Fonaments d'Informàtica

Sessions de Teoria

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Contingut

1. Introducció a la programació

Motivació

- La programació és una disciplina fonamentada en:
  - Teoria
  - Metodologia
  - Conjunt de tècniques

  que contribueixen a que sigui una tasca:
  - Eficaç
  - Eficient
- Paradigmes de programació: imperativa, orientada a objectes, genèrica, ...
- Llenguatges d'alt nivell: C, Pascal, C++, Java, ...

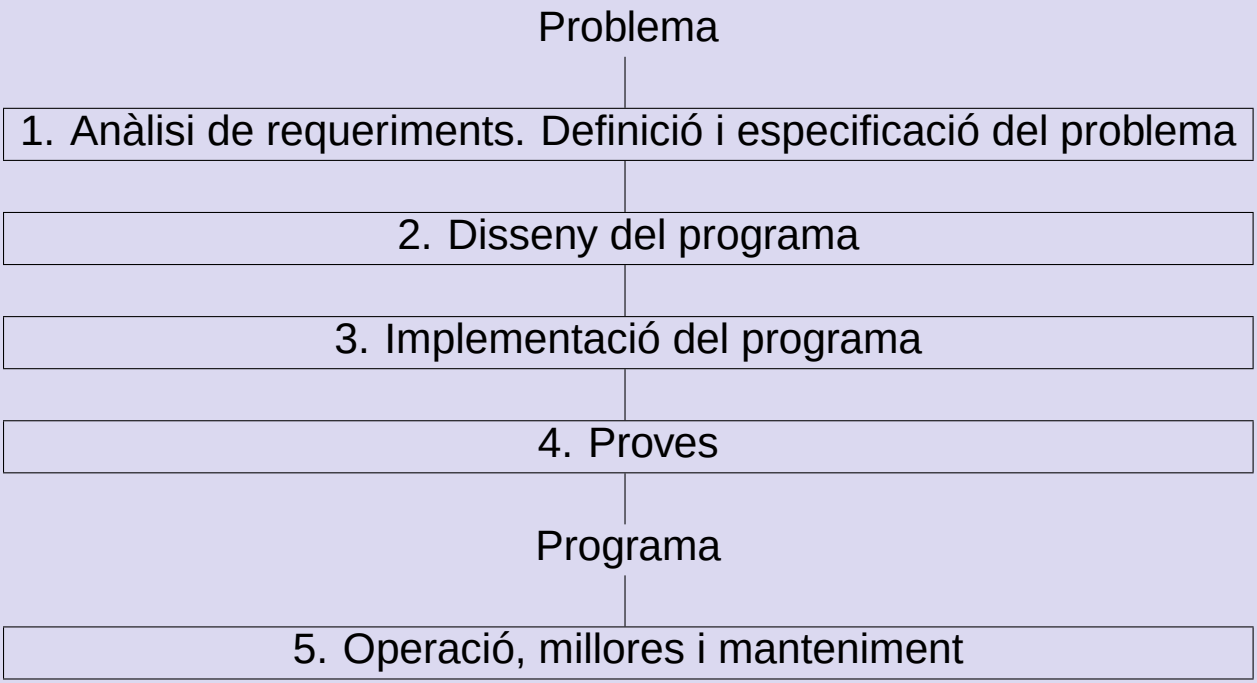
Desenvolupament d'un programa

Les etapes per a desenvolupar un programa són bàsicament:

- Entendre/especificar el problema
- Plantejar/planificar la solució
- Formular la solució
- Avaluar la correcció de la solució proposada

La formulació de la solució es basa en el concepte d'algorisme.

Desenvolupament d'un programa



Definició de conceptes

Algorisme: descripció no ambigua i precisa d'accions que cal realitzar per a resoldre un problema ben definit en un temps finit.

Problema ben definit: saber quin és l'estat inicial, Precondició, i quin és l'estat final, Postcondició.

Processador: entitat capaç de comprendre i executar un algorisme.

Entorn: conjunt d'objectes necessaris per dur a terme una tasca determinada.

Un algorisme parteix d'un estat inicial del entorn i el modifica fins arribar a un estat final que es correspond amb la solució del problema.

Definició de conceptes

Acció: esdeveniment finit en el temps que pot modificar i/o observar l'entorn.

Acció elemental: acció que el processador de l'algorisme és capaç d'entendre.

Programa: quan el processador es un ordenador, l'algorisme expressat amb un conjunt d'accions que entén l'ordenador s'anomena programa.

Llenguatge de programació: s'ha de conèixer la sintaxis i la semàntica dels seus elements.

Ens interessen les tècniques de programació, no els detalls del llenguatge!

Objectius

L'objectiu bàsic del curs és dissenyar i implementar programes de forma:

- Correcta: qualitat fonamental i imprescindible
- Intel·ligible: clar i ben estructurat.
- Eficient: *"ràpid"* i fa un *"bon ús dels recursos"* (memòria).
- General: de fàcil ús, manteniment, etc.

2. Conceptes bàsics de Programació Estructurada en C++

Estructura d'un programa

```
// Declaració de llibreries
#include <iostream>
using namespace std;

// Programa principal
int main() {
    // Declaració d'objectes

    // Accions/Sentències
    cout << "Hola mon!"<< endl;
    return 0;
}
```

Objectes

L'entorn d'un programa està format per objectes l'estat dels quals es pot observar i/o modificar. Un objecte té tres atributs:

- Nom o identificador
- Tipus
- Valor

Identificadors:

- És una seqüència de caràcters (lletres, dígits i subratllats) i serveix per identificar de forma unívoca l'objecte.
  - El primer caràcter ha de ser una lletra o el subratllat
  - Las lletres minúscules i majúscules són diferents
  - Poden tenir qualsevol longitud (en alguns compiladors està restringida)
  - No poden haver-hi espais en blanc
  - No es poden utilitzar les paraules reservades - utilitzades per el propi llenguatge- com per exemple `const`, `int`, `double`, `while`, etc.
- Exemples `voltatge`, `_max`, `dies_setmana`

Concepte de tipus:

- Conjunt finit de valors
- Operacions aplicables als elements del conjunt. Les operacions poden ser:
  - Externes
  - Internes

i al mateix temps:

- Totals
- Parcials

Valors:

- L'objecte només pot tenir un dels valors especificats per el seu tipus. Depenent de si el seu valor es pot modificar o no un objecte pot ser:
  - Constant
  - Variable

Classificació dels tipus

- Tipus elementals o predefinits: els proporcionats pel llenguatge de programació.
- Tipus definits per l'usuari. C++ ofereix mecanismes per definir tipus nous a partir dels tipus existents (predefinits o definits prèviament).

Tipus elementals de C++

C++ ofereix tipus predefinits per els valors lògics (booleans), els caràcters, els números enters i els números reals.

Tipus lògic: **bool**

Rang de valors:	**false**, **true**
Operadors interns:	`!`, `&&`, `||`, `==`, `!=`, `<`, `<=`, `>`, `>=`
Operadors externs:	
Sintaxi de valors:	**false**, **true**

Taula de veritat dels operadors booleans:

`a`	`b`	`!a`	`a && b`	`a || b`
true	**true**	**false**	**true**	**true**
true	**false**	**false**	**false**	**true**
false	**true**	**true**	**false**	**true**
false	**false**	**true**	**false**	**false**

Lleis de *Morgan*

```
!(a && b) = (!a) || (!b)
!(a || b) = (!a) && (!b)
```

Tipus caràcter: **char**

Rang de valors:	Conjunt finit de caràcters alfanumèrics
Operadors interns:	
Operadors externs:	==, !=, <, <=, >, >=
Sintaxi de valors:	Es representen posant el caràcter alfanumèric entre cometes simples. Exemples: `'a' 'W' '1'`

Tipus enter: **int**

Rang de valors:	Conjunt de valors enters compresos entre un valor mínim `INT_MIN` i un valor màxim `INT_MAX`
Operadors interns:	- (canvi de signe), +, -, *, /, %
Operadors externs:	==, !=, <, <=, >, >=
Sintaxi de valors:	Es representen tal i com ho fem normalment. Exemples: 3, 5654, -8999

Notes:

- `INT_MIN` ha de ser -32767 o més petit, `INT_MAX` ha de ser com a mínim 32767
- El tipus elemental **int** té les següents variacions (modificadors): **short**, **long**. Els tres es poden combinar amb **unsigned**.

- Utilitzant l'operador `sizeof(tipus)` i la llibreria `<limits>` podem saber els bytes de memòria que ocupa el tipus per a una implementació / compilador concret.

```
Integer types, DevC++ Version 4.9
Size of short int types is 2 bytes.
Signed short min:  -32768 max:  32767
Unsigned short min:  0 max:  65535
Size of int types is 4 bytes.
Signed int min:  -2147483648 max:2147483647
Unsigned int min:  0 max:  4294967295
Size of long int types is 4 bytes.
Signed long min:  -2147483648 max:
2147483647
Unsigned long min:  0 max:  4294967295
```

Tipus real: **float**, **double**, **long double**

Rang de valors:	Conjunt finit de valors reals
Operadors interns:	- (canvi de signe), +, -, *, /
Operadors externs:	==, !=, <, <=, >, >=
Sintaxi de valors:	Es representen sempre amb el punt decimal i opcionalment amb notació exponencial. Exemples: `0.6`, `5.0E-8`, `42.22E+0.8`, `1.0E5`, `5.0`

- **float**, també anomenat de *precisió simple*, $\pm 3.4 \times 10^{-38}$ a $\pm 1.7 \times 10^{38}$ (amb 7-dígits de precisió)
- **double**, també anomenat de *precisió doble*, $\pm 1.7 \times 10^{-308}$ a $\pm 3.4 \times 10^{308}$ (amb 15-dígits de precisió)
- **long double**, també anomenat de *precisió estesa*, $\pm 3.4 \times 10^{-4932}$ a $\pm 1.7 \times 10^{4932}$ (amb 18-dígits de precisió).

Tipus definits per l'usuari: Tipus enumerats

enum <*nom*> { <*valor*$_1$>, <*valor*$_2$>, . . . , <*valor*$_n$> };

Exemples

```
enum tColor {verd, blau,  vermell, groc,
             magenta, cyan,  negre, blanc};
enum tDia {dilluns, dimarts,  dimecres,
           dijous, divendres,
           dissabte, diumenge};
enum tMes {gener, febrer,  marc, abril,
           maig, juny,  juliol, agost,
           setembre,octubre,
           novembre, desembre};
```

Declaració d'objectes: Declaració de constants

const <*tipus*> <*nom*> = <*expressió*> {, <*nom*> = <*expressió*>};

Exemples

```
const double PI = 3.1415927, ARREL_2 = 1.4142136;
const double DOS_PI = PI * 2.0;
const double LN_2 = 0.69314718056;
const double PTS_EURO = 166.386;
const int DIES_SETMANA = 7;
const char SEPARADOR = '_';
const tMes MES_VACANCES = agost;
```

Declaració d'objectes: Declaració de variables

<tipus> <nom> {= *<expressió>*} {, *<nom>* {= *<expressió>*}};

Exemples

```
int saldo = 0, Preu, nombreArticles;
double radiCercle = 15.6;
double longitudCircumferencia = radiCercle * DOS_PI;
char caracterLlegir;
bool b1, b2 = false; int cont;
tDia dia;
```

Expressions

Una expressió és qualsevol combinació correcta d'operands i operadors.

Sintaxi

- Un valor és una expressió
- Una variable és una expressió
- Una constant és una expressió
- Si E és una expressió (E) també ho és
- Si E és una expressió i $\circ$ és un operador unari $\circ E$ també ho és
- Si E_1 i E_2 són expressions i $\circ$ és un operador binari $E_1 \circ E_2$ també ho és
- Si E_1, E_2,...,E_n són expressions i f és una funció d'aritat n $f(E_1, E_2, \ldots, E_n)$ també ho és

Exemples

- `saldo`
- `longitudCircumferencia`
- `caracterLlegir`
- `b1`
- `PI`
- `PTS_EURO`
- `SEPARADOR`
- `(PI)`
- `!b1`
- `-saldo`
- `2.0 * PI`
- `b1 && b2`
- `f(3,PI)`

Avaluació d'expressions

El resultat d'avaluar una expressió és un valor i per tant té un únic tipus.
El resultat d'avaluar una expressió E és:

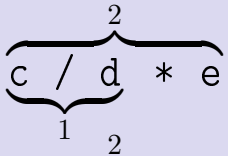
- Si E és un valor el resultat és directament el valor
- Si E és una variable el resultat és el valor que conté la variable en el moment de l'avaluació
- Si E és una constant el resultat és el seu valor (fixat a la seva definició)
- Si E és (E_1) el resultat és el resultat d'avaluar E_1
- Si E és $\circ E_1$ el resultat és el resultat d'aplicar l'operador unari $\circ$ al resultat d'avaluar E_1
- Si E és $E_1 \circ E_2$ el resultat és el resultat d'aplicar l'operador $\circ$ als resultats d'avaluar E_1 i E_2
- Si E és $f(E_1, E_2, \ldots, E_n)$ el resultat és el resultat d'aplicar f als resultats d'avaluar E_1, $E_2 \ldots E_n$

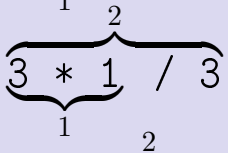
L'ordre d'avaluació d'una expressió es fa segons les següents prioritats:

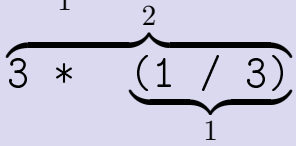
1. Les expressions entre parèntesis avaluant primer els parèntesis més interns
2. Les funcions
3. - (canvi de signe), !
4. *, /, %
5. +, -
6. ==, !=, <, <=, >, >=
7. &&
8. ||

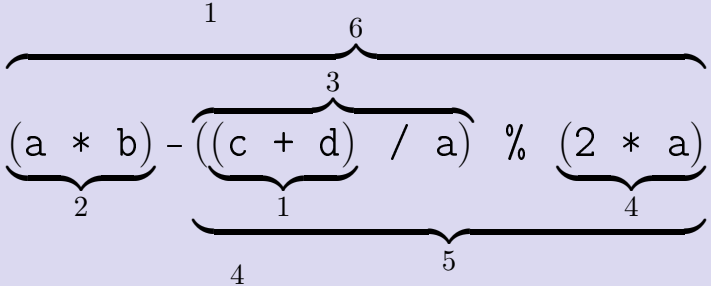
Els parèntesis serveixen per a modificar les prioritats.
A igualtat de prioritats s'avalua d'esquerra a dreta.

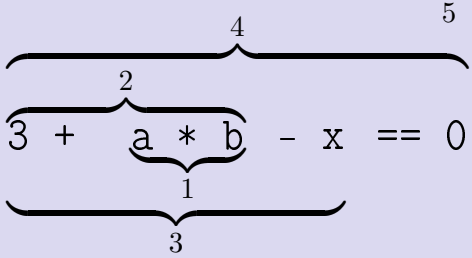
Exemples d'avaluació:

- 

- 

el resultat serà 1
- 

el resultat serà 0
- 

- 

Exemple: *Programa longitud de la circumferència*

```
// Declaració de constants
const double PI = 3.1415927;
const double DOS_PI = PI * 2.0;

int main(void) {
  // Declaració d'una variable per guardar el radi
  double radi;
  // Demanem el valor del radi a l'usuari
  cout << "Introdueix el valor del radi:  ";
  // El llegim i el guardem a la variable radi
  cin >> radi;
  // Declarem una variable per a la longitud de
  // la circumferència amb el valor corresponent
  double longitudCircumferencia = radi * DOS_PI;
  // Escrivim la longitud de la circumferència
  cout << "La longitud de la circumferència és:  ";
  cout << longitudCircumferencia;
  return 0;
}
```

Accions Elementals

Las instruccions (o *sentències*) bàsiques per desenvolupar un programa en C++ són:

- La instrucció d'assignació
- Els operadors i les funcions de conversió
- Les instruccions d'entrada i sortida

L'assignació

La instrucció d'assignació permet assignar un valor a una variable mitjançant l'operador d'assignació.

Sintaxi

<nom> = *<expressió>* ;

Semàntica

La variable *nom* que es troba a l'esquerra de l'operador d'assignació pren per valor el resultat d'avaluar l'*expressió* que es troba a la dreta de l'operador.

Exemples

```
double d;
int i;
char c;
bool b;
```

Exemples

```
double d;
int i;
char c;
bool b;

d = (3.4 * 2.0 - 0.8) / 5.0;
```

Exemples

```
double d;
int i;
char c;
bool b;

d = (3.4 * 2.0 - 0.8) / 5.0;
// d=1.2
```

Exemples

```
double d;
int i;
char c;
bool b;

d = (3.4 * 2.0 - 0.8) / 5.0;
// d=1.2
i = 56;
```

Exemples

```
double d;
int i;
char c;
bool b;

d = (3.4 * 2.0 - 0.8) / 5.0;
// d=1.2
i = 56;
// d=1.2   i=56
```

Exemples

```
double d;
int i;
char c;
bool b;

d = (3.4 * 2.0 - 0.8) / 5.0;
// d=1.2
i = 56;
// d=1.2   i=56
i = i % 5;
```

Exemples

```
double d;
int i;
char c;
bool b;

d = (3.4 * 2.0 - 0.8) / 5.0;
// d=1.2
i = 56;
// d=1.2   i=56
i = i % 5;
// d=1.2   i=1
```

Exemples

```
double d;
int i;
char c;
bool b;

d = (3.4 * 2.0 - 0.8) / 5.0;
// d=1.2
i = 56;
// d=1.2   i=56
i = i % 5;
// d=1.2   i=1
c = 'a';
```

Exemples

```
double d;
int i;
char c;
bool b;

d = (3.4 * 2.0 - 0.8) / 5.0;
// d=1.2
i = 56;
// d=1.2   i=56
i = i % 5;
// d=1.2   i=1
c = 'a';
// d=1.2   i=1   c='a'
```

Exemples

```
double d;
int i;
char c;
bool b;

d = (3.4 * 2.0 - 0.8) / 5.0;
// d=1.2
i = 56;
// d=1.2  i=56
i = i % 5;
// d=1.2  i=1
c = 'a';
// d=1.2  i=1  c='a'
b = (c == 'b'||i == 1) && d < 3.0;
```

Exemples

```
double d;
int i;
char c;
bool b;

d = (3.4 * 2.0 - 0.8) / 5.0;
// d=1.2
i = 56;
// d=1.2  i=56
i = i % 5;
// d=1.2  i=1
c = 'a';
// d=1.2  i=1  c='a'
b = (c == 'b'||i == 1) && d < 3.0;
// d=1.2  i=1  c='a'  b=true
```

Exemple: *Intercanvi dels valors de dues variables*

```
int x,y;

// Accions/Senténcies
x = 3;
// x=3
y = 5;
// x=3   y=5
```

Exemple: *Intercanvi dels valors de dues variables*

```
int x,y;

// Accions/Senténcies
x = 3;
// x=3
y = 5;
// x=3   y=5


x = y;
```

Exemple: *Intercanvi dels valors de dues variables*

```
int x,y;

// Accions/Senténcies
x = 3;
// x=3
y = 5;
// x=3   y=5


x = y;
// x=5   y=5
```

Exemple: *Intercanvi dels valors de dues variables*

```
int x,y;

// Accions/Senténcies
x = 3;
// x=3
y = 5;
// x=3   y=5


x = y;
// x=5   y=5
y = x;
```

Exemple: *Intercanvi dels valors de dues variables*

```
int x,y;

// Accions/Senténcies
x = 3;
// x=3
y = 5;
// x=3   y=5


x = y;
// x=5   y=5
y = x;
// x=5   y=5
```

Exemple: *Intercanvi dels valors de dues variables*

```
int x,y;

// Accions/Senténcies
x = 3;
// x=3
y = 5;
// x=3   y=5
```

Exemple: *Intercanvi dels valors de dues variables*

```
int x,y;
int aux;
// Accions/Senténcies
x = 3;
// x=3
y = 5;
// x=3   y=5
```

Exemple: *Intercanvi dels valors de dues variables*

```
int x,y;
int aux;
// Accions/Senténcies
x = 3;
// x=3
y = 5;
// x=3   y=5
aux = x;
```

Exemple: *Intercanvi dels valors de dues variables*

```
int x,y;
int aux;
// Accions/Senténcies
x = 3;
// x=3
y = 5;
// x=3   y=5
aux = x;
// x=3   y=5   aux=3
```

Exemple: *Intercanvi dels valors de dues variables*

```
int x,y;
int aux;
// Accions/Senténcies
x = 3;
// x=3
y = 5;
// x=3   y=5
aux = x;
// x=3   y=5   aux=3
x = y;
```

Exemple: *Intercanvi dels valors de dues variables*

```
int x,y;
int aux;
// Accions/Senténcies
x = 3;
// x=3
y = 5;
// x=3   y=5
aux = x;
// x=3   y=5   aux=3
x = y;
// x=5   y=5   aux=3
```

Exemple: *Intercanvi dels valors de dues variables*

```
int x,y;
int aux;
// Accions/Senténcies
x = 3;
// x=3
y = 5;
// x=3   y=5
aux = x;
// x=3   y=5   aux=3
x = y;
// x=5   y=5   aux=3
y = aux;
```

Exemple: *Intercanvi dels valors de dues variables*

```
int x,y;
int aux;
// Accions/Senténcies
x = 3;
// x=3
y = 5;
// x=3   y=5
aux = x;
// x=3   y=5   aux=3
x = y;
// x=5   y=5   aux=3
y = aux;
// x=5   y=3   aux=3
```

Notes

- L'identificador de l'esquerra de l'operador d'assignació ha de ser sempre el nom d'una variable.
- El tipus de la variable i el tipus de l'expressió han de ser el mateix (excepte en el cas de conversions implícites).
- Primer s'avalua tota l'expressió i després s'assigna el valor resultant de l'avaluació a la variable.
- La primera assignació d'un valor a una variable rep el nom d'*inicialització* (es pot fer al declarar la variable).
- Si la variable contenia algun valor abans de l'assignació el perd i emmagatzema el nou valor.
- El valor d'una variable es manté fins que se li assigna un de nou.

Conversions de tipus (*casting*)

A vegades és necessari fer conversió de tipus, és a dir convertir el resultat d'avaluar una expressió en un altre tipus (per exemple convertir un resultat *real* en *enter* i vice-versa i convertir un *caràcter* en *enter* i vice-versa).

Les instruccions de conversió de tipus permeten donada una expressió d'un tipus calcular un valor d'un altre tipus.

La conversió del valor d'un objecte no afecte l'estat del mateix!

La conversió de tipus es pot fer de dues maneres:

- *Implícita*: la realitza automàticament el compilador. Es important conèixer el seu funcionament!
- *Explícita*: el programador la fa explícitament utilitzant les funcions de conversió.

Conversions implícites

Les conversions implícites consisteixen en que els operands de tipus més baix es converteixen a valors de tipus més alts: *promoció*.

- Son conversions basades en el **rang**: valors de rang inferior es converteixen en valors de tipus de major rang
- El compilador convertirà *tots* els valors dels objectes a un únic tipus compatible amb la/es operació/ons a realitzar
- Les regles són las següents:
  - Si l'operand es de tipus `char`, `short` o `enumerat` es converteix a `int`
  - Regla de la promoció integral: en cas d'operands de diferents tipus la conversió es fa segons la següent llista: `int`, `long`, `float`, `double`, `long double`

Conversions explicites

El programador indica explícitament de quin tipus vol que sigui el resultat de convertir l'expressió. El llenguatge C++ ofereix dos mecanismes bàsics per realitzar la conversió explicita:

- El tradicional operador de mutació del llenguatge C, amb la següent sintaxi:

  (*<tipusDesti>*)(*<expressió>*)

- Les funcions de conversió, amb la següent sintaxi:

  <tipusDesti>(*<expressió>*)

Els dos mecanismes tenen el mateix comportament: donat el valor resultant d'avaluar l'*expressió* calcular un nou valor de tipus *tipusDesti*

Exemples

- Utilitzant l'operador de mutació de C

```
int nbResist = 60;      // Número total de resistències
int categories = 7;     // Número total de categories
double mitjana=(double)(nbResist)/(double)(categories);
```

- Utilitzant les funcions de conversió de C++

```
int nbResist = 60;      // Número total de resistències
int categories = 7;     // Número total de categories
double mitjana=double(nbResist)/(double)(categories);
```

Comportament de les funcions de conversió

int(`expr`) i `expr` avalua a un tipus real	Retorna el valor de tipus **int** equivalent a truncar el valor real al que avalua `expr` (resultat de "treure-li" els decimals)
double(`expr`) i `expr` avalua a un tipus enter	Retorna el valor real equivalent al resultat d'avaluar `expr`
char(`expr`) i `expr` avalua a un tipus enter	Retorna el caràcter que es codifica amb el valor enter resultant d'avaluar `expr`
int(`c`) i c és un caràcter	Retorna l'enter amb el que s'ha codificat el caràcter `c`

Entrada i sortida d'informació

- Normalment, els càlculs que realitza un programa requereixen l'*entrada* de dades en base a las que s'obtenen resultats que constitueixen la *sortida* del programa.
- Les operacions d'entrada permeten la *lectura* de dades de dispositius d'entrada (teclat, unitats de disc etc.)
- Les operacions de sortida permeten la *escriptura* de dades en un dispositius de sortida (pantalla, unitats de disc, etc.)

Entrada i sortida en C++

- El sistema d'entrada/sortida de C++ opera sobre fluxos (*streams*, en angles).
- Un flux és un dispositiu lògic que produeix o consumeix informació (caràcters).
- Els fluxos es vinculen a un dispositiu físic. Tots els fluxos operen de forma similar, a pesar de que estiguin connectats a dispositius físics molt diferents (la pantalla, una impressora o un fitxer de text).

Fluxos cin i cout

C++ ofereix dos fluxos predefinits (creats per defecte) els fluxos *cin* y *cout*:

- *cin* correspond a l'entrada estàndard (per exemple, el teclat) i ens permet la introducció de dades. El canal d'entrada *cin* té associat l'operador de entrada $>>$ per a llegir valors del canal de entrada.
- *cout* correspond a la sortida estàndard (per exemple, la pantalla) i ens permet visualitzar dades o escriure-les a un fitxer. El canal de sortida *cout* té associat l'operador de sortida $<<$ per a escriure valors al canal de sortida.

Nota: S'ha d'incloure la biblioteca (llibreria) *iostream* mitjançant la directiva `#include<iostream>`.

L'operador de sortida <<

Està predefinit per als tipus bàsics i per les cadenes de caràcters. Es pot sobrecargar (redefinir). Veiem la seva utilització amb un exemple.

```
// Uso del operador de salida <<
#include<iostream>
using namespace std;

int main(void){
   int a=10; double x = 3.14; char c='A';
   char dia[]="Lunes";

   cout << "Este programa escribe los valores de a, x y c"<< "\n";
   cout << "a = "<< a << "\t x = "<< x << "\t c= "<< c << endl;
   cout << "Hoy es "<< dia ;

   return 0;
}
```

Algunes altres funcions del canal cout

- `cout.put(c);` on `c` és un caràcter
- `cout.write(cadena,longitud);` per a escriure `longitud` caràcters de la cadena.
- I, funcions per donar format a la sortida: `cout.width(amplada);` `cout.precision(nbDigits);`

L'operador d'entrada >>

Està predefinit per als tipus bàsics i per les cadenes de caràcters. Es pot sobrecargar (redefinir). Veiem la seva utilització amb un exemple.

```
// Uso del operador de entrada >>
#include<iostream>
using namespace std;

int main(void){
    int a;
    double x;
    char c;
    cout << "Introduzca un entero, un real y un caracter..."<< "\n";
    cin >> a >> x >> c;
    cout << "Los valores de a, x y c son"<<"\n";
    cout<<"a= "<< a <<"\t x= "<< x <<"\t c= "<< c << endl;

    return 0;
}
```

Composició d'instruccions

Les instruccions de un programa es poden composar

- Composició seqüencial
- Composició alternativa
- Composició iterativa

Composició seqüencial

Sintaxi: $A_1; A_2; \ldots; A_n;$ o bé

$A_1;$
$A_2;$
$\vdots$
$A_n;$

Exemple: *Programa per calcular la suma de dos enters donats*

```
// Programa per sumar dos números enters
```

Exemple: *Programa per calcular la suma de dos enters donats*

```
// Programa per sumar dos números enters
// Declaració de constants


int main(void) {
    // Declaració de variables


    // Accions/Sentències




    return 0;
}
```

Exemple: *Programa per calcular la suma de dos enters donats*

```
// Programa per sumar dos números enters
// Declaració de constants

// Precondició: cin: E1 E2
int main(void) {
    // Declaració de variables


    // Accions/Sentències




    return 0;
}
```

Exemple: *Programa per calcular la suma de dos enters donats*

```
// Programa per sumar dos números enters
// Declaració de constants

// Precondició: cin: E1 E2
int main(void) {
    // Declaració de variables


    // Accions/Sentències









    return 0;
}
// Postcondició: cout: E1+E2
```

Exemple: *Programa per calcular la suma de dos enters donats*

```
// Programa per sumar dos números enters
// Declaració de constants

// Precondició: cin: E1 E2
int main(void) {
    // Declaració de variables
    int x,y;

    // Accions/Sentències




    return 0;
}
// Postcondició: cout: E1+E2
```

Exemple: *Programa per calcular la suma de dos enters donats*

```
// Programa per sumar dos números enters
// Declaració de constants

// Precondició: cin: E1 E2
int main(void) {
    // Declaració de variables
    int x,y;

    // Accions/Sentències
    cin >> x;




    return 0;
}
// Postcondició: cout: E1+E2
```

Exemple: *Programa per calcular la suma de dos enters donats*

```
// Programa per sumar dos números enters
// Declaració de constants

// Precondició: cin: E1 E2
int main(void) {
    // Declaració de variables
    int x,y;

    // Accions/Sentències
    cin >> x;
    // cin: E2 i x=E1




    return 0;
}
// Postcondició: cout: E1+E2
```

Exemple: *Programa per calcular la suma de dos enters donats*

```
// Programa per sumar dos números enters
// Declaració de constants

// Precondició: cin: E1 E2
int main(void) {
    // Declaració de variables
    int x,y;

    // Accions/Sentències
    cin >> x;
    // cin: E2 i x=E1
    cin >> y;



    return 0;
}
// Postcondició: cout: E1+E2
```

Exemple: *Programa per calcular la suma de dos enters donats*

```
// Programa per sumar dos números enters
// Declaració de constants

// Precondició: cin: E1 E2
int main(void) {
    // Declaració de variables
    int x,y;

    // Accions/Sentències
    cin >> x;
    // cin: E2 i x=E1
    cin >> y;
    // x=E1 i y=E2


    return 0;
}
// Postcondició: cout: E1+E2
```

Exemple: *Programa per calcular la suma de dos enters donats*

```
// Programa per sumar dos números enters
// Declaració de constants

// Precondició: cin: E1 E2
int main(void) {
    // Declaració de variables
    int x,y;

    // Accions/Sentències
    cin >> x;
    // cin: E2 i x=E1
    cin >> y;
    // x=E1 i y=E2
    cout << x+y;

    return 0;
}
// Postcondició: cout: E1+E2
```

Exemple: *Programa per calcular la suma de dos enters donats*

```
// Programa per sumar dos números enters
// Declaració de constants

// Precondició: cin: E1 E2
int main(void) {
    // Declaració de variables
    int x,y;

    // Accions/Sentències
    cin >> x;
    // cin: E2 i x=E1
    cin >> y;
    // x=E1 i y=E2
    cout << x+y;
    // x=E1 i y=E2 i cout: E1+E2
    return 0;
}
// Postcondició: cout: E1+E2
```

Exemple: *Programa que llegeix un real i n'escriu el seu quadrat*

```
// Programa per calcular el quadrat d'un número real
```

Edicions UPC
Inici
Contingut
Pàgina 48 de 91
Tornar
Pantalla Completa
Tancar
Sortir

Exemple: *Programa que llegeix un real i n'escriu el seu quadrat*

```
// Programa per calcular el quadrat d'un número real
// Declaració de constants

int main(void) {
    // Declaració de variables

    // Accions/Sentències


    return 0;
}
```

Exemple: *Programa que llegeix un real i n'escriu el seu quadrat*

```
// Programa per calcular el quadrat d'un número real
// Declaració de constants

// Precondició: cin: R1
int main(void) {
    // Declaració de variables


    // Accions/Sentències



    return 0;
}
```

Exemple: *Programa que llegeix un real i n'escriu el seu quadrat*

```
// Programa per calcular el quadrat d'un número real
// Declaració de constants

// Precondició: cin: R1
int main(void) {
    // Declaració de variables


    // Accions/Sentències



    return 0;
}
// Postcondició: cout: R1*R1
```

Exemple: *Programa que llegeix un real i n'escriu el seu quadrat*

```
// Programa per calcular el quadrat d'un número real
// Declaració de constants

// Precondició: cin: R1
int main(void) {
    // Declaració de variables
    double x;

    // Accions/Sentències



    return 0;
}
// Postcondició: cout: R1*R1
```

Exemple: *Programa que llegeix un real i n'escriu el seu quadrat*

```
// Programa per calcular el quadrat d'un número real
// Declaració de constants

// Precondició: cin: R1
int main(void) {
    // Declaració de variables
    double x;

    // Accions/Sentències
    cin >> x;



    return 0;
}
// Postcondició: cout: R1*R1
```

Exemple: *Programa que llegeix un real i n'escriu el seu quadrat*

```
// Programa per calcular el quadrat d'un número real
// Declaració de constants

// Precondició: cin: R1
int main(void) {
    // Declaració de variables
    double x;

    // Accions/Sentències
    cin >> x;
    // x=R1


    return 0;
}
// Postcondició: cout: R1*R1
```

Exemple: *Programa que llegeix un real i n'escriu el seu quadrat*

```
// Programa per calcular el quadrat d'un número real
// Declaració de constants

// Precondició: cin: R1
int main(void) {
    // Declaració de variables
    double x;

    // Accions/Sentències
    cin >> x;
    // x=R1
    cout << x*x;

    return 0;
}
// Postcondició: cout: R1*R1
```

Exemple: *Programa que llegeix un real i n'escriu el seu quadrat*

```
// Programa per calcular el quadrat d'un número real
// Declaració de constants

// Precondició: cin: R1
int main(void) {
    // Declaració de variables
    double x;

    // Accions/Sentències
    cin >> x;
    // x=R1
    cout << x*x;
    // x=R1 i cout: R1*R1
    return 0;
}
// Postcondició: cout: R1*R1
```

Composició alternativa

Sintaxi:

if(C) {
    S_1
}
else {
    S_2
}

Semàntica:

- La condició C ha de ser una expressió booleana
- Les S_i són accions que poden estar composades utilitzant una o totes de les composicions possibles (seqüencial, alternativa i iterativa)
- Només s'executen les accions S_i d'una única branca. Les accions S_1 quan la condició C avalua a cert o les accions S_2 quan la condició C avalua a fals
- La branca else és optativa

Exemple: *Calcular la part entera per dalt d'un valor real donat*

```
// Càlcul de la part entera per dalt d'un valor real donat
// Declaració de constants

// Precondició: cin: R1
int main(void) {
    // Declaració de variables
    double r;
    int e;

    // Accions/Sentències
    cin >> r;
    // x=R1
    e = int(r);
    // x=R1   i   e=floor(r)
    if(double(e) < r) {
        e = e+1;
    }
    // x=R1   i   e=ceil(r)
    cout << e;
    // r=R1   i   e=ceil(R1)   i   cout: ceil(R1)
    return 0;
}
// Postcondició: cout: ceil(R1)
```

Exemple: *Calcular el màxim de dos valors enters*

Exemple: *Programa que donat un valor enter escrigui una p si és parell o una s si és senar*

Composició iterativa

Sintaxi:

while(C) {
S
}

Semàntica:

- La condició C ha de ser una expressió booleana
- S són accions que poden estar composades utilitzant qualsevol de les composicions possibles (seqüencial, alternativa i iterativa)
- Mentre C avaluï a cert s'executen les accions S
- Ha d'existir un valor $n \geq 0$ tal que executant n vegades S la condició C avalua a fals

Exemple: *Donats dos enters* E_1 *i* E_2 *amb* $E_2 \geq 0$ *calcular* $E_1^{E_2}$

// Programa que donats dos enters E1 i E2 amb E2 >= 0 calculi $E1^{E2}$

Exemple: *Donats dos enters E_1 i E_2 amb $E_2 \geq 0$ calcular $E_1^{E_2}$*

```
// Programa que donats dos enters E1 i E2 amb E2 >= 0 calculi E1^E2
// Declaració de constants

int main(void) {
    // Declaració de variables

    // Accions/Sentències

    return 0;
}
```

Exemple: *Donats dos enters E_1 i E_2 amb $E_2 \geq 0$ calcular $E_1^{E_2}$*

```
// Programa que donats dos enters E1 i E2 amb E2 >= 0 calculi E1^E2
// Declaració de constants

// Precondició: cin: E1 E2   i   E2 >= 0
int main(void) {
    // Declaració de variables


    // Accions/Sentències




    return 0;
}
```

Exemple: *Donats dos enters E_1 i E_2 amb $E_2 \geq 0$ calcular $E_1^{E_2}$*

```
// Programa que donats dos enters E1 i E2 amb E2 >= 0 calculi E1^E2
// Declaració de constants

// Precondició: cin: E1 E2   i   E2 >= 0
int main(void) {
    // Declaració de variables


    // Accions/Sentències









    return 0;
}
// Postcondició: cout: E1^E2
```

Exemple: *Donats dos enters E_1 i E_2 amb $E_2 \geq 0$ calcular $E_1^{E_2}$*

```
// Programa que donats dos enters E1 i E2 amb E2 >= 0 calculi E1^E2
// Declaració de constants

// Precondició: cin: E1 E2   i   E2 >= 0
int main(void) {
    // Declaració de variables
    int x,y,k,resul;

    // Accions/Sentències




    return 0;
}
// Postcondició: cout: E1^E2
```

Exemple: *Donats dos enters E_1 i E_2 amb $E_2 \geq 0$ calcular $E_1^{E_2}$*

```
// Programa que donats dos enters E1 i E2 amb E2 >= 0 calculi E1^E2
// Declaració de constants

// Precondició: cin: E1 E2   i   E2 >= 0
int main(void) {
    // Declaració de variables
    int x,y,k,resul;

    // Accions/Sentències
    cin >> x >> y;




    return 0;
}
// Postcondició: cout: E1^E2
```

Exemple: *Donats dos enters E_1 i E_2 amb $E_2 \geq 0$ calcular $E_1^{E_2}$*

```
// Programa que donats dos enters E1 i E2 amb E2 >= 0 calculi E1^E2
// Declaració de constants

// Precondició: cin: E1 E2   i   E2 >= 0
int main(void) {
    // Declaració de variables
    int x,y,k,resul;

    // Accions/Sentències
    cin >> x >> y;
    k = 0;




    return 0;
}
// Postcondició: cout: E1^E2
```

Exemple: *Donats dos enters E_1 i E_2 amb $E_2 \geq 0$ calcular $E_1^{E_2}$*

```
// Programa que donats dos enters E1 i E2 amb E2 >= 0 calculi E1^E2
// Declaració de constants

// Precondició: cin: E1 E2   i   E2 >= 0
int main(void) {
    // Declaració de variables
    int x,y,k,resul;

    // Accions/Sentències
    cin >> x >> y;
    k = 0; resul = 1;




    return 0;
}
// Postcondició: cout: E1^E2
```

Exemple: *Donats dos enters E_1 i E_2 amb $E_2 \geq 0$ calcular $E_1^{E_2}$*

```
// Programa que donats dos enters E1 i E2 amb E2 >= 0 calculi E1^E2
// Declaració de constants

// Precondició: cin: E1 E2   i   E2 >= 0
int main(void) {
    // Declaració de variables
    int x,y,k,resul;

    // Accions/Sentències
    cin >> x >> y;
    k = 0; resul = 1;
    while(k < y) {



    }

    return 0;
}
// Postcondició: cout: E1^E2
```

Exemple: *Donats dos enters E_1 i E_2 amb $E_2 \geq 0$ calcular $E_1^{E_2}$*

```
// Programa que donats dos enters E1 i E2 amb E2 >= 0 calculi E1^E2
// Declaració de constants

// Precondició: cin: E1 E2   i   E2 >= 0
int main(void) {
    // Declaració de variables
    int x,y,k,resul;

    // Accions/Sentències
    cin >> x >> y;
    k = 0; resul = 1;
    while(k < y) {
        resul = resul * x;

    }

    return 0;
}
// Postcondició: cout: E1^E2
```

Exemple: *Donats dos enters E_1 i E_2 amb $E_2 \geq 0$ calcular $E_1^{E_2}$*

```
// Programa que donats dos enters E1 i E2 amb E2 >= 0 calculi E1^E2
// Declaració de constants

// Precondició: cin: E1 E2   i   E2 >= 0
int main(void) {
    // Declaració de variables
    int x,y,k,resul;

    // Accions/Sentències
    cin >> x >> y;
    k = 0; resul = 1;
    while(k < y) {
        resul = resul * x;
        k = k + 1;
    }

    return 0;
}
// Postcondició: cout: E1^E2
```

Exemple: *Donats dos enters E_1 i E_2 amb $E_2 \geq 0$ calcular $E_1^{E_2}$*

```
// Programa que donats dos enters E1 i E2 amb E2 >= 0 calculi E1^E2
// Declaració de constants

// Precondició: cin: E1 E2   i   E2 >= 0
int main(void) {
    // Declaració de variables
    int x,y,k,resul;

    // Accions/Sentències
    cin >> x >> y;
    k = 0; resul = 1;
    while(k < y) {
        resul = resul * x;
        k = k + 1;
    }
    cout << resul;
    return 0;
}
// Postcondició: cout: E1^E2
```

Exemple: *Codi de les lletres majúscules*

```
// Programa que escriu les lletres majúscules
// conjuntament amb el seu codi
```

Exemple: *Codi de les lletres majúscules*

```
// Programa que escriu les lletres majúscules
// conjuntament amb el seu codi
// Declaració de constants

int main(void) {
    // Declaració de variables

    // Accions/Sentències



    return 0;
}
```

Exemple: *Codi de les lletres majúscules*

```
// Programa que escriu les lletres majúscules
// conjuntament amb el seu codi
// Declaració de constants

// Precondició:
int main(void) {
    // Declaració de variables


    // Accions/Sentències




    return 0;
}
```

Exemple: *Codi de les lletres majúscules*

```
// Programa que escriu les lletres majúscules
// conjuntament amb el seu codi
// Declaració de constants

// Precondició:
int main(void) {
    // Declaració de variables


    // Accions/Sentències




    return 0;
}
// Postcondició: Al canal cout s'han escrit les lletres
// majúscules conjuntament amb el seu codi
```

Exemple: *Codi de les lletres majúscules*

```
// Programa que escriu les lletres majúscules
// conjuntament amb el seu codi
// Declaració de constants

// Precondició:
int main(void) {
    // Declaració de variables
    char lletra = 'A';

    // Accions/Sentències



    return 0;
}
// Postcondició: Al canal cout s'han escrit les lletres
// majúscules conjuntament amb el seu codi
```

Exemple: *Codi de les lletres majúscules*

```
// Programa que escriu les lletres majúscules
// conjuntament amb el seu codi
// Declaració de constants

// Precondició:
int main(void) {
    // Declaració de variables
    char lletra = 'A';

    // Accions/Sentències
    while(lletra <= 'Z') {


    }
    return 0;
}
// Postcondició: Al canal cout s'han escrit les lletres
// majúscules conjuntament amb el seu codi
```

Exemple: *Codi de les lletres majúscules*

```
// Programa que escriu les lletres majúscules
// conjuntament amb el seu codi
// Declaració de constants

// Precondició:
int main(void) {
    // Declaració de variables
    char lletra = 'A';

    // Accions/Sentències
    while(lletra <= 'Z') {
        cout << lletra << ":  "<< int(lletra) << endl;

    }
    return 0;
}
// Postcondició: Al canal cout s'han escrit les lletres
// majúscules conjuntament amb el seu codi
```

Exemple: *Codi de les lletres majúscules*

```
// Programa que escriu les lletres majúscules
// conjuntament amb el seu codi
// Declaració de constants

// Precondició:
int main(void) {
    // Declaració de variables
    char lletra = 'A';

    // Accions/Sentències
    while(lletra <= 'Z') {
        cout << lletra << ":  "<< int(lletra) << endl;
        lletra = char(int(lletra)+1);
    }
    return 0;
}
// Postcondició: Al canal cout s'han escrit les lletres
// majúscules conjuntament amb el seu codi
```

Composició iterativa: **for**

Sintaxi:

```
for(I; C; AI) {
    S
}
```

Semàntica:

- La inicialització I es dur a terme un únic cop abans de començar la iteració
- La condició C ha de ser una expressió booleana
- S són accions que poden estar composades utilitzant qualsevol de les composicions possibles (seqüencial, alternativa i iterativa)
- Mentre C avaluï a cert s'executen les accions S i a continuació l'acció d'increment AI
- Ha d'existir un valor $n \geq 0$ tal que executant n vegades S i AI la condició C avalua a fals

Exemple: *Codi de les lletres majúscules utilitzant for*

```
// Programa que escriu les lletres majúscules
// conjuntament amb el seu codi
// Declaració de constants

// Precondició:
int main(void) {
    // Declaració de variables
    char lletra;

    // Accions/Sentències
    for(lletra = 'A'; lletra <= 'Z'; lletra = char(int(lletra)+1)) {
        cout << lletra << ":  "<< int(lletra) << endl;
    }
    return 0;
}
// Postcondició: Al canal cout s'han escrit les lletres
// majúscules conjuntament amb el seu codi
```

Estructura d'un programa

```
// Declaració de llibreries
#include <...>

// Declaració de constants

// Programa principal
int main() {
    // Declaració de variables

    // Accions/Sentències

    return 0;
}
```

3. Esquemes algorísmics bàsics

- Per resoldre un problema concret, no hi ha prou amb conèixer un llenguatge de programació. S'han d'aplicar tècniques i mètodes programació.
- Desafortunadament, no existeix una metodologia o tècnica que es pugui aplicar a qualsevol problema. No obstant, hi han metodologies i tècniques que són aplicables a famílies de problemes.
- A molts problemes, les dades a tractar estan organitzades en forma de una seqüència. Aquest problemes es coneixen en programació com problemes de **tractament seqüencial**.
- Hi ha dos esquemes de programació que permeten resoldre aquest tipus de problemes de forma eficaç i eficient.
- Aplicar els esquemes proporciona garantia de programes correctes.

3.1. Seqüències

Un conjunt finit d'elements està organitzat en forma de seqüència si és possible definir els conceptes següents:

- Primer element: x_1
  - Element amb la característica de que s'ha d'haver accedit a ell per a poder accedir a la resta d'elements de la seqüència.
- Obtenció del següent: $x_{k+1} = f(x_k, k)$
  - Tot element menys l'últim té un següent, determinat per la relació anterior.
  - A partir del primer es coneixen tots.
- Darrer element de la seqüència:
  - Coneixement de l'últim element que anomenarem *sentinella* (element que realment no pertany a la seqüència): $x_k = sentinella$
  - Propietat p que compleixen tots els elements de la seqüència: $\neg p(x_k)$
  - Propietat concreta fi del primer element que no pertany a la seqüència: $fi(x_k)$
  - Coneixement del nombre d'elements que té la seqüència: $k = n$

Cas particular. Seqüències d'entrada

- Primer element: x_1 = LlegirElement
- Obtenció del següent: x_{k+1} = LlegirElement
- Darrer element de la seqüència: normalment per *sentinella* però es poden donar també els altres casos.

Exemples

1) Seqüència dels nombres del 1 al 100.

- Primer element: $x_1 = 1$
- Obtenció del següent: $x_{k+1} = x_k + 1$
- Darrer element de la seqüència: $x_k > 100$

2) Les potències de dos més petites que 1024.

- Primer element: $x_1 = 1$
- Obtenció del següent: $x_{k+1} = x_k * 2$
- Darrer element: $x_k \geq 1024$

Exemples

3) A l'entrada tenim una frase acabada en *punt*.

- Primer element: x_1 = LlegirCaracter
- Obtenció del següent: x_{k+1} = LlegirCaracter
- Darrer element: x_k = '.'

4) Donat un conjunt de valors reals que representen les arestes d'un polígon definir la seqüència de les arestes.

- Primer element: $x1_1$ = LlegirReal; $y1_1$ = LlegirReal
  $x2_1$ = LlegirReal; $y2_1$ = LlegirReal
- Obtenció del següent: $x1_{k+1} = x2_k$; $y1_{k+1} = y2_k$
  $x2_{k+1}$ = LlegirReal;
  $y2_{k+1}$ = LlegirReal
- Darrer element: $x2_k = x1_1 \wedge y2_k = y1_1$

3.2. Programació amb esquemes

Tots els problemes que tenen associada una seqüència es poden resoldre utilitzant un dels dos esquemes següents:

- Recorregut: esquema que cal aplicar quan per a resoldre el problema sempre cal accedir a tots els elements de la seqüència.
- Cerca: esquema que cal aplicar quan en algun cas es pot resoldre el problema sense accedir a tots els elements de la seqüència.

3.3. Esquema de recorregut

```
IniciTractament
ObtenirPrimerElement
while(!DarrerElement) {
        TractarElement
        ObtenirSeguentElement
}
TractamentFinal
```

Exemple: *Dissenyar un algorisme que donada una frase acabada en* $punt$ *compti el nombre de caràcters que té*

Exemple: *Dissenyar un programa que donada una frase acabada en* $punt$ *compti el nombre de* as *que té*

3.4. Esquema de cerca

```
bool trobat = false;
IniciTractament
ObtenirPrimerElement
while(!DarrerElement && !trobat) {
    if(PropietatElement) {
        trobat = true;
    }
    else {
        TractarElement
        ObtenirSeguentElement
    }
}
if(trobat) {
    TractarTrobat
}
else {
    TractarNoTrobat
}
```

Exemple: *Dissenyar un programa que donada una frase acabada en $punt$ escrigui una s si la frase té una z o una n en cas contrari*

Exemple: *Dissenyar un programa que donada una frase acabada en* $punt$ *escrigui el nombre de caràcters que hi ha abans de la primera z si és que la frase en té una o una n si no en té*

4. Subprogrames: Accions i funcions

Motivació

- Els programes que hem implementat fins ara consten d'un tot, els càlculs se es fan totalment dins del (únic) flux d'execució implementat a la funció `main`.
- En general, això comporta dificultats a l'hora de resoldre problemes sobretot quan tenen certa complexitat:
  - Els problemes són difícils de resoldre, ja que sempre es parteix de zero.
  - Els programes són llargs i difícils d'entendre (poc llegibles).
  - No es pot evitar resoldre el mateix problema més d'una vegada. Es repeteix codi.
  - Els manteniment dels programes (canvis, extensions, millores, etc.) es força complicat, canvis a una part del programa impliquen canvis a altres parts.

Solució: Disposar d'un conjunt d'accions i funcions més ampli

Si el conjunt d'accions i/o funcions disponibles fos més ampli:

- Seria més fàcil resoldre els problemes.
- Obtindríem un major nivell d'abstracció.
- Els programes resultarien més curts.
- Els programes serien més intelůligibles.
- Evitaria haver de resoldre un mateix problema més d'un cop. No repetició de codi.
- Facilitaria el manteniment dels programes. Millores, canvis, ampliacions, etc.

Edicions UPC
Inici
Contingut
Pàgina 73 de 91
Tornar
Pantalla Completa
Tancar
Sortir

Exemples

Suposem que volem dissenyar un programa que donats quatre valors enters: E1, E2, E3 i E4 calculi $E1^{E2}$ i $E3^{E4}$

Edicions UPC
Inici
Contingut
Pàgina 73 de 91
Tornar
Pantalla Completa
Tancar
Sortir

Exemples

Suposem que volem dissenyar un programa que donats quatre valors enters: E1, E2, E3 i E4 calculi $E1^{E2}$ i $E3^{E4}$

```
// Precondició: cin: E1 E2 E3 E4   i   E2 >= 0   i E4 >= 0
int main(void)
{
    int w,x,y,z,k,res;
    cin >> w >> x >> y >> z;
    k = 0; res = 1;
    while(k < x) {
      res = res * w; k = k+1;
    }
    cout << res;




    return 0;
}
// Postcondició: cout: E1^E2 E3^E4
```

Exemples

Suposem que volem dissenyar un programa que donats quatre valors enters: E1, E2, E3 i E4 calculi $E1^{E2}$ i $E3^{E4}$

```
// Precondició: cin: E1 E2 E3 E4   i   E2 >= 0   i E4 >= 0
int main(void)
{
    int w,x,y,z,k,res;
    cin >> w >> x >> y >> z;
    k = 0; res = 1;
    while(k < x) {
      res = res * w; k = k+1;
    }
    cout << res;
    k = 0; res = 1;
    while(k < z) {
      res = res * y; k = k+1;
    }
    cout << res;
    return 0;
}
// Postcondició: cout: E1^E2 E3^E4
```

Exemples

Suposem ara que per resoldre el mateix problema disposem d'una funció `exp(x,y)` que donats dos valors enters `x`, `y` calcula x^y.

Exemples

Suposem ara que per resoldre el mateix problema disposem d'una funció `exp(x,y)` que donats dos valors enters `x`, `y` calcula x^y.

```
// Precondició: cin: E1 E2 E3 E4   i   E2 >= 0   i E4 >= 0
int main(void)
{
    int w,x,y,z,k,res;
    cin >> w >> x >> y >> z;
    cout << exp(w,x) << ' '<< exp(y,z);
    return 0;
}
// Postcondició: cout: E1^E2 E3^E4
```

Edicions UPC
Inici
Contingut
Pàgina 75 de 91
Tornar
Pantalla Completa
Tancar
Sortir

Els llenguatges de programació ofereixen un mecanisme d'abstracció que permet ampliar el conjunt d'accions i/o funcions dels que disposem. Aquest mecanisme és la definició d'accions i/o funcions parametritzades.
Les accions i funcions parametrizades són subprogrames que:

- Resolen un problema concret.
- Les dades que necessiten per resoldre el problema es reben mitjançant els paràmetres.
- Tenen un entorn propi que consisteix en:
  - Els paràmetres formals.
  - Les variables que es defineixen a l'acció o funció. Aquestes variables s'anomenen variables locals.
- No es poden executar de forma independent.
- Per executar-se han de ser cridades o bé des de un programa o bé des d'una altra acció o funció.

Definició d'accions

Les accions són subprogrames que poden modificar l'estat de l'programa, acció o funció que les criden. La definició d'una acció consta de dues parts:

- Capçalera: és on es dóna un nom a l'acció i on es defineixen els paràmetres formals que té l'acció (nombre i tipus).
- Cos: és el subprograma que resol el problema associat a l'acció.

Sintaxi:

```
void <nom>(<paramFormal1>,<paramFormal2>,...,<paramFormaln>) {
    // Declaració d'objectes (constants i variables) locals
    // Sentencies (composició d'accions)
}
```

Paràmetres formals d'una acció

S'anomenen paràmetres formals els paràmetres que apareixen a la capçalera de la definició d'una acció o funció.

- Els paràmetre són el mecanisme que permet a una acció o funció comunicar-se amb la resta de l'entorn (program, altre acció o altre funció) on s'ha invocat.
- A traves dels paràmetres, l'entorn proporciona a l'acció els valors que aquesta necessita per a resoldre la tasca associada i l'acció li proporciona a l'entorn els resultats.
- Aquest mecanisme de comunicació rep el nom de pas de paràmetres.

Edicions UPC
Inici
Contingut
Pàgina 78 de 91
Tornar
Pantalla Completa
Tancar
Sortir

Els paràmetres formals es poden classificar en:

- Entrada. Un paràmetre d'entrada és un valor que necessita conèixer el subprograma per resoldre el problema. En aquest cas el pas de paràmetres es fa per valor.
- Sortida. Un paràmetre de sortida és una variable, que no ha de tenir un valor inicial, on el subprograma deixarà un valor (resultat). En aquest cas el pas de paràmetres es fa per referència i s'indica posant el símbol & davant del nom del paràmetre.
- Entrada/sortida. Un paràmetre de entrada/sortida és una variable que ha de tenir un valor inicial i que el subprograma pot modificar. En aquest cas el pas de paràmetres es fa per referència i s'indica posant el símbol & davant del nom del paràmetre.

Sintaxi: *<tipus> {&}<nom>*

Exemples

Acció que donats dos enters a i b calcula $c = a * b$ i canvia el valor de a pel de b.

```
// Precondició: a = E1   i   b = E2
void nose(int &a, int b, int &c) {
    c = a*b;
    a = b;
}
// Postcondició: a = E2   i   b = E2   i   c = E1*E2
```

Acció que intercanviï el valor de dos enters.

```
// Precondició: a = E1   i   b = E2
void intercanvia(int &a, int &b) {
    int aux;
    aux = a; a = b; b = aux;
}
// Postcondició: a = E2   i   b =E1
```

Un altre exemple

Acció que calcula si una equació de segon grau ($ax^2 + bx + c = 0$) té solució o no i en cas que en tingui calcula les seves arrels.

```
void arrels(double a, double b, double c, double &x1, double &x2, bool &s) {
    double aux;
    aux = b*b-4.0*a*c;
    if(aux < 0.0) {
        s = false;
    }
    else {
        aux = sqrt(aux);
        x1 = (-b+aux)/(2.0*a);
        x2 = (b+aux)/(2.0*a);
        s = true;
    }
}
```

Crida d'accions

La crida a una acció consisteix en executar el subprograma corresponent amb uns paràmetres concrets. Els paràmetres concrets amb els que es crida l'acció s'anomenen paràmetres actuals o arguments.

Sintaxi: *<nom>(<paramActual1>,<paramActual2>,...,<paramActualn>)*

- Quan es crida a una acció primer s'actualitzen els seus paràmetres formals segons els paràmetres actuals corresponents i després s'executa el subprograma.
- La correspondència entre els paràmetres formals i els actuals es fa segons l'ordre, és a dir, el primer amb el primer, el segon amb el segon, ... MAI SEGONS EL NOM!

L'actualització dels paràmetres formals es fa, depenent de si el pas és per valor o per referència, de la següent forma:

- Pas per Valor: es copia el valor del paràmetre actual en el paràmetre formal.
- Pas per Referència: es copia l'adreça de memòria de la variable corresponent al paràmetre actual en el paràmetre formal i per tant cada vegada que es fa referència al paràmetre formal al cos de l'acció s'està fent referència a la variable que s'ha passat com a paràmetre actual (les modificacions i consultes es fan sobre la variable corresponent al paràmetre actual).

Edicions UPC
Inici
Contingut
Pàgina 83 de 91
Tornar
Pantalla Completa
Tancar
Sortir

Ha d'existir una correspondència entre els paràmetres actuals i els paràmetres formals en:

- Nombre: si l'acció té n paràmetres formals a la seva definició quan es crida ha de tenir n paràmetres actuals.
- Tipus: el tipus dels paràmetres actuals ha de coincidir amb el tipus amb el que s'han definit el paràmetres formals corresponents (el primer amb el primer, el segon amb el segon, ...).
- Significat: per exemple quan cridem a l'acció arrels el primer paràmetre actual ha de ser el coeficient de segon grau, el segon el de primer grau i el tercer el de grau zero. Si volem resoldre la equació de segon grau $3x^2 + x + 5 = 0$ hem de cridar a l'acció de la següent manera:

```
arrels(3.0,1.0,5.0,x1,x2,s)
```

Definició de funcions

Les funcions són subprogrames que no poden modificar l'estat del programa, acció o funció que les criden. Les funcions calculen un valor a partir dels paràmetres i per tant tenen associat un tipus. La definició d'una funció, al igual que les accions, consta de dues parts:

- Capçalera: és on es dóna un nom a la funció, on es defineixen els paràmetres formals que té la funció (nombre i tipus) i on s'indica de quin tipus és el valor que calcula.
- Cos: és el subprograma que resol el problema associat a la funció.

Sintaxi:

```
<tipus> <nom>(<paramFormal1>,<paramFormal2>,...,<paramFormaln>) {
    // Declaració d'objectes (constants i variables) locals
    // Sentencies (composició d'accions)
    return <expressió>;
}
```

Paràmetres formals d'una funció

Tots els paràmetres formals d'una funció són d'entrada.

Sintaxi: *<tipus> <nom>*

Exemple

Funció que calculi la part entera per dalt d'un valor real donat.

```
// Precondició: r=R1
int arrodonir(double r) {
    int e;
    e = int(r);
    if (double(e) < r) {
        e = e+1;
    }
    return e;
}
// Postcondició: arrodonir(R1)=ceil(R1)
```

Exemple

Funció que calculi el màxim de dos valors enters.

```
// Precondició: x=E1  i  y=E2
int max(int x, int y) {
    int m;
    if(x<y) {
      m = y;
    }
    else {
      m = x;
    }
    return m;
}
// Postcondició: (max(E1,E2) = E1  i  E1>=E2)  o
//                (max(E1,E2) = E2  i  E2>=E1)
```

Exemple

Funció que donats dos enters E_1 i E_2 amb $E_2 \geq 0$ calculi $E_1^{E_2}$.

```
// Precondició: x=E1  i  y=E2  i  E2 >= 0
int exp(int x, int y) {
    int k, resul;
    k = 0; resul = 1;
    while(k < y) {
        resul = resul*x;
        k = k+1;
    }
    return resul;
}
// Postcondició: exp(E1,E2) = E1^E2
```

Crida de funcions

La crida a una funció consisteix en executar el subprograma corresponent amb uns paràmetres concrets. Els paràmetres concrets amb els que es crida l'acció s'anomenen paràmetres actuals o arguments.

Sintaxi: *<nom>(<paramActual1>,<paramActual2>,...,<paramActualn>)*

- Quan es crida a una funció primer s'actualitzen els seus paràmetres formals segons els paràmetres actuals corresponents, després s'executa el subprograma i finalment es substitueix la crida a la funció per el valor que retorna la funció.
- La correspondència entre els paràmetres formals i els actuals (al igual que en les accions) es fa segons l'ordre, és a dir, el primer amb el primer, el segon amb el segon, . . . MAI SEGONS EL NOM!
- Recordeu que una funció és una expressió i per tant la crida a una funció només pot aparèixer on pot aparèixer una expressió.

Exemple

Funció que calculi el màxim de tres valors enters.

```
// Precondició: x=E1  i  y=E2  i  z=E3
int max3(int x, int y, int z) {
      return max(max(x,y),z);
}
// Postcondició:(max3(E1,E2,E3) = E1  i  E1>=E2  i  E1>=E3)  o
//                (max3(E1,E2,E3) = E2  i  E2>=E1  i  E2>=E3)  o
//                (max3(E1,E2,E3) = E3  i  E3>=E1  i  E3>=E2)
```

5. Tipus Estructurats: Taules i tuples

Motivació

- Disposem de variables escalars (valors elementals; ordre dins del conjunt de valors)
- Seqüències $\Rightarrow$ accés seqüencial. No es pot accedir directament a les dades.