[10];
	...
	a= (b+2)/(*erak * a);		/*erak1 hasieratu gabe*/
	*erak1 = a * b;			/*erak1 hasieratu gabe*/
	erak2 = c;			/* & falta da*/
	erak2 = &d;			/* datu mota desberdinak*/
	erak3 = &erak2 ;		/*erakusle bikoitza, erakuslea  apuntatzen duen erakuslea. Datu mota
	errorea.*/
	erak3 = *erak4;			/* ez da helbidea*/
	e++;				/* helbide konstantea*/
	e -=2;				/* helbide konstantea*/
	erak1 = erak2 + erak3;		/* erakusleen batura. Konpiladoreak ez ditu baimentzen horrelako
	eragiketak*/
	erak1 = erak1 + &e[0];		/* erakusleen batura. Konpiladoreak ez ditu baimentzen horrelako
	eragiketak*/
	...
}
```

FUNTZIOAK edo AZPIPROGRAMAK

Goi-mailako gainontzeko programazio-lengoaietan bezala, C-n funtzioak (azpiprogramak) idazteko aukera daukagu.

Programa konplexuak egin behar direnean, normalean *divide and conquer* edo *zatitu eta irabazi* teknika erabiltzen da. Arazo konplexua arazo xinpleagoetan banatzen da eta bakoitzari funtzio bat esleituz hasierako arazo orokorrari erantzuna ematen zaio.

Programak honela egitearen abantailak:

- Programak errazagoak dira irakurtzeko eta zuzentzeko.
- Eginkizun desberdinentzat aldagaien balioak ez nahastea ahalbidetzen du
- Programa baten barruan zeregin berdina duten sententzia multzoak ez errepikatzea ahalbidetzen du
- Programa baten exekuzio eta jarraipena egitea errazten du (debugging), programa zatika aztertu dezakegulako.

Bi funtzio-mota desberdin dauzkagu: balio bat edo batzuk itzultzen dutenak eta baliorik itzultzen ez dutenak.

Ikusi dezagun adibide baten bitartez zein den funtzioen ideia. Suposatu, pantailan hurrengoa azaltzea nahi dugula:

```
****************************************
Gure lehenengo programa funtzioak erabiliz
****************************************
```

Orain arte ikusitakoarekin honelako kodea sortuko genuke:

```
#include <stdio.h>
void main(){
 int kont;
for (kont=1;kont<=40;kont++)
        printf("*");
printf ("\n");
printf ("Gure lehenengo programa funtzioak erabiliz \n");
for (kont=1;kont<=40;kont++)
        printf("*");
printf ("\n");
}
```

Ikusten denez, kode zati bat birritan idazten da; gainera, bi kasuetan kode zatiak helburu berdina du. Honelakoak ez errepikatzeko funtzioak erabiliko ditugu.

Programa honela geldituko litzateke:

```
#include <stdio.h>
/*prototipo deklarazioa*/
void asteriskoak();

void main(){
asteriskoak();
printf ("Gure lehenengo programa funtzioak erabiliz \n");
asteriskoak();
}
```

Idatzitako programan *asteriskoak* izeneko funtzioa deitzen dugu asteriskoak idatz ditzan. Funtzioa deitzen den bakoitzean, bere definizioan sartutako kodea exekutatuko da. Kasu honetan, jarraian azaltzen dena.

```
#include <stdio.h>
void asteriskoak(){
int kont;
for (kont=1;kont<=40;kont++)
        printf("*");
        printf ("\n");
}
```

Funtzioen egitura

main funtzio soil bat erabili beharrean, funtzioak erabiltzea erabakitzen badugu, hurrengo egitura izango du gure programak:

Liburutegiak
konstanteak
Funtzioen prototipoa
main funtzioa
Funtzioen definizioa

Ikusten den moduan hiru zati nagusi dauzkagu. Alde batetik prototipoen deklarazioa, bestetik programa nagusia (funtzio deiekin) eta azkenik funtzioak (funtzioei dagokien kodea). Funtzio deiak funtzioen barruan ere azaldu daitezke.

Funtzioen prototipoa

Funtzioen prototipoak main funtzioa baino lehen jartzen dira konpiladoreak jakiteko funtzioak itzuliko duen balio mota, izena eta behar dituen parametroak; honako egitura jarraitzen dute funtzioen prototipoek:

datu_mota izena ([datu_mota parametro_izena]);*

Lehenengo datu_motak adierazten du funtzioak itzuliko duen balioaren mota, izenak funtzioren izena definitzen du eta parentesi barruan, gure funtzioak erabiliko dituen parametroen motak azaltzen dira.

Aurreko kasuan ondokoa genuen: *void asteriskoak().*Hasierako void-k adierazten du funtzioak ez duela ezer itzultzen, asteriskoak da bere izena eta ez du parametrorik behar.

main funtzioa

main funtzioa C programa bat hastean exekutatzen den lehenengo funtzioa da eta funtzio hau bukatzerakoan programaren exekuzioa amaituko da.

main funtzioak beste funtzioak definitzeko erabilitako arau berdinak erabili behar dira.

Funtzioen deia

Funtzio bat egikaritu nahi denean funtzioari dei bat egiten zaio. Hau egiterakoan, aipaturiko datuekin (parametro efektiboekin) funtzioaren definizioan definitutako kodea egikaritzen da eta emaitza itzuliko da

return sententzia

Funtzioen gorputzean funtzioaren helburuak lortzeko beharrezkoak diren aginduak sartu behar dira. Haien artean, badago bat berezia dena: return sententzia. Bi helburu dauzka sententzia honek, alde batetik funtzioaren amaiera behartzea (programaren egikaritzapena funtzio-deia egindako zati kodera itzuliko da), eta beste aldetik funtzioak itzuli beharreko balioa zehaztea.

Lehenengo helbururako sententzia hau ez da derrigorrezkoa konpiladoreak funtzioaren azken agindura iristerakoan funtzioa amaitzen duelako.

Bai da derrigorrezkoa ordea, gure funtzioak zero ez den balio bat itzultzea nahi badugu.

return sententzia baten bitartez gure funtzioak emaitza bakarra eman dezake, ez dugu ikusiko balio gehiago itzultzeko egin beharrekoa.

Adibidez, kosinu funtzioak parametro bezala 0 balioa hartzen badu 1 itzuli beharko du, hau da, return (1) jarri beharko genuke gure programan.

return sententzia programaren edozein zatitan egon daiteke eta behin baino gehiagotan azaldu daiteke.

```
#include <stdio.h>
int  azalera(int );
void main()
{
        int i=10;
        printf ("%d da %d aldea duen karratuaren azalera ", azalera (i),i);
}
int azalera (int j)
{
        j=j*j;
        return (j);
}
```

Aldagai lokalak vs kanpokoak

Aurreko adibideko asteriskoak funtzioaren barruan kont aldagai bat erazagutu dugu. main funtzioan kont=kont+2; jarriko bagenu errorea emango liguke, funtzio baten barruan definitzen diren aldagaiak soilik funtzio horretan baitira ezagunak. Honelako aldagaiak lokalak deitzen dira.

Batzutan, funtzio guztientzat ezagunak diren aldagaiak definitzea komenigarria da, honelako aldagaiak kanpoko aldagaiak deitzen dira. Hauek main edo funtzio guztien aurretik definitu behar dira. Honelakoak erabiltzea programen ulergarritasuna murrizten du.

Funtzioen argumentuak

Funtzioen definizioan azaltzen diren parametro formalak, aldagai lokalak bezalakoak dira, funtzioan sartzerakoan sortu eta ateratzerakoan desagertu egiten direnak. Argumentu formalak eta funtzio-deian erabilitakoak mota berdinekoak izan behar dute. C lengoaian parametro guztiak balio bidez pasatzen dira.

- Balio bidez: funtzioaren parametro formalean argumentuari dagokion balioa kopiatzen da, beraz, funtziotik ateratzerakoan egindako aldaketak ez dute eraginik izango programa deitzaileko aldagaietan.

Adibidez:

```
#include <stdio.h>
int  azalera(int );

void main(){
int i=10;
printf ("%d da %d aldea duen karratuaren azalera ", azalera (i),i);
}

int azalera (int j) {
  j=j*j;
 return (j);}
```

Adibide honetan, j-k 10 balioa hartuko du hasieran eta azalera funtzioan azalera kalkulatzerakoan, aldagaiak jasango dituena j aldagai lokala izango da, programa nagusiko i aldagaiak 10 balioa mantenduko duelarik. Programa honek irteera moduan ondoko mezua idatziko du pantailan: "*100 da 10 aldea duen karratuaren azalera*".

Adibideak

1. Funtzio bat egin nahi dugu zeini karaktere bat eta osoko balio bat emanda sarrera moduan, karaktere horrekin adierazitako luzerako marra bat egingo duen.

```
#include <stdio.h>
void marra(char, int );

void main(){
int luzera=1;
char karak;
printf ("Sartu margotzeko erabili beharreko karakterea \n ");
scanf ("%c", &karak);
printf ("Sartu marraren luzera \n ");
scanf ("%d", &luzera);
marra (karak, luzera)
}

void marra (char nire_karak, int nire_luzera) {
int j;
for (j=1;j<=nire_luzera; j++)
  printf ("%c", nire_karak);

printf ("\n");
}
```

2. $1+2^2+3^3+4^4+\ldots..+n^n$ segidaren balioa kalkulatzen duen programa idatz ezazu.

```
#include <stdio.h>
#include <conio.h>
//prototipoaren deklarazioa
long berredura (int);

void main(){
int muga, kontagailua;
long emaitza=0;
clrscr();
printf ("Sartu mugaren balioa  \n ");
scanf ("%d", &muga);
for (kontagailua=1;kontagailua<=muga;kontagailua++)
  {emaitza=emaitza+berredura (kontagailua);
  printf("\n emaitza hau da: %d", emaitza);}

long berredura(int zbkia){
int kont;
long balioa;
for (kont=1,balioa=1;kont<=zbkia; kont++)
balioa=balioa*zbkia;
return (balioa);
}
```