תכנות מונחה עצמים בשפת C++

תרגול מספר 1

אוהד ברזילי
אוניברסיטת תל אביב

חדש חדיש ומחודש

תוספות של C++ לשפת C

Hello World

```
#include<iostream>

using namespace std;

int main()
{
	cout << "Hello World\n";
}
```

Scopes

- בלוק חדש נוצר ע"י: { בלוק חדש }
- ניתן להגדיר משתנים חדשים איפה שרוצים (כמעט)
- טווח הכרה וטווח קיום של משתנים
- כל משתנה יוכר מהנקודה שבה הוגדר ועד לסיום הבלוק *הפנימי* ביותר שבו הוגדר
- משתנה יקרא *גלובלי* אם הוא נמצא מחוץ לכל פונקציה, מחלקה, מרחב שמות או בלוק
- הגדרות של משתנה בבלוק פנימי מסתירות הגדרות עוטפות והגדרות גלובליות

Scopes

```
int x ; // global x

void f ()
{
   int x ; // local x hides global x
   x = 1 ; // assign to local x
   {
       int x ; // hides first local x
       x = 2 ; // assign to second local x
       ::x = 2 ; // assign to global x
   }
   x = 3 ; // assign to first local x
}

int * p = &x ; // take address of global x
```

הגדירי משתנים מקומיים ב scope קטן ככל הניתן

- למשל, במשפט אתחול של for:

```
for (int i=0; i<10 ; i++) {...}
```

- למשל, בתוך תנאי ה if:

```
if (double d = prim (true))
{
  left /= d ;
  break ;
}
```

אחרת...

```
double d ;
// ...
d2 = d ; // oops!
// ...
if (d = prim (true ))
{
  left /=d;
  break;
}
// ...
d = 2.0; // two unrelated uses of d
```

העמסת פונקציות

- נממש את max המקבלת שני ארגומנטים ומחזירה את הגדול מביניהם
- בשפת C יש לממש max נפרדת (בשם נפרד) עבור כל טיפוס:

```
double max_double(double x, double y)
long max_long(long x, long y)
```

- העמסת פונקציה מאפשרת הגדרת <u>פונקציות שונות</u> <u>בשם זהה</u> ובלבד שלא תהיה להן אותה חתימה (טיפוס ומספר ארגומנטים)

העמסת פונקציות

- בשפת C++ ניתן להגדיר:

```
double max(double x, double y)
long max(long x, long y)
```

- איזו מהפונקציות תופעל במקרים הבאים:

```
max(1L , 1L);        // max(long,long)
max(1.0 , 1.0);      // max(double,double)
max(1L , 1.0);       // error ! Ambiguous
max(1 , 1);          // error ! Ambiguous
```

העמסת פונקציות

המהדר מנסה למצוא את הגרסה המתאימה ביותר עבור כל קריאה לפונקציה על פי טיפוסי הארגומנטים של הקריאה לפי הסדר הבא:

1. **התאמה מדויקת** (או המרה טריויאלית)
   - מערך למצביע, פונקציה למצביע לפונקציה, הוספת const
2. **קידום** (promotion)
   - bool,char,short ל- int וגרסאות ה unsigned
   - float ל- double , double ל- long double

העמסת פונקציות

3. **ביצוע המרות סטנדרטיות**
   - int ל- double ולהיפך
   - upcasting
   - מצביע לטיפוס ל- * void
   - int ל- unsigned int
4. **המרות שהוגדרו ע"י המשתמש**
5. **פונקציות שהוגדרו ע"י ...** (שלוש נקודות)

אם לא נמצאת התאמה או שנמצאות שתי התאמות "באותה רמה" או שפרמטרים שונים מתאימים לפונקציות שונות המהדר מודיע על אי בהירות (ambiguity)

ערכי ברירת מחדל לארגומנטים - מוטיבציה

- הפונקציה `int compute(int x,int base)` מבצעת חישוב כלשהו על הארגומנט `x` בבסיס ספירה `base`
- בדרך כלל קוראים לה בבסיס עשרוני והמשתמשים מעוניינים לחסוך את הצורך להעביר לה 2 ארגומנטים
- בסיס הספירה יועבר רק אם הוא שונה מ-10
- לדוגמא כאשר המתכנת כותב `compute(a)` כאילו נכתב: `compute(a,10)`

ערכי ברירת מחדל לארגומנטים (default arguments)

- נצהיר על הפונקציה כך:

```
int compute(int x , int base=10);
```

- אם מימוש הפונקציה (definition) מופרד מההצהרה (הכרזה, declaration) על הפונקציה ערכי ברירת המחדל יופיעו רק בהצהרה
- ערכי ברירת מחדל ניתנים לכל הארגומנטים החל ממקום מסוים והלאה

ערכי ברירת מחדל לארגומנטים (default arguments)

- העמסה בשילוב עם ערכי ברירת מחדל עלולה ליצור ambiguity, למשל בהגדרת הפונקציות:

```
int compute(int x , int base=10);
int compute(int x);
```

- איזו פונקציה תיקרא ?
  - compute(4,16) // הפונקציה הראשונה
  - compute(4) // שגיאת קומפילציה

העברה by value
והעברה by reference

- בשפת C נתונים מועברים לפונקציה ומוחזרים ומפונקציה ע"י העתקתם (by value)
- גם מצביעים מועתקים!
- לא ניתן לשנות נתון המועבר לפונקציה
- ב ++C ניתן לבחור האם רוצים להעביר העתק של הנתון או את הנתון עצמו (by reference)

C-Style swap

```
void swap(int x , int y)
{
   int temp = x;
   x = y;
   y = temp;
}

int main()
{
   int x=3, y=4;
   printf("before swap: x=%d y=%d\n",x,y);
   swap(x,y);
   printf("after swap: x=%d y=%d\n",x,y);
}
```

מה יודפס ?

C-Style swap

```
void swap(int *x , int *y)
{
   int temp = *x;
   *x = *y;
   *y = temp;
}

int main()
{
   int x=3, y=4;
   printf("before swap: x=%d y=%d\n",x,y);
   swap(&x,&y);
   printf("after swap: x=%d y=%d\n",x,y);
}
```

מה יודפס ?

C++ Style swap

```
void swap(int& x , int& y)
{
   int temp = x;
   x = y;
   y = temp;
}

int main()
{
   int x=3, y=4;
   cout << "before swap: x=" << x << " y=" << y << endl;
   swap(x,y);
   cout << "before swap: x=" << x << " y=" << y << endl;
}
```

מה יודפס ?

by reference החזרה

```
struct point {int x,y;};

int &getX(point *p) {return p->x;}

int main()
{
  point p = {3,4};
  getX(&p) = 5;
  cout << "p.x = " << p.x << endl;
}
```

?מה יודפס

`const` by reference

- פונקציות אשר מקבלות משתנים by reference משיקולי נוחות יציינו כי אינן משנות ערכים אלו ע"י הוספת const

```
void lo_naganu(const int &x) ;
```

- פונקציות המחזירות ערכים by reference משיקולי נוחות יגנו על ערכים אלו ע"י הוספת const לטיפוס הערך המוחזר

```
const int &alti_gabo(point *p);
```

`const` with pointers

- כאשר מצביע לטיפוס כלשהו מוגדר כקבוע (`const`) האם אסור לשנות את *הנתון* או את *המצביע*?
  - כאשר ה const מופיע <u>בתחילת</u> ההגדרה – *הנתון* המוצבע הוא קבוע
  - כדי לציין *שהמצביע עצמו* קבוע נחליף את ה - * שבהגדרת המצביע ל- `*const`

אפשר דוגמא?

```
void f1 (char  *p)
{
   char s[] = "Gorm";

   const char * pc = s ;     // pointer to constant
   pc [3 ] = ´g ´;           // error: pc points to constant
   pc = p ;                  // ok

   char *const cp = s ;      // constant pointer
   cp [3 ] = ´a ´;           // ok
   cp =p;                    // error: cp is constant

   const char *const cpc = s ; // const pointer to const
   cpc [3 ] = ´a ´;  // error: cpc points to constant
   cpc = p ;         // error: cpc is constant
}
```

קלט \ פלט

- פלט בעזרת ה"פקודה" cout:
  - `cout << “hello world\n”`
  - `int i = 10;`
    `cout << “i = ” << i << endl;`

- קלט בעזרת ה"פקודה" cin:
  - `int i;`
    `cin >> i;`

- `#include<iostream>`

קלט \ פלט

- אין צורך בטיפוסים – איך?
- אין צורך בהעברת כתובתם של משתני הקלט – איך?
- מייתר את השימוש בספרייה `stdio.h`

מה חדש?

- הקצאת זכרון דינאמית תעשה בעזרת האופרטור new (מילה שמורה) המחליף את malloc
- new מקבלת כארגומנט שם טיפוס, מערך או פונקציית אתחול (בנאי):
  - `int *i = new int;          // הקצאת int`
  - `int *i = new int[10]; // הקצאת מערך של 10 int`
  - `int *i = new int(); // הקצאת int מאותחל לאפס//`

לשחרר את ווילי

- שחרור זכרון שהוקצה דינאמית יעשה בעזרת האופרטור `delete` (מילה שמורה) המחליף את `free`
- `delete` מקבלת כארגומנט מצביע שהוקצה ע"י `new` ומשחררת את הזכרון שהוקצה
  - `delete i;` // שחרור נתון בודד
  - `delete []arr;` // שחרור מערך

עבודה עם מערכים

- תוכנית ++C לא אמורה להכיל מערכים כלל
- לשם כך הומצא מבנה הנתונים `vector<T>` אשר חוסך למתכנת את ההתמודדות עם ניהול זכרון (ע"י פונקציות כדוגמת `realloc()`) הטומן בחובו באגים לעתיד
- מחרוזות (מערכים מטיפוס * char) יוחלפו בטיפוס הסטנדרטי `string`

הערות

- יותר משורה אחת: /* ... */
- עד סוף השורה: //
- בשפת JAVA הוסיפו סוג נוסף - הערות לצורכי תיעוד: /** ... */

הפרה-פרוססור עשה את שלו -
הפרה-פרוססור יכול ללכת

- הגדרת קבועים:

```
const int MAX = 100;
```

- הגדרת סדרת קבועים:

```
enum traffic_lights { RED=1 , YELLOW , GREEN };
```

- המילה השמורה inline מבקשת מהמהדר ל"פרוש' את הגדרת הפונקציה בכל קריאה:

```
inline int max(int a, int b)
{
   return a > b ? a : b ;
}
```

bool

- טיפוס חדש לטיפול בערכים בולאנים
- הליטרלים `true` ו- `false` הן חלק מהשפה (מילים שמורות)
- ניתן להמיר ערכים בולאנים למספרים שלמים לפי כללי ההמרה הרגילים (להבדיל מ Java למשל)

עוד מילים שמורות חדשות בשפה...

C++ Keywords That Are Not C Keywords

and	*and_eq*	*asm*	*bitand*	*bitor*	*bool*
catch	*class*	*compl*	*const_cast*	*delete*	*dynamic_cast*
explicit	*export*	*false*	*friend*	*inline*	*mutable*
namespace	*new*	*not*	*not_eq*	*operator*	*or*
or_eq	*private*	*protected*	*public*	*reinterpret_cast*	*static_cast*
template	*this*	*throw*	*true*	*try*	*typeid*
typename	*using*	*virtual*	*wchar_t*	*xor*	*xor_eq*

C++ Keywords That Are C Macros

and	*and_eq*	*bitand*	*bitor*	*bool*	*compl*	*false*	
not	*not_eq*	*or*	*or_eq*	*true*	*wchar_t*	*xor*	*xor_eq*

static_cast<new_type>

- המרת טיפוסים מפורשת ע"י המשתמש:

```
void f(void *x)
{
   char *str = static_cast<char *>(x);
   cout << str;
}
```

סמוך עליי - אני יודע
מה אני עושה...

זהו.