Entorn de treball - Repàs de C++

1.1 Entorn de treball

Per escriure programes en C++ cal un editor de text i un terminal on escriure les ordres de compilació, muntatge i execució del programa escrit.

- Obriu l'editor que preferiu: `kate`, `gedit`, `emacs`, etc.
- Creeu un nou fitxer i escriviu-hi el següent programa que llegeix 3 enters de teclat i mostra per pantalla la seva suma.

```
#include <iostream>
using namespace std;
int main() {
  cout << "Entra tres nombres enters: " << endl;
  int x, y, z;
  cin >> x >> y >> z;
  int suma = x+y+z;
  cout << "La suma de " << x << ", " << y << " i " << z;
  cout << " es " << suma <<endl;
}
```

- Graveu el programa amb un nom de fitxer amb extensió `.cc` (per ex. `suma3.cc`).
- Aneu al terminal i compileu el programa amb la comanda:

  ```
  g++ -D_GLIBCXX_DEBUG -Wall -Wextra -Werror -std=c++11 -c suma3.cc
  ```

  Observació: Durant tot el curs utilitzarem l'estàndard ISO C++11 del llenguatge C++. La resta d'opcions de la comanda serveixen per activar diferents tipus de Warnings del compilador i fer aquests avisos més entenedors.
- Si apareix algun error és que el programa C++ no està ben escrit. Cal tornar a l'editor i corregir l'error abans de tornar a compilar.
- Quan ja no apareguin errors, veureu que s'ha creat un fitxer objecte `suma3.o`. Aquest fitxer encara no és executable. Per aconseguir el seu executable cal muntar (enllaçar o linkar) fent:

  ```
  g++ -o suma3.exe suma3.o
  ```

Si ho heu fet tot correctament hauríeu de veure l'executable: `suma3.exe`

Observació: Podem compilar i muntar en un sol pas si combinem les dues comandes anteriors:

```
g++ -D_GLIBCXX_DEBUG -Wall -Wextra -Werror -std=c++11 -o suma3.exe suma3.cc
```

- Podrem executar-lo fent:

```
./suma3.exe
```

Consell: Us podeu definir un àlies per evitar escriure totes les opcions de compilació:

```
alias pro1++='g++ -D_GLIBCXX_DEBUG -Wall -Wextra -Werror -std=c++11'
```

Ara podreu compilar (o compilar/muntar) usant l'àlies `pro1++`:

```
pro1++ -o suma3.exe suma3.cc
```

1.1.1 Execució de programes en C++

Els programes que codificarem en aquesta assignatura faran la gestió de l'E/S pels canals estàndards (teclat/pantalla). Per programes llargs, redireccionarem l'E/S a fitxers de text. Per exemple, si volem executar el programa que es troba al fitxer `suma3.exe`, llegint les dades pel teclat i escrivint els resultats per pantalla, escriurem l'ordre:

```
./suma3.exe
```

Si volem executar el mateix programa sobre dades emmagatzemades en el fitxer `suma3.ent`, però volem veure els resultats per pantalla, redireccionarem el canal estàndard d'entrada cap al fitxer `suma3.ent`:

```
./suma3.exe < suma3.ent
```

Si a més volem que els resultats no s'escriguin per pantalla sinó en un altre fitxer `suma3.sort`, executarem l'ordre:

```
./suma3.exe < suma3.ent > suma3.sort
```

Aquests són els mecanismes de redireccionament dels canals estàndards d'entrada i sortida en Linux. Normalment usareu les comandes anteriors amb aquest mateix ordre, la primera per provar si el programa funciona amb l'entrada teclejada per vosaltres, després la segona comanda per provar si el programa funciona amb un fitxer d'entrada concret i finalment la tercera comanda per desar la sortida en un fitxer i així comparar-la amb la sortida prevista (ho podeu fer amb la comanda `diff suma3.sort suma3prevista.sort` que no dirà res si els dos fitxers són idèntics, en cas contrari indicarà les línies on hi ha les diferències i quines són).

1.1.2 Estructura d'un programa en C++

Un programa en C++ pot constar d'un sol fitxer (normalment amb l'extensió `.cc` o `.cpp`; en aquesta assignatura fem servir l'extensió `.cc`). L'estructura d'aquest fitxer seria la següent:

- inclusions i espai de noms
- definicions de constants o tipus
- accions i/o funcions
- programa principal (`main`)

Les inclusions permeten al programa fer servir elements definits en altres llocs. Moltes són classes o llibreries del C++:

```
#include <iostream>:  canals estàndard d'entrada i sortida
#include <vector>:  classe vector
#include <cmath>:  funcions matemàtiques
```

Quan un programa creix força, segurament necessitarà una descomposició en mòduls. El C++ ja ofereix un mecanisme per obtenir programes modulars: la classe. A PRO1 usarem les classes per implementar mòduls de dades. Per cada programa C++, tindrem un fitxer `.cc` amb el `main` corresponent i diversos fitxers que contindran la resta de classes que necessiti.

1.2 Entorn de treball en el vostre PC

Els que treballeu amb portàtil o amb el PC de casa vostra, podeu accedir remotament al servidor `ahto` de l'escola on teniu el mateix entorn que en els PCs de les aules informàtiques. Ho podeu fer executant la següent comanda en una terminal, on usuari és el vostre usuari de linux (habitualment el vostre DNI canviant el primer número per una lletra):

```
ssh usuari@ahto.epsevg.upc.edu
```

Tot el material de l'assignatura PRO1 està a la carpeta `/home/public/pro1`.

Si preferiu copiar els fitxers de PRO1 del servidor `ahto` en el vostre portàtil o PC per tenir el vostre propi entorn amb carpetes que siguin iguals a les originals, ho podeu fer executant les següents comandes en el vostre PC:

```
sudo mkdir -p /home/public/pro1
sudo chown -R usuarivostrepc:users /home/public
scp -r usuari@ahto.epsevg.upc.edu:/home/public/pro1/* /home/public/pro1
```

1.3 Repàs de pas de paràmetres. Exercicis amb funcions/accions

En aquesta assignatura demanarem que totes les variables que intervinguin en les accions i funcions que dissenyem, estiguin a les seves capçaleres o bé siguin locals (no es permeten variable globals).

Recordem els diferents tipus de pas de paràmetres en C++:

1. **Paràmetres per valor**: l'acció rep una còpia del paràmetre i treballa amb aquesta còpia. Quan acaba, la còpia és destruïda. El valor del paràmetre no canvia.

   Per indicar que un paràmetre es passa per valor no cal fer res.

2. **Paràmetres per referència**: l'acció rep l'adreça de memòria del paràmetre i qualsevol modificació que se li faci és permanent.

   Per indicar que un paràmetre es passa per referència, es posa el signe & davant del seu nom.

3. **Paràmetres per referència constant**: com passa amb els paràmetres per referència, l'acció rep l'adreça de memòria del paràmetre, però, si intenta modificar el seu valor, el compilador dóna un error. Així doncs, funcionen com a paràmetres per valor.

   Per indicar que un paràmetre es passa per referència constant, es posa el signe & davant del seu nom i la paraula `const` davant del seu tipus.

1.3.1 Exercici: suma d'una seqüència d'enters

Escriviu un programa que sumi els valors d'una seqüència d'enters acabada en zero. Aneu processant els valors a mida que els llegiu. No useu cap estructura de dades per guardar la seqüència. Executeu el programa de forma interactiva i redireccionant l'entrada.

1.3.2 Exercici: cerca lineal en un vector d'enters

En el fitxer `cercalineal.cc` que us donem hi ha un programa basat en l'algorisme de cerca lineal en un vector d'enters. La capçalera de la funció de cerca és la següent:

```
bool cercalineal(const vector<int> &v, int x)
/* Pre: cert */
/* Post: el resultat és un booleà que indica si x es
   troba dins de v */
```

Modifiqueu la funció perquè retorni un resultat booleà que indiqui si l'element buscat està en el vector i, en cas afirmatiu, indiqui la posició en la que es troba l'element. Reescriviu la postcondició perquè el valor del resultat quedi ben definit en el cas de que l'element buscat no estigui en el vector.

El resultat del programa ha de ser un dels següents (només pot usar-se `cout` en el `main`):

```
L'element no es troba en el vector.
L'element es troba en el vector en la posició <la que sigui>
```

Per poder tornar dos resultats els hem d'agrupar dins d'un `struct` o un `pair`, ja que el llenguatge C++ té la limitació que les funcions només poden retornar un únic resultat. En el nostre cas necessitem retornar un booleà i un enter:

```
struct info_cerca {
  bool trobat;
  int posicio;
};
```

o bé

```
pair<bool, int>
```

Podeu inicialitzar els valors d'un pair en el moment que es declara la variable:

```
pair<bool, int> res(false, 0);
```

I podeu accedir a cada membre del pair per consultar-lo o modificar-lo amb `first` i `second`.

```
res.first = true;
cout << res.first;
```

1.3.3 Exercici: intercanvi de dos elements d'un vector

Escriviu un programa que llegeixi n enters i els guardi en un vector. A continuació, ha de localitzar les posicions en què es troben el màxim i el mínim, intercanviar-los de posició, i escriure el vector resultant.

Feu que el programa s'estructuri en tres funcions o accions: Una que llegeixi els elements del vector, una que localitzi i intercanviï el màxim i el mínim, i una altra que escrigui el vector resultant. La mida del vector és una dada d'entrada més a llegir. Podeu aprofitar les funcions de llegir i mostrar vector de l'exercici anterior.

1.4 Python

El llenguatge Python (https://www.python.org/) és un llenguatge de programació interpretat, d'alt nivell i propòsit general. Creat l'any 1991 per Guido Van Rossum Python està dissenyat per facilitar la llegibilitat dels programes, tant a petita com a gran escala.

El llenguatge de programació Python està dissenyat per ser multi-paradigma, suportant des de programació orientada a objectes i programació estructurada fins a meta-programació i programació funcional basada en *list comprehension*, *filter*, *map* i *reduce*.

Inicialment desenvolupat per tasques de *scripting* actualment disposa d'un gran nombre de llibreries que el fan adient per desenvolupar aplicacions en camps com la Intel·ligència Artificial o la Bio-Informàtica.

1.4.1 Python: Introducció

Es tracta d'un llenguatge interpretat (tot i que suporta compilació a binari), per exemple, per executar el programa contingut al fitxer *exemple.py* hem d'executar al terminal:

```
python3 exemple.py
```

Python treballa amb tipat dinàmic, és a dir, en comptes d'especificar el tipus de valor quan definim una variable, aquest tipus s'infereix automàticament al assignar-li un valor, per exemple:

```
prova = 32  # Aquí prova és un enter
prova = 32.54  # Aquí prova és un real (float)
prova = "32"  # Aquí prova és un string
prova = '32'  # Aquí prova és un string, " ' """ són delimitadors
prova = int("32")  # prova és un enter creat a partir d'un string
```

D'aquesta forma, hem de transformar les dades que llegim del canal d'entrada al tipus de dada adequat, per exemple:

```
edat = int(input("Introdueix la teva edat: "))
nom = str(input("Introdueix el teu nom: "))
alsada = float(input("Introdueix la teva alçada: "))
print ("La teva edat és %d, el teu nom %s i la teva alçada %f."
  % (edat, nom, alsada))
```

1.4.2 Python: Introducció: Exercici

Desenvolupa i executa un programa Python que guardi en dues variables els valors *resposta* i *42* com a string i enter respectivament. Després, el programa ha de mostrar per pantalla el missatge *La resposta es '42'* fent servir les variables definides anteriorment.

1.4.3 Python: Operadors

Python disposa del següent conjunt d'operadors:

```
x = 3  # Operador d'assignació
y = 5  # Operador d'assignació
z = x + y  # Suma
z = x - y  # Resta
z = x * y  # Multiplica
z = x / y  # Divideix
z = x ** y  # Potència
z = x // y  # Quocient de la divissió entera
z = x % y  # Residu (mòdul) de la divissió entera
b1 = True  # Booleans
b2 = False  # Booleans
b3 = not (b1 and b2) or (b1)  # Operacions booleanes
b3 = x == y  # Comparació, igualtat
b3 = x != y  # Comparació, diferencia
b3 = x < y  # Comparació, menor que estricte
b3 = x <= y  # Comparació, menor que no estricte
b3 = x > y  # Comparació, major que estricte
b3 = x >= y  # Comparació, major que no estricte
```

També permet treballar amb vectors i matrius. Els vectors es representen amb llistes de Python i les matrius amb llistes de llistes:

```
vector = [5,7,9]  # Llista amb valors => Vector
vectorb = []  # Llista buida => Vector buit
vector.append(12)  # Afegim valor al vector
vector = vector.append(15)  # No feu això mai!!
print(vector[0])  # Consulta primera posició del vector
print(len(vector))  # Nombre d'elements al vector
vector[-1] = 666  # Modifica valor de posició vector
del(vector[len(vector)-1])  # Esborra l'última posició
mini = vector[0:2]  # Nou vector amb pos. 0 i 1 del original
mat = [ [0,1] , [2,3] ]  # Matriu com a llista de llistes
```

Les funcions a Python es defineixen d'aquesta forma. Fixeu-vos que Python no té caràcters especials per delimitar blocs de codi (coms els { i } de C++); Python usa la identació (espai en blanc al principi de cada línia) per diferenciar els blocs de codi.

```
def nom_funcio (param_1, ...., param_n):
  """
  Explicació:
  PRE:
  POST:
  """
  return resultats #La variable conté els resultats
```

Per exemple:

```
def suma_enters (x, y):
  """
  Explicació: Suma dos enters
  PRE: x i y son enters
  POST: Retorna x + y
  """
  # Compte amb el tabulador!
  suma = x + y
  return suma
```

Els tipus de dades simples (com ara enter, real, string, booleà) es passen per valor (còpia) i els complexes (vectors, matrius, diccionaris) es passen per referència.

Python suporta composició alternativa:

```
# Compte amb el tabulador!
if (x > y):
  major = x
elif (x < y):
  major = y
else:
  major = cap
```

També composició iterativa, en les següents formes:

```
limit = 10
i = 0
vector = []
while i < limit:
  # Compte amb el tabulador!
  vector.append(i)
  i += 1 #Auto increment (Python no té operador ++ de C++)
```

```
  # Element va prenent per valor cada element del vector
  for element in vector:
    print (element)

# i comença amb 0, acaba amb len(vector) sense incloure
# l'element i augmenta d'un en un
for i in range(0, len(vector), 1):
  print (vector[i])
```

1.4.4 Python: Operadors: Exercici I

Desenvolupa un programa Python que llegeixi la mida d'un vector d'enters i el vector corresponent des del canal d'entrada. Desenvolupa una funció que busqui el màxim i el mínim al vector d'enters llegit. Crida la funció des del programa principal, recull els resultats i mostra'ls per la sortida estàndard.

1.4.5 Python: Operadors: Exercici II

Desenvolupa un programa Python que llegeixi un enter des del canal d'entrada estàndard i mostri per el canal de sortida estàndard els primers nombres parells menors a aquest enter. El programa pot tenir només un bucle *for*, no es permet fer servir estructures *if*. Pista: 0 és parell

1.4.6 Python: Mòduls

Finalment Python disposa d'un gran nombre de llibreries per abordar diferents tasques de desenvolupament, des de anàlisi estadístic fins a interpretació del llenguatge natural. Aquestes llibreries es poden instal·lar amb tecnologies com *pip* o *conda* i s'importen als vostres programes en forma de mòduls. Per exemple:

```
import random as rnd  #Importa mòdul i espai de noms
print (rnd.random()*10) #Invoca el mòdul en base al espai de noms
```

1.5 Exercicis Complementaris

1.5.1 Suma de matrius

En C++ les matrius es representen com a vectors de dues dimensions i la seva parametrització és com la dels vectors d'una dimensió.

Per exemple, una funció que sumi dues matrius d'enters es pot escriure així:

```
vector <vector<int> > suma(const vector <vector<int> > &m1,
const vector <vector<int> > &m2)
/* Pre: m1 i m2 tenen la mateixa dimensio */
/* Post: el resultat es la suma de m1 i m2 */
{
vector <vector<int> > s;
...
...
return s;
}
```

Podem escurçar l'escriptura usant la paraula clau `typedef`, que permet reanomenar el tipus `vector <vector<int> >` com:

```
typedef vector <vector<int> > Matriu;

Matriu suma(const Matriu &m1, const Matriu &m2)
/* Pre: m1 i m2 tenen la mateixa dimensio */
/* Post: el resultat es la suma de m1 i m2 */
{
  Matriu s;
  ...
  ...
  return s;
}
```

En el fitxer `suma_mat.cc` que us donem hi ha una implementació d'aquesta funció i també de les operacions de lectura i escriptura de matrius. En el fitxer `suma_mat.ent` hi ha un exemple de dades d'entrada. En el fitxer `suma_mat.sort` hi ha el resultat corresponent.

Com a exercici, modifiqueu la capçalera de l'operació per tal que la matriu suma s'obtingui com a paràmetre de sortida enlloc d'un resultat de la funció. Per tant la funció es convertirà en una acció doncs no retornarà res (`void`) i caldrà afegir un nou paràmetre de sortida. Comproveu que el programa resultant segueix sent correcte, o sigui que passi els jocs de prova donats redireccionant l'entrada i sortida i comparant la vostra sortida amb la del fitxer `suma_mat.sort` donat amb la comanda `diff`.

1.5.2 Producte de matrius

Desenvolupa i prova una funció que calculi el producte de dues matrius tant en C++ com en Python. Capçalera de la funció en C++:

```
typedef vector <vector<int> > Matriu;

Matriu producte(const Matriu &m1, const Matriu &m2)
/* Pre: nombre de columnes de m1 igual al nombre de files de m2 */
/* Post: el resultat es el producte de m1 i m2 */
{
  Matriu prod;
  ...
  ...
  return prod;
}
```

Capçalera de la funció en Python:

```
def producte(m1, m2):
/* Pre: nombre de columnes de m1 igual al nombre de files de m2 */
/* Post: el resultat es el producte de m1 i m2 */
{
  prod = []
  ...
  ...
  return prod
}
```

Exemple:

$$a = \begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \end{pmatrix} \qquad b = \begin{pmatrix} 3 \\ 3 \\ 3 \end{pmatrix} \qquad a \cdot b = \begin{pmatrix} 9 \\ 18 \end{pmatrix}$$