Camp de jour Folie Technique
Été 2019

Informatique Avancée

Thématique 14-17 ans

POLYTECHNIQUE MONTRÉAL
Par Fermion & Béryllium

Table des matières

Introduction 2
Horaire de la semaine 2
Hello World! 3
MasterMind 7
SuperClassBros 9

Introduction

La semaine *Informatique Avancé* introduira aux jeunes le langage C++ qui est un langage fondamental de l'informatique. Ils apprenderont comment manipuler des variables, des fonctions ainsi que le système de classe offert par C++. Ils apprenderont comment toutes les notions abordés au cours de la semaine peuvent être utilisés simultanement pour créer des programmes intéractifs et amusants.

Il est pris pour aquis que les jeunes inscrits ont déjà fait de la programmation, peu importe le langage[1], mais sans toutefois avoir fait de la programmation de haut-niveau. Par exemple, un jeune ayant participé à une semaine *Informatique* auparavant serait parfaitement en mesure de participer à cette semaine.

Horaire de la semaine

<table>
<tr><th></th><th>Lundi</th><th>Mardi</th><th>Mercredi</th><th>Jeudi</th><th>Vendredi</th></tr>
<tr><td>9:00</td><td rowspan="2">Spectacle du lundi</td><td rowspan="3">Super Class Bros</td><td rowspan="3">Machine Learning</td><td rowspan="3">Jeux Vidéo</td><td rowspan="3">MasterMind</td></tr>
<tr><td>9:30</td></tr>
<tr><td>10:00</td><td rowspan="2">Introduction</td></tr>
<tr><td>10:30</td><td></td><td></td><td></td><td></td></tr>
<tr><td>11:00</td><td rowspan="2">Diner</td><td rowspan="2">Diner</td><td rowspan="2">Diner</td><td rowspan="2">Diner</td><td rowspan="2">Diner</td></tr>
<tr><td>11:30</td></tr>
<tr><td>12:00</td><td rowspan="2">Activité à l'extérieur</td><td rowspan="2">Activité à l'extérieur</td><td rowspan="2">Activité à l'extérieur</td><td rowspan="2">Activité à l'extérieur</td><td rowspan="2">Activité à l'extérieur</td></tr>
<tr><td>12:30</td></tr>
<tr><td>13:00</td><td rowspan="3">Intro C++</td><td rowspan="2">Langages program-mation</td><td rowspan="5">MasterMind</td><td rowspan="5">Unity3D</td><td rowspan="5">kermesse</td></tr>
<tr><td>13:30</td></tr>
<tr><td>14:00</td><td rowspan="4">MasterMind</td><td rowspan="4">MasterMind</td></tr>
<tr><td>14:30</td></tr>
<tr><td>15:00</td></tr>
<tr><td>15:30</td></tr>
<tr><td>16:00</td></tr>
</table>

[1] Excepté le «langage» HTML qui n'est pas un langage de programmation.

Hello World!

Sommaire

L'activité *Hello World* servira d'activité d'introduction à la programmation en C++ ainsi que de vérifier le niveau de connaissance des jeunes. Elle introduit des notions de base qui serviront à l'actité *MasterMind*. Les jeunes devront faire un programme intéractif qui leur demande des informations sur eux (*exemple: nom, âge, etc.)* et devra être capable de bien fonctionner malgré des erreurs que l'utilisateur pourrait faire.

Durée approximative	**Coût approximatif par jeune**
1h	0.00$
Niveau de difficulté (1 à 5)	**Groupe d'âge**
2	14-17 ans
Matériel	**Notions abordées**
aucun	- Variables - Fonctions - Boucles

Théorie

En programmation, une **variable** est simplement une espace mémoire alloué qui est associé à un symbole donné par le programmeur. Une variable, en C++, n'est pas obligée de contenir une valeur, mais son type doit toujours être précisé! La valeur de la variable peut être *librement* modifiée à tout moment dans le programme, d'où le nom *variable*.

Tel que mentionné, une variable à ce qu'on appelle un **type** qui est tout simplement la nature de la valeur contenue à l'intérieur de la variable. Les nombres entiers et décimaux sont de type *int* et *float*, les caractères simples sont des *char* tandis que les chaines de caractères sont des *string* et il y a le type *boolean* qui peut seulement prendre 0 ou 1 comme valeur. La nature du type influence le rôle que la variable peut avoir dans le programme; les *string* ne sont pas utilisés pour compter, mais bien pour stocker des phrases ou autres chaines de caratères.

Les **boucles**, en programmation, sont utilisées pour effectuer une certaine tâche plusieurs fois sans devoir à copier-coller des sections de code et ainsi le programme inutilement. De plus, le nombre de fois qu'une tâche doit être répéter peut être inconnu lors de la compilation ou même varier. En C++, les deux boucles que l'on peut utilisées sont les boucles *for* et *while*; habituellment, la boucle *for* est employée lorsque le nombre d'itération est «connu» tandis que la boucle *while* est utilisée lorsque le nombre d'itération dépend d'une certaine condition.

Peu importe le type de boucle utilisée, le fonctionnement reste le même: avant d'entrer à l'intérieur de la boucle, une certaine condition -préciser par le programmeur- détermine si oui ou non la boucle peut se répéter. Dans le cas oùla condition est respectée, on entre dans le corps de la boucle et le code s'ytrouvant se fait exécuter. Lorsque l'exécution du coprs est fait, on vérifie si la condition est toujours respectée; si c'est le cas on entre à nouveau dans la boucle, sinon on continue dans le programme en ignorant le corps de la boucle.

Voici un exemple de chaque type de boucle:

```
// Boucle de type FOR
for (int i = 0; i < 10; i++) {
    // Corps de la boucle
}

// Boucle de type WHILE
int i = 0;
while(i < 10) {
    // Corps de la boucle
    i++;
}
```

Figure 1: Démonstration de boucles `FOR` et `WHILE`

Les **fonctions**, en programmation, sont utilisées pour exécuter une certaine procédure en appelant la fonction aux endroits désirés; tout comme les boucles, l'utilisation de fonctions permet d'écrire du code à un seul endroit plutôt que de recopier la procédure à chaque fois qu'il faut l'exécuter. Une fonction peut avoir, ou non, des arguments qui sont tout simplement les variables qui seront utilisées par la fonctions, exactement comme une fonction en mathématique. Lorsqu'on programme en C++, les fonctions ne peuvent retourner qu'une seule valeur et le type de la valeur retournée doit être le même que le type de la fonction. Donc si la fonction retourne un nombre entier, son type devrait être *int*. Une fonction peut toutefois rien retourner dans le programme et son type serait alors *void*. Lorsqu'une fonction est utilisée à l'intérieur d'un programme, disons que notre fonction se nomme *print*, on écrit simplement le nom de la fonction suivie des arguments qui seront utilisés entre parenthèses. Voici un exemple d'une fonction:

```
// Exemple d'une définition
float polynom(float x, float a, float b, float c) {
    return a*x*x + b*x + c;
}

// Exemple d'utilisation
float x = 0.0f;
float y;
while (x <= 10.0) {
    y = polynom(x, 2, 3, 5);
    printf("Quadratique: %d\n", y);
    x = x + 0.1;
}
```

Figure 2: Exemple de fonction

Toutes les notions sont abordées à l'intérieur de la présentation *Introdution à C++* avec des exemples plus détailés.

Manipulation

Commencer l'activité avec la présentation *Introduction à C++* qui introduit des notions de bases du langages. Pour chaque nouvelle notion abordée, assurez-vous que les jeunes comprennent bien avant de passer à la prochaine. La présentation parlera des bons comportements à avoir lorsqu'on programme, des variables, des types, des boucles ainsi que des fonctions.

Lorsque la présentation est terminé, créeer un nouveau projet avec les jeunes nommé *Hello World!* et expliquer leur le but de l'activité. Continuier ensuite par leur montrer comment utiliser les fonctions *cout* et *cin* du langage pour qu'ils sachent comment créer une intéraction entre l'utilisateur et la console (le programme).

Les jeunes devront maintenant essayer de programmer le code qui fait les tâches suivantes:

1. Message d'introdution
2. Demande à l'utilisateur son nom
3. Demande à l'utilisateur son age
4. Message personnalisé avec le nom et l'age
5. Pécise à l'utilisatuer s'il est majeur ou non

Le tout devra être à l'intérieur d'une boucle et la console demande à la fin si l'utilisateur veut quitter le programme; si il répond oui le programme se termine sinon il se répère du début.

Présentation suggérée

Il est fortement recommendé de faire l'activité avec la présentation *Introduction à C++* puisque toutes les notions utilisées y sont présentées.

Code de référence

```cpp
int main() {
    int age;
    string nom, choix;

    while (true) {
        cout << "Bonjour! Comment t'appelles-tu?\n" << endl;
        cin >> nom;

        cout << "\nMaintenant que je sais que tu t'appelles " << nom <<
                ", j'aimerais connâitre ton age\n";
        cin >> age;

        cout << "Alors, ton nom est " << nom << " et tu as " << age << "ans.\n";

        if (age >= 18) { cout << "Tu es donc majeur.\n";}
        else { cout << "Tu es donc mineur.\n";}

        cout << "Veux-tu quitter? (oui ou non)\n";
        cin >> choix;

        if (choix[0] == 'o' || choix[0] == 'O') {
            cout << "Au revoir!\n";
            break;
        } else {
            cout << "Parfait, on va recommencer!\n";
        }
    }
    return 0;
}
```

Figure 3: Exemple d'un code pour HelloWorld!

MasterMind

Sommaire

Cette activité est l'activité principale de la semaine durant laquelle les jeunes devront reproduire le jeu *Mastermind* à l'aide d'un programme qu'ils écriront. Chaque jeune devra écrire le code du jeu, mais ils peuvent tout de même communiquer entre eux pour s'aider. L'activité se déroulera durant toute la semaine pour construire itérativement le code; d'autres activités/présentations introduiront de nouveaux concepts qui seront par la suite utilisés pour améliorer leur programme. Le but de l'activité est de familiariser les jeunes avec les concepts vus durant la semaine ainsi que d'améliorer leur compétence à abstraire et segmenter un problème pour le reproduire à l'aide d'un programme informatique.

Durée approximative	**Coût approximatif par jeune**
12h	0.00$
Niveau de difficulté (1 à 5)	**Groupe d'âge**
4	14-17 ans
Matériel	**Notions abordées**
- a	- a

Théorie

La théorie de cette activité provient des autres présentations faites durant la semaine; cette activité n'introduit aucune nouvelle notion aux jeunes, mais les fusionne en une seule activité. Les jeunes apprendront donc à utiliser tous les concepts vus durant la semaine et comment les utiliser dans un seul programme.

Déroulement

La première étape de l'activité est de présenter aux jeunes le vrai jeu *Mastermind* et de leur expliquer comment on y joue. Les règlements se trouvent plus loin dans cette section. Assurez-vous que les jeunes ont bien compris les règlements ainsi que les étapes d'une partie de *Mastermind*, car ils auront beaucoup de difficulté par la suite sinon. Essayer ensuite de leur faire décortiquer les mécanismes du jeu et comment les joueurs interagissent lors d'une partie; n'hésiter pas à les aider pour compléter cette partie puisqu'elle est cruciale pour le reste de l'activité.

Lorsque le jeu est bien compris par tous, les jeunes peuvent maintenant commencer à faire le code. La première étape à réaliser est de créer le projet avec les jeunes. Nommez le projet *Mastermind* et le premier fichier *Source.cpp*. Créer ensuite une classe *MasterMindGame* qui ajoutera au projet deux fichiers portant le même nom avec les extensions *.cpp* et *.h*.

Les jeunes programmeront le code entièrement par eux-même; aucun code troué ne leur sera fourni; le programme sera donc construit itérativement en implémentant les conceptes abordés par les autres activités et modifiant le code existant en conséquence des ajouts apportés. L'animateur doit toutfois être capable de les aider au besoin. Votre rôle sera donc d'introduire les notions utiles ainsi que d'aider les jeunes à les implémenter dans leur code.

Étapes du programme

Les

1^{e} itération interaction joueur-console

Le première étape du code serait d'implementer une première version du systeme permettant l'intéraction joueur-console. Ce systeme est simplement une boucle qui demande au joueur d'entrer son choix, le stock dans une certaine variable et puis l'affiche à la console. À chaque itération, le nombre d'essaie actuel du joueur ainsi que le nombre d'essaie maximum doivent être affichés. La condition d'arrêt de la boucle est simplement lorsque la valeur de l'essaie actuelle est plus grande que la valeur maximale. Demander d'abord aux jeunes d'utiliser la fonction `cin` et montrer leur pourquoi il serait préférable d'utiliser la fonction `getline()`.

Petit rappel: La fonction *cin* sépare le input à chaque espacement fait et stock les mots dans différents *buffer* ce qui cause certains problème pour le jeu que l'on veut faire.

2^{e} itération initialisation de la classe

La seconde étape consiste à implémenter une première version de la classe *MasterMindGame* et d'utiliser ses méthodes pour automatiser l'attribution des valeurs à certaines variables. Dans le *header file* de la classe, il faut ajouter les attributs *codeCache, codeLongueur, essaie, essaieMax* comme attributs *privés* et les méthodes *getters GetCodeLongueur, GetEssaie et GetEssaieMax* ainsi que la méthode *Reset* comme méthodes publiques. Repréciser aux jeunes la différence entre les propriétés privé et publique et pourquoi on choisi de mettre, par exemple, l'attribut *codeCache* privé.

3^{e} itération choix du joueur

La troisième itération consiste à implémenter une fonction (à l'intérieur du fichier *Source.cpp*) pour s'assurer que le choix du jouer est valide. La fonction devra afficher le nombre d'essaie du joueur et lui demander son choix. Le choix sera directement utilisé par la méthode *GetStatusChoix* de la classe *MasterMindGame*; la méthode devra retourné le status *OK* si le choix est valide ou *NOT_SAME_LENGTH* ou *NOT_VALID_CARACTER* selon le cas. La méthode *GetStatusChoix* devra donc être capable de déterminer si la longueur du choix fait est la même que celle du code caché et si le choix contient uniquement des caractères valides. Si le choix n'est pas valide, un message lui précisant son erreur devrait apparaître à la console pour qu'il puisse faire un meilleur choix à son prochain essai. Presque l'entierté de la fonction se trouve à l'intérieur d'une boucle dont l'unique manière de sortir est lorsque le choix fait est valide, ou autrement dit, lorsque le statu retourné par la méthode est *OK*. Le choix fait est alors retourné par la fonction.

4^{e} itération

La quatrième itération du code implémente une manière de mettre à jour le score du joueur selon le choix qu'il vient de faire. Tout le code qui sera ajouté par cette étape se retrouve presqu'entièrement à l'intérieur de la classe *MasterMindGame*. Pour ce faire il faut ajouter la méthode *UpdateScore* qui accepte comme argument le choix du choix sous forme de *string* et retourne simplement le score du joueur. Pour déterminer le score du joueur, il suffit simplement de faire un boucle chaque caractère du choix et à l'intérieur de cette boucle, de faire une seconde boucle sur chaque caractère du code caché. Pour chaque pair de caractère, il est facile de déterminer s'il s'agit d'un *blanc* ou d'un *noir* et ainsi mettre à jour le score. C'est également à l'intérieur de cette méthode que l'on vérifie si le joueur à gagné. Ceci est la manière *naïve* et comporte un petit *bug* qui n'est aps nuissible; vous pouvez demander aux jeunes d'essayer de régler ce problème lorsqu'ils auront fini.

SuperClassBros

Sommaire

Durant cette activité, les jeunes devront reconstruire la hiérarchie de classes du jeu *Super Mario Bros* à l'aide d'une image qui sera présentée à l'écran. L'activité peut se faire en équipe, ou seul si un jeune le souhaite, et sa durée est approximativement 30 minutes; le temps alloué peut facilement être modifié pour s'adapter à l'intérêt des jeunes.

Le but de cette activité est de familiariser les jeunes avec les concepts d'héritage, de méthodes et d'attributs d'une classe et d'interaction entre classes. Cette activité développera l'aptitude des jeunes à reconnaître ce que peut être une classe, à visualiser la hiérarchie d'un groupe de classe -en remarquant la similitude entre les classes- ainsi que de déterminer quels méthodes et attributs une classe pourrait posséder.

Durée approximative	**Coût approximatif par jeune**
1h30min	0$

Niveau de difficulté (1 à 5)	**Groupe d'âge**
3	14-17 ans

Matériel	**Notions abordées**
- Présentation *SuperClassBros* - Papier - Crayon	- Attibuts - Méthodes - Héritage - Class

Théorie

En informatique, une classe est une structure de donnée permettant l'encapsulation de fonctions et de variables à l'intérieur d'un objet et est utilisée dans la programmation orientée objet qui favorise l'interaction entre plusieurs classes. Une classe peut également avoir des classes «filles» qui héritent de toutes les méthodes et attributs de leur «mères» et doivent en posséder davantage; les classes filles sont moins générales que leur classe mère et leurs méthodes sont plus précises.

Par exemple, soit les classes *Personne*, *Élève*, *Personnel*, *Enseignant* tel que la classe *Personne* a les attributs suivants: *prénom*, *nom*, *date_de_naissance*. Cette classe pourrait être la classe mère des classes *Élève* et *Personnel*; on ajouterait alors à la classe *Élève* d'autres attributs tel que *programme*, *moyenne*, etc. et à la classe *Personnel* les attributs *ancienneté*, *salaire*, etc. Puis, la classe *Enseignant* serait la classe fille de *Personnel* et on lui ajouterait les attributs *domaine_de_recherche*, *cours*, etc.

Tel que mentionné, une classe est une strucutre de données qui permet d'encapsuler des variables ainsi que des fonctions; les variables et fonctions attachées à une classe sont respectivement nommées **attributs** et **méthodes**. Si une fonction peut être vue comme une procédure qui est executé lorsqu'elle est appelé, alors une classe est une *blueprint* qui construit un ceertain objet avec certains attributs et que l'on peut manipuler par la suite à l'aide des méthodes de la classe. Plusieurs objets d'une même classe peuvent re instanciés en même temps et chaque objet se comportera indépendamment des autres. Par exemple, pour un jeu vidéo, une classe *Joueur* peut être utiliser autant de fois qu'il y a de personne jouant au jeu et chacune

de ces personnes contrôlera une entité différente. Les attributs et méthodes d'une classe peuvent soient être *publique* ou *privée*; un attribut/fonction *publique* peut directement être utilisé ou modifié à *l'exterieur* de la classe en utilisant le «.» tandis qu'un attribut/fonction *privé* n'est accessible ou modifiable qu'à l'intérieur de la classe. En reprenant l'exemple de la classe *Joueur*, une attribut pourrait être la vie du joueur et un tel attribut ne devrait pas être modifié hors de la classe, mais uniquement en utilisant des méthodes publiques de la classe. On protège donc certains attributs/méthodes en les *taggant* comme privés.

Matériel

Simplement du papiers et des crayons pour que les jeunes puissent écrire leur hiérarchie plus facilement. Oridnateur également nécéssaire.

Déroulement

Débutez l'activité avec la théorie sur les classes présente dans la présentation *SuperClassBros*. Assurez-vous que les jeunes comprennent bien ce qu'est une classe, la différences entre ce qu'est un attributs et une méthodes ainsi que ce qu'est une hiérarchie. Lorsque les jeunes sont prêts, commencez l'activité.

Présentez à l'écran l'image du jeu *Super Mario bros* et expliquez interactivement avec les jeunes les éléments présents sur l'image ainsi qu'une petite explication du jeu, si nécéssaire. Puis, passez à la présentation qui contient les mots clés et laissez les jeunes faire la hiérarchie! Votre rôle n'est pas de leur donner les réponses, mais plutôt de les guider si jamais ils sont perdus. Précisez également aux jeunes qu'il n'y a pas de mauvaises réponces, mais que certaines hiérarchies vont simplifier la construction du jeu et que d'autres vont compliquer les choses.

Lorsque les jeunes ont terminé, construiser avec eux la hiérarchie en utilisant leur réponses et modifier-les aux besoins. Tel que mentionné il n'y a pas vraiment de mauvaise réponse, mais certaine compliqueront la hiérarchie et il serait préférable de ne pas les utiliser. Lorsque la hiérarchie est complétée, et si les jeunes semble toujours intéressés, ajouter des nouveaux éléments au jeu et demander aux jeunes à quel endroit dans la hiérarchie chaque élément devrait se trouver. Par exemple, si vous ajouter le personnage *Princess Fermion*, on devrait retrouver cet élément sous la classe *Entité* et selon le cas, *Enemie* ou *NPC/Joueur*. Cette activité devrait facilement montrer aux jeunes la force d'utiliser une hiérarchie de classe et à quel pointe cela facilite l'ajout de nouveaux éléments.

Présentation suggérée

Seule la présentation *SuperClassBros* est nécéssaire; elle contient la théorie sur les classes ainsi que l'activité elle-même.