UNIVERSITE LIBRE DE BRUXELLES 9 mars 2009
Faculté des Sciences Appliquées

INFO-H-100 : Introduction à la programmation
Projet 1 : Blackjack

On vous demande de réaliser un programme `C++` permettant à un utilisateur de jouer contre l'ordinateur (la banque) à une version simplifiée du Blackjack.

Au départ, le joueur aura une somme de 10 jetons et une partie se déroulera comme suit :

1. L'ordinateur donne une carte au joueur.
2. Le joueur mise un certain nombre de jetons.
3. L'ordinateur donne une carte au joueur.
4. Tant que le joueur désire continuer et s'il a un score strictement inférieur à 21, l'ordinateur lui redonne une carte.
5. Si le joueur a un score strictement supérieur à 21, il perd sa mise et le programme va au point 11.
6. La banque tire des cartes jusqu'à atteindre un score supérieur ou égal à 17.
7. Si la banque a un score strictement supérieur à 21, le joueur gagne deux fois sa mise et le programme va au point 11.
8. Si la banque a le même score que le joueur, le joueur récupère sa mise et le programme va au point 11.
9. Si le joueur a un score strictement supérieur à celui de la banque, il gagne deux fois sa mise et le programme va au point 11.
10. Si la banque a un score strictement supérieur à celui du joueur, le joueur perd sa mise et le programme va au point 11.
11. S'il reste des jetons au joueur, le programme retourne au point 1.

Pour simplifier, le jeu de cartes est considéré comme infini. C'est à dire qu'à chaque tirage, chaque carte a la même probabilité d'être tirée. De même, le programme ne prendra pas en compte la couleur de la carte (coeur, pique, carreau, trèfle). Les cartes sont donc représentées à l'aide d'un entier allant de 1 à 13. Le 1 représente l'as, le 11 le valet, le 12 la dame et le 13 le roi. L'as vaut 1 ou 11 points. Le valet, la dame et le roi valent 10 points. Les autres cartes ont leur valeur nominale.

Le programme ne retiendra pas les cartes tirées mais le score du joueur et de la banque. On retiendra par contre à l'aide d'un booléen si l'on possède un as valant 11 dans sa main. Par défaut, l'as vaudra 11. En cas de dépassement du score maximum, si ce booléen a la valeur "vrai", 10 points seront retirés afin que l'as vaille 1 et le booléen prendra la valeur "faux".

On vous demande de découper votre programme en fonctions dont voici les prototypes :

- `void playBlackJack()` qui lance le jeu.
- `void printCard(int card)` qui affiche en français la carte. Par exemple, si l'on donne `2` en paramètre, cette fonction affichera "`un 2`". Si on lui donne `12`, elle affichera "`une dame`".
- `int playerBets(int cash)` qui lit au clavier la mise du joueur, juste après que celui-ci ait reçu sa première carte, et la renvoie si elle est comprise entre 1 et `cash` (sinon redemande au joueur de miser).
- `int playerPlays(int firstCard)` qui fait jouer le joueur juste après que celui-ci ait misé et renvoie son score.
- `int bankPlays()` qui fait jouer la banque et renvoie son score.
- `void updateTotal(int& total, bool& ace11, int card)` qui mettra à jour le score (`total`) en lui ajoutant la valeur de la carte (`card`). La variable `ace11` vaudra "vrai" s'il y a un as valant 11 dans la main.
- `void compareTotal(int playerTotal, int bankTotal, int& cash, int bet)` qui compare les totaux du joueur et de la banque et qui met à jour le nombre de jetons (`cash`) du joueur en fonction de sa mise (`bet`).

Ajoutez d'autres fonctions si cela est nécessaire.

Génération de nombre pseudo-aléatoires

Voici un exemple complet de code C++ affichant, tant que l'utilisateur appuie sur 'o', des entiers pseudo-aléatoires entre 0 et la constante `RAND_MAX`.

```
#include <iostream>
#include <cstdlib>
#include <ctime>

using namespace std;

int main()
{
  srand(time(NULL));
  bool continuer = true;
  char c;

  while(continuer)
  {
    cout << rand() << endl;
    cout << "Continuer ? (o/n)" << endl;
    cin >> c;
    if(c != 'o')
      continuer = false;
  }
  return 0;
}
```

L'instruction `srand(time(NULL));` initialise le générateur de nombres pseudo-aléatoires. Elle doit être exécutée une et une seule fois en début de programme.

La fonction `int rand()` renvoie pseudo-aléatoirement un entier entre 0 et `RAND_MAX`.

Consignes

Le projet est à réaliser **seul** et à remettre **au plus tard le mercredi 25 mars à 12h30** au UB4.131 (à côté de la salle informatique Socrate). Les seules librairies autorisées sont `iostream`, `cstdlib` et `ctime`, les deux dernières ne pouvant être utilisées que dans le cadre de la génération de nombres pseudo-aléatoires.

Vous devez rendre un rapport dactylographié comprenant :

1. une introduction présentant succinctement les différentes fonctions de votre programme sur une page maximum,
2. une capture d'écran présentant une exécution de votre programme et
3. votre code complet intelligemment commenté.

Les critères d'évaluation sont les suivants :

- Le code : **résout exactement le problème demandé sans ajout ni apport personnel**, qualité de l'algorithmique, syntaxe correcte (doit compiler sans erreur ni avertissement), parfaitement indenté, les variables bien nommées, le minimum de commentaires utiles, respect des conventions et des règles de bonne pratique vues au cours, aux séances d'exercices et résumées sur la page web des projets (`http://cs.ulb.ac.be/public/teaching/infoh100/projets`)
- Le rapport : qualité du contenu, présentation, français correct.
- Le respect des consignes.

Les élèves assistants sont disponibles tous les midis dans la salle Socrate afin de répondre à vos questions.

INFO-H-100 : Règles de bonne pratique

Ces règles ont été déterminées dans un but pédagogique et/ou de bonne partique générale en programmation.

Structure

- Utiliser une découpe intelligente en fonctions. Chaque fonction doit réaliser **une** tâche clairement identifiée.
- Ne pas faire de fonctions de plus de 25 lignes sauf dans des cas exceptionnels à justifier.
- Eviter la redondance dans votre code (copier coller). Si cela arrive, c'est qu'il manque soit une fonction, soit une boucle, soit que des tests conditionnels peuvent être regroupés.
- Ne pas utiliser de variable globale.
- Déclarer les variables en début de bloc.
- Utiliser adéquatement des constantes globales (par exemple, la taille des tableaux sera généralement une constante globale).
- Veiller à ce que tous les paramètres et variables d'une fonction soient utilisés dans cette fonction.
- Les fonctions qui prennent des paramètres par référence doivent avoir le type de retour `void` sauf dans les cas particuliers où :
  - on veut retourner un code d'erreur entier ou
  - on veut retourner un booléen pour dire que l'opération s'est bien ou mal passée.

Contrôle de flux

- Ne pas utiliser l'instruction `goto`.
- Ne pas utiliser l'instruction `continue`.
- Ne pas utiliser l'instruction `break` sauf dans les `switch case`.
- Ne faire qu'un et un seul `return` par fonction (sauf celles de type `void`) à la dernière ligne de cette fonction.

Style

- Ne pas utiliser le type `int` pour les booléens. Par exemple, les formes

```
bool ok = 0;
```

ou

```
int ok = 1;
while(ok==1)
{
   <instruction>
}
```

sont à proscrire.

- Ne pas utiliser explicitement de pointeurs.
- Utiliser la forme raccourcie

```
if(isLeapYear(2008))
```

plutôt que la forme équivalente

```
if(isLeapYear(2008)==true)
```

- Utiliser la forme

```
return <expression booléenne>
```

plutôt que la forme équivalente

```
bool res;
if(<expression booléenne>)
{
  res = true;
}
else
{
  res = false;
}
return res;
```

- Ne pas exécuter plusieurs fois une fonction alors qu'une exécution suffit. Par exemple, ne pas faire

```
for(int currentDay=0; currentDay<daysInMonth(12); currentDay++)
```

mais plutôt

```
int daysInDecember = daysInMonth(12);
for(int currentDay=0; currentDay<daysInDecember; currentDay++)
```

Ici, la fonction `daysInMonth(12)` renverra toujours la même valeur. Mettre l'appel avant la boucle signifie :

  - que le contenu de la variable `daysInDecember` ne changera pas durant l'exécution de cette fonction
  - deuxièmement, cette fonction sera appelée une et une seule fois et non à chaque tour de boucle.

Gestion des erreurs

- Gérer les erreurs aux niveau du programme : toutes les entrées des utilisateurs doivent être vérifiées et traitées le cas échéant.
- Au niveau des fonctions : soit gérer les erreurs, soit documenter le domaine des paramètres. Dans ce dernier cas, si une valeur donnée par paramètre n'est pas dans le domaine, le comportement de la fonction est indéterminé. Exemple :

```
int daysInMonth(int month, int year)
{
   int nbDays = 31;
   if(month==4 or month==6 or month==9 or month==11)
      nbDays = 30;
   else if(month==2)
      if(isLeapYear(year))
         nbDays = 29;
      else
         nbDays = 28;
   return nbDays;
}
```

  Cette fonction attend un `month` entre 1 et 12. Le résultat pour les nombres inférieurs à 1 et supérieurs à 12 n'a pas de sens.

- La fonction `int main()` doit retourner un entier. En général, cette fonction renvoie 0 pour signifier que le traitement s'est effectué correctement.

INFO-H-100 : Conventions

Conventions de nommage

- Nommer les variables et les constantes selon leur contenu et les fonctions selon leur tâche. Exemples : `int age` et `bool isLeapYear(…)`.
- Ne pas utiliser d'accents dans les noms.
- Choisir une langue (français/anglais) et l'utiliser de bout en bout (sauf eventuellement pour les commentaires).
- Choisir une convention de nommage tel que l'on puisse différencier les constantes et types définis par rapport aux fonctions, variables et paramètres. Ci-dessous, un exemple de convention de nommage.

Nommage des fonctions, variables et paramètres

- Notation lowerCamelCase [http://en.wikipedia.org/wiki/CamelCase].
- Le nom commence par une minuscule.
- Chaque nouveau mot du nom est collé au précédent et commence par majuscule.
- Exemple : `isLeapYear(…)`

Nommage des types définis ("typedef")

- Notation CamelCase [http://en.wikipedia.org/wiki/CamelCase].
- Le nom commence par une majuscule.
- Chaque nouveau mot du nom est collé au précédent et commence par majuscule.
- Exemple : `SquareMatrix`

Nom des constantes

- Toutes les lettres du nom sont en majuscule.
- Les mots du nom sont séparés par un underscore (_).
- Exemple : `SIZE_MAX`

Conventions de mise en forme du code

- Les accolades doivent être au même niveau vertical que les instructions du bloc englobant.
- Il faut respecter la même indentation de bout en bout pour les instructions des blocs englobés, par exemple quatre espaces.
- Exemple :

```
int fonction()
{
   if(<condition>)
   {
      for(int i=0;i<N;i++)
```

```
      {
         <instructions>
      }
   }
   else if(<condition>)
   {
      <instructions>
   }
   else
   {
      <instructions>
   }
   return <int>;
}
```

- Si un bloc est composé d'une seule instruction, vous pouvez écrire soit :
  - ```
    if(<condition>) <instruction>
    ```
  - ```
    if(<condition>)
       <instruction>
    ```
  - ```
    if(<condition>)
    {
       <instruction>
    }
    ```
- Si le nombre de caractères d'une ligne dépasse la largeur d'une page, indenter proprement. Les retours à la ligne sont considérés comme des espaces par le compilateur. Par exemple :
  - ```
    if(<condition>
       and <condition>
       and <condition>
       and <condition>
       and <condition>
       and <condition>)
    ```
- En conclusion, vous devez avoir une convention de mise en forme et la respecter pour tout votre programme. Si dans certains cas limités, respecter ces règles d'indentation conduisent à une lecture plus difficile, vous pouvez utiliser un autre type d'indentation.
- Votre programme doit pouvoir s'imprimer proprement et lisiblement sur des pages A4.