Προγραμματισμός Υπολογιστών με C++

ΚΛΗΣΗ ΣΥΝΑΡΤΗΣΕΩΝ

Γεώργιος Παπαϊωάννου (2013-16)
gepap@aueb.gr

Περιεχόμενο Παρουσίασης

- Περιγραφή:
  - Μορφές μεταβίβασης ορισμάτων σε συναρτήσεις (και μεθόδους) και οι επιπτώσεις τους
  - Επιστροφή τιμών από κλήση συναρτήσεων
  - Υπερφόρτωση συναρτήσεων

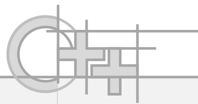
- Τελευταία ενημέρωση: Οκτώβριος 2016

Τρόποι Μεταβίβασης Ορίσματος

- Υπάρχουν 2 τρόποι στη C/C++ να περάσουμε ένα όρισμα σε συνάρτηση/μέθοδο:
  - Κατά τιμή (by value)
  - Με αναφορά (by reference)

Μεταβίβαση Ορίσματος κατά Τιμή

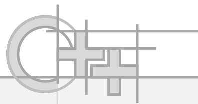
- Σε αυτό τον τρόπο μεταβίβασης ορίσματος που τον γνωρίζεται και από τη Java (δεν έχει η Java άλλον τρόπο), η μεταβλητή που περνάμε **αντιγράφεται** στο όρισμα της συνάρτησης (και άρα σε μια μεταβλητή στη στοίβα κλήσεων)
- Αν τροποποιήσουμε το όρισμα μέσα στη συνάρτηση, η αλλαγή χάνεται όταν επιστρέψουμε από αυτή



Παράδειγμα Μεταβίβασης Ορίσματος με Τιμή (1)

```
int incr(int arg) {    // Το arg είναι ακέραια μεταβλητή.
   arg++;              // Αυξάνει τη μεταβλητή στην οποία
   return arg;         // Επιστρέφει τη νέα τιμή της μεταβλητής.
}


int main() {
   int i = 1, j = 0;
   j = incr(i);                // Αλλάζει μόνο το j.
   cout << i << ", " << j;     // Τυπώνει «1, 2".
}
```



Παράδειγμα Μεταβίβασης Ορίσματος με Τιμή (2)

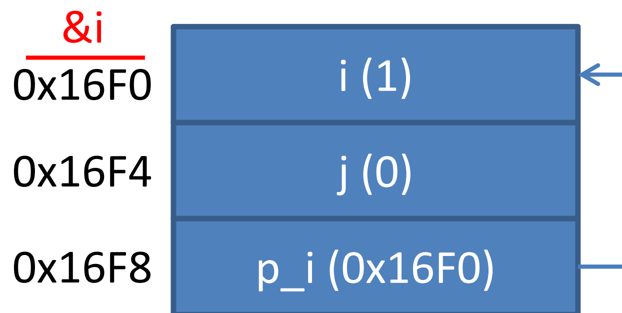
```
int incr(int *arg) { // Το arg είναι δείκτης σε ακέραια μεταβλητή.
    (*arg)++;          // Αυξάνει το περιεχόμενο του δείκτη
                       // Προσοχή: εδώ αντιγράφεται ο δείκτης (τιμή της
                       // arg: η  διεύθυνση που δείχνει), άρα
                       // αλλάζοντας το περιεχόμενο του δείκτη, η
                       // αλλαγή ισχύει και μετά την έξοδο της συνάρτησης
   return *arg;        // Επιστρέφει τη νέα τιμή.
}

int main() {
   int i = 1, j = 0;
   int * p_i = &i;
   j = incr(p_i);                  // Αλλάζει και το i και το j.
   cout << i << ", " << j;         // Τυπώνει ″2, 2″.
}
```

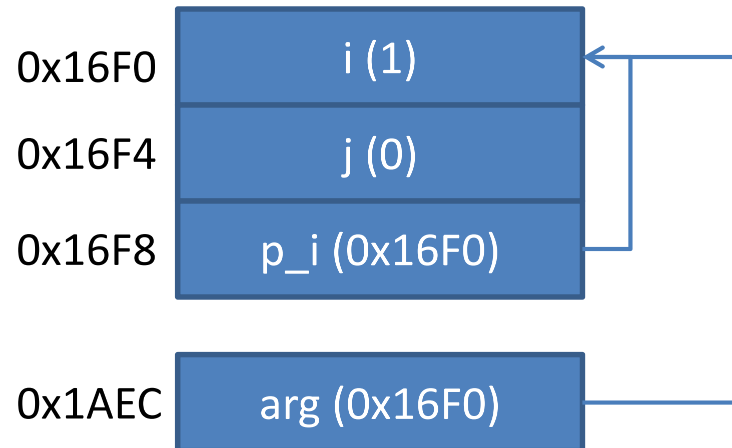
Παρατηρήσεις

- Όταν περνάμε διεύθυνση ή πίνακα (δηλ. διεύθυνση του πρώτου στοιχείου) ως όρισμα, η τιμή που αντιγράφεται είναι ο δείκτης, δηλαδή η μεταβλητή που κρατάει τη διεύθυνση:

```
int i = 1, j = 0;
p_i = &i;
  j = incr(&i);
```




```
int incr(int *arg) {
    *arg++;
    return *arg
}
```



Μεταβίβαση Ορίσματος με Αναφορά

- Είδαμε ότι μπορούμε να μεταβάλουμε τιμές που δίνουμε σε συναρτήσεις, περνώντας τες με δείκτη σε αυτές ως όρισμα
- Υπάρχει άλλος τρόπος;
- Ναι, γιατί στη C/C++ μπορούμε επιπλέον να περάσουμε ένα όρισμα **με αναφορά**
- Ένα όρισμα που μεταβιβάζεται με αναφορά, παίρνει το σύμβολο & πριν το όνομα της παραμέτρου

Παράδειγμα Μεταβίβασης Ορίσματος με Αναφορά

```
int incr(int&  arg) {          // To arg είναι αναφορά σε ακέραια
                               // μεταβλητή.
                               / To arg είναι τοπική αναφορά, αλλά
                               // αναφέρεται στο i της main.
   arg++;                      // Αυξάνει τη μεταβλητή στην οποία
                               // αναφέρεται το arg.
   return arg;                 // Επιστρέφει τη νέα τιμή της μεταβλητής.
}

int main() {
   int i = 1, j = 0;
   j = incr(i);                 // Αλλάζει και το j και το i.
   cout << i << ", " << j;      // Τυπώνει "2, 2".
}
```

Πέρασμα με Δείκτη ή Πέρασμα με Αναφορά;

- Η διαφορές τους είναι ελάχιστες:
  - Επί της ουσίας έχουμε τις ίδιες δυνατότητες ενημέρωσης του ορίσματος
  - Στο πέρασμα με αναφορά δε χρειάζεται να ελέγξουμε το όρισμα αν είναι nullptr (δε γίνεται να είναι, γιατί ΔΕΝ περνάμε δείκτη αλλά μια υπαρκτή μεταβλητή)
  - Μια αναφορά που περνάμε ως όρισμα συνήθως υλοποιείται ως δείκτης από το μεταγλωττιστή, αλλά υπάρχει ενδεχόμενο να γίνουν επιπρόσθετες βελτιστοποιήσεις
  - Μια τοπική μεταβλητή τύπου αναφοράς συνήθως απλά εξαλείφεται και αποτελεί απλά ψευδώνυμο στην μεταβλητή που αναφέρεται

Τι διαφορές Εντόπισα σε Σχέση με τη Java;

C/C++

Μπορώ να περάσω ορίσματα τόσο με αντιγραφή τιμής (by value) όσο και με αναφορά (by reference)

Χρησιμοποιούμε την έννοια της μεταβίβασης δείκτη, εναλλακτικά με το πέρασμα με αναφορά

Java

Περνάω όλα τα ορίσματα μόνο ως τιμές (by value)

Όλα τα αντικείμενα εκτός από τους βασικούς τύπους, δεσμεύονται στο σωρό και μας επιστρέφεται μια «αναφορά» (δηλ. δείκτης!)

Οπότε όταν περνάμε:
void Method(Object data), στην ουσία το Object είναι δείκτης στο δεσμευμένο χώρο του αντικειμένου που τον περνάμε by value

Δείκτης ή Αναφορά;

- Μια αναφορά είτε μετατρέπεται από τον compiler σε δείκτη είτε αντικαθίσταται από τη μεταβλητή που αναφέρεται, κατά τη μεταγλώττιση
- Ο δείκτης είναι πάντα απλά μια διεύθυνση
- Ο δείκτης μπορεί να πάρει τιμή NULL, ενώ η αναφορά για τον προγραμματιστή έχει την έννοια του ψευδώνυμου (ταύτισης) με άλλη μεταβλητή και άρα δεν μπορεί να μην έχει έγκυρη τιμή

Συμβουλές

- Κατά τη μεταβίβαση μεγάλων δομών δεδομένων, η μεταβίβαση κατά τιμή είναι **αργή**, γιατί αντιγράφει τις δομές.

```
int myFun(vector<int> v);
```

Κατά τιμή. Αργό. Π.χ. κατά την κλήση myFun(someVector), το τοπικό vector v της myFun γίνεται αρχικά αντίγραφο του someVector (δεσμεύεται χώρος και καλείται ο κατασκευαστής αντιγραφής!). Πρέπει να αντιγραφούν όλα τα περιεχόμενα του someVector στο v.

- *Στη μεταβίβαση κατά αναφορά (με χρήση δεικτών ή αναφορών) αντιγράφεται μόνο η **διεύθυνση** της δομής, πιο **γρήγορο**.*

```
int myFun(vector<int>& v); //
```

Κατά αναφορά. Γρήγορο. Π.χ. κατά την κλήση myFun(someVector), η τοπική αναφορά v της myFun γίνεται απλά "ψευδώνυμο" (εδώ δείκτης) του someVector.

Πέρασμα Σταθερών Ορισμάτων

- Όταν περνάμε ορίσματα κατά τιμή (by value), έτσι κι αλλιώς δε μπορούμε να μεταβάλουμε τη μεταβλητή που περάσαμε γιατί την αντιγράφουμε σε νέα
- Όταν περνάμε όμως όρισμα με αναφορά, μπορούμε να μεταβάλουμε τη μεταβλητή που περάσαμε
- Μπορούμε να απαγορεύσουμε την αλλαγή του ορίσματος με αναφορά, αν δηλώσουμε το όρισμα ως σταθερό (const):

  Π.χ int myFun(**const** int **&** v);

Συμβουλές – Πότε Χρησιμοποιούμε const;

```
int myFun(vector<int>& v);
```

Κατά αναφορά. Γρήγορο, αλλά π.χ. κατά την κλήση myFun(someVector) έχει η myFun τη δυνατότητα να αλλάξει τα περιεχόμενα του someVector.

```
int myFun(const vector<int>&  v);
```

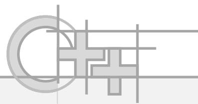
Κατά αναφορά. Γρήγορο και π.χ. κατά την κλήση myFun(someVector) δεν μπορεί η myFun να αλλάξει τα περιεχόμενα του someVector.

- Με μεγάλες δομές δεδομένων, αντί για μεταβίβαση κατά τιμή χρησιμοποιήστε μεταβίβαση με αναφορά (με δείκτες ή αναφορές)
- Χρησιμοποιήστε το const αν θέλετε να εξασφαλίσετε ότι η καλούμενη συνάρτηση δεν θα μπορεί να αλλάξει αυτό που της μεταβιβάζουμε ως όρισμα
- Δεν έχει νόημα για πέρασμα by value!

Πέρασμα const Ορισμάτων - Παράδειγμα

```
void f(string& arg) {  cout << arg; }
void g(const string& arg) { // Υπόσχεται να μην αλλάξει το όρισμα.
   cout << arg;
}

int main() {
   string s1 = "hello";          // Το s1 μπορεί να αλλάξει.
   const string s2 = "world";    // Το s2 δεν μπορεί να αλλάξει.
   f(s1);                        // OK.
   f(s2);                        // Λάθος. Η f δεν εγγυάται ότι δεν θα
                                 // αλλάξει το s2.
   g(s1);  g(s2);   // OK και τα δύο.
   g(s1 + s2);      // Δημιουργείται προσωρινή σταθερά const string
                    // για το άθροισμα, που μεταβιβάζεται κατά αναφορά.
   f(s1 + s2);      // Λάθος. Δεν εγγυάται ότι δεν θα αλλάξει την
}                   // προσωρινή σταθερά του αθροίσματος.
```



Πίνακες ως Ορίσματα Συναρτήσεων (1)

- Στα **ορίσματα συναρτήσεων**, **int arg[]** και **int* arg** είναι **ισοδύναμα**. (Το ίδιο για float arg[] και float* arg κλπ.)

X void incr(**int arg[3]**, int size); *// Δεν επιτρέπεται.*

✓ void incr(**int arg[]**, int size); *//* ***Το ίδιο*** *με το ακόλουθο:*

✓ void incr(**int* arg**, int size);

Πίνακες ως Ορίσματα Συναρτήσεων (2)

```
int main() {
   int myArray[] = {1, 2, 3};
   incr(myArray, 3);     // Το myArray χρησιμοποιείται ως δείκτης.
      // Κατά την κλήση της incr, ο arg δείχνει εκεί που δείχνει ο
      // myArray, δηλαδή δείχνει στην αρχή του πίνακα myArray.
   for(int i = 0; i < 3; i++) {
      cout << myArray[i] << ", " << endl;   // Τυπώνει «2, 3, 4 ».
   }
}
void incr(int* arg, int size) {  // Το arg δείχνει στην πρώτη θέση
                                 // του myArray της main.
   for(int i = 0; i < size; i++) {
      arg[i]++;                  // Αλλάζει τον πίνακα της main.
   }                             // Χρειάζεται να περνάμε το μέγεθος
}                                // του πίνακα, αλλιώς δεν το έχουμε…
```

Τα Ορίσματα argc και argv της main() (1)

```
#include <iostream>
using namespace std;


int main( int argc, char* argv[] ) { // ή char** argv
   for(unsigned i = 0; i < argc; i++) {
      cout << argv[i] << endl;
   }
}
```

Αριθμός ορισμάτων που περνάμε στο πρόγραμμα, συμπεριλαμβανόμενου και του ίδιου του ονόματος του εκτελέσιμου

Ένας δείκτης σε πίνακα χαρακτήρων για κάθε λέξη της γραμμής εντολών. Ο πίνακας χαρακτήρων περιέχει την αντίστοιχη λέξη. Πρώτη λέξη θεωρείται το όνομα του προγράμματος.

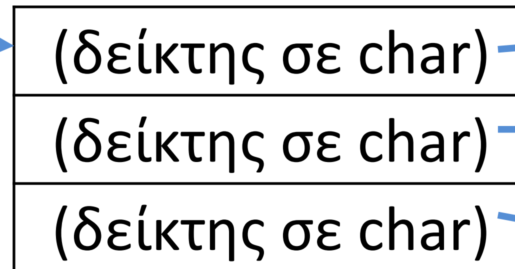
Τα Ορίσματα argc και argv της main() (2)

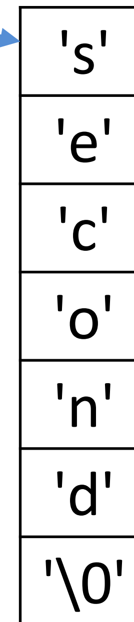
- γραμμή εντολών: `> myprog.exe first second`

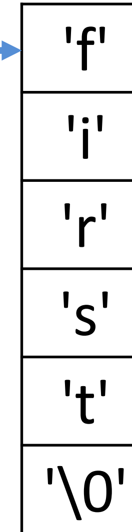
«πίνακας με πίνακες χαρακτήρων»

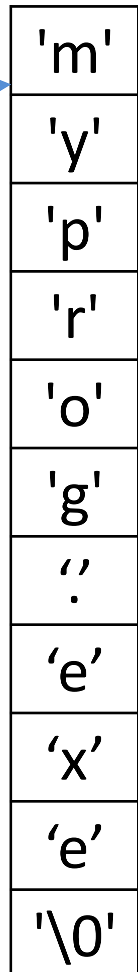
argv

argc = 3









Άρα: char* argv[]

Ισοδύναμα: char** argv

Επιστροφή κατά Τιμή

```
int f(int& arg) {      // Το arg θα αναφέρεται στο i της main.
   arg++;              // Αυξάνει αυτό στο οποίο αναφέρεται το arg.
   return arg;         // Επιστρέφει την τιμή του i της main.
}
int main() {
   int i = 1;
   f(i);          // i = 2. Η τιμή που επιστρέφεται (2) αγνοείται.
   int j = f(i); // i = 3. Η επιστρεφόμενη τιμή (3) αποθηκεύεται στο j.
   i++;           // i = 4.
   cout << j << " " << i << endl;   // Τυπώνει 3 4.
   cout << f(i) << endl;            // Τυπώνει 5.
   // f(i) = 10;                    // Δεν επιτρέπεται. Για τον ίδιο
}                                   // λόγο που δεν επιτρέπεται το 6=10
```

Επιστροφή Αναφοράς (1)

```
int& g(int& arg) { // Το arg θα αναφέρεται στο i της main.
   arg++;          // Αυξάνει αυτό στο οποίο αναφέρεται το arg.
   return arg;
}
```

- Δεν είναι καλή πρακτική να επιστρέφουμε αναφορές
- Προσοχή: **ΠΟΤΕ** δεν επιστρέφω αναφορά σε τοπικές μεταβλητές της συνάρτησης ή μεθόδου!
- Οι μεταβλητές αυτές παύουν να ισχύουν (δεσμεύονται στη στοίβα, όχι στο σωρό) μετά την έξοδο από τη συνάρτηση και άρα οι διεύθυνση της αναφοράς είναι άκυρη.

Παράδειγμα Χρήσης

```
int main() {
 int i = 1;
 int& ref = i;  // Αναφορά στο i.
 g(i);          // i = 2. Το g(i) δεν χρησιμοποιείται
 int j = ref;   // Το j παίρνει την τιμή του i, δηλαδή j = 2
 j = g(i);      // i = 3. Το j παίρνει την τιμή του i, δηλαδή 3
 int& r = g(i); // i = 4. Το r αναφέρεται σε αυτό στο οποίο
                // αναφέρεται και το g(i), δηλαδή αναφορά στο i.
 int& r2 = f(i);  // Δεν επιτρέπεται
 r++; cout << i << endl; // i = 5. Τυπώνει 5.
 cout << g(i) << endl;   // i = 6, τυπώνει 6
 g(i) = 10;              // i = 7, αλλά αμέσως μετά i = 10, γιατί
}                        // το g(i) είναι αναφορά στο i.
```

Επιστροφή Δείκτη

- Κυρίως χρησιμοποιείται όταν δημιουργώ ένα νέο δεδομένο που θέλω να επιστρέψω μέσα στο σώμα μιας συνάρτησης, δεσμεύοντας μια μεταβλητή από το σωρό:

```
float * CreateRampSequence(size_t elements) {
    float * data = new float[elements] ();
    for (size_t i = 0; i<elements; i++ )
         data[i] = i;
    return data;
}
```

- ΔΕΝ επιστρέφω δείκτες σε διευθύνσεις τοπικών μεταβλητών!
- Δεν ξεχνώ ότι πρέπει έξω από τη συνάρτηση σε κάποιο σημείο να αποδεσμεύσω τη μνήμη με delete[]

Επιστροφή με Δείκτη - Λεπτομέρεια

- Επειδή δε γνωρίζω από την υπογραφή της συνάρτησης αν η μνήμη δεσμεύτηκε με new ή με new[] (ως πίνακας), είναι κομψότερο, σε τέτοιες περιπτώσεις να παρέχω και αντίστοιχες συναρτήσεις αποδέσμευσης:

```
void ReleaseSequence(float * data) {
    if (data != nullptr)
        delete[] data;
}
```

- Βοηθάει κάποιον που θα χρησιμοποιήσει τον κώδικα να μην κάνει λάθη

Προκαθορισμένες Τιμές Ορισμάτων

- Στη δήλωση μιας συνάρτησης / μεθόδου μπορούμε να προσδιορίσουμε «default» τιμές για ορίσματα:

```
void print(int i, const string& s = "the value is: ");

int main() {
   int i = 5;
   print(i, "i = ");     // Τυπώνει «i = 5».
   print(i);             // Τυπώνει «the value is: 5».
}

// Ορισμός της συνάρτησης. Η προκαθορισμένη τιμή
// δίνεται στη δήλωση, όχι εδώ:
void print(int i, const string& s) {
   printf(“%s%i\n”, s.c_str(), i);
}
```

Κανόνες Χρήσης Προκαθορισμένων Ορισμάτων

```
void g(int i, int j = 0, float f = 1.0f);
```

- Αν προκαθορίσω τιμή για ένα όρισμα, πρέπει να το κάνω και για όλα τα ορίσματα δεξιά του.

```
X void g(int i, int j = 0, float f);
// Λάθος. Πρέπει να ορίσω προκαθορισμένη τιμή και για το f.
```

Υπερφόρτωση Συναρτήσεων

```
void print(float f);                          // Για τύπωμα float.
void print(string s);                         // Για τύπωμα string.
void print(float arg[], unsigned size); // Τύπωμα πίνακα float.

int main() {
   float f = 5.0F;  int i = 10;  string s = "test";
   float arr[] = {1.0F, 2.0F, 3.0F};
   print(f);                  // Χρησιμοποιείται η πρώτη μορφή.
   print(s);                  // Χρησιμοποιείται η δεύτερη μορφή.
   print(arr, 3);             // Χρησιμοποιείται η τρίτη μορφή.
   print(i);                  // Χρησιμοποιείται η πρώτη μορφή μετά
}                             // από μετατροπή του i σε float.
```

- 3 ξεχωριστοί ορισμοί για τις μορφές της print() …

Παράδειγμα – Οι Ορισμοί των Συναρτήσεων

```
void print(float f) {
   cout << f << endl;
}

void print(string s) {
   cout << s << endl;
}

void print(float arg[], unsigned size) {
   for(unsigned i = 0; i < size; i++) {
      printf("%f ", arg[i]);
   }
   printf("\n");
}
```

Περισσότερα για την Υπερφόρτωση

```
void f(int i);              // Οι μορφές δεν μπορούν να διαφέρουν
int f(int i);               // μόνο στον τύπο επιστροφής.

void g(string s);           // Δε θεωρείται αρκετή η διαφορά μεταξύ
void g(string& s);          // τύπου και αναφοράς στον ίδιο τύπο.

void h(string& s);            // Αν έχουν οριστεί και οι δύο,
void h(const string& s);
// Χρησιμοποιείται η 2η μορφή όταν μεταβιβάζονται σταθερές και η
// 1η μορφή όταν μεταβιβάζονται μη σταθερές.
```

inline Συναρτήσεις

- Όταν μια συνάρτηση δηλωθεί “inline”, τότε αντί να κληθεί, ο κώδικάς της ενσωματώνεται στο σημείο της κλήσης
  - Δε δημιουργείται νέο stack frame
  - Δεν έχουμε το κόστος της κλήσης της συνάρτησης (πρόλογος/επίλογος, πέρασμα και αντιγραφή δεδομένων)
  - Δε μπορούμε να έχουμε inline αναδρομικές συναρτήσεις
  - Ο κώδικας μεγαλώνει, γιατί σε κάθε σημείο που θα γινόταν κλήση της συνάρτησης, επαναλαμβάνεται ο κώδικάς της
- Οι μεταγλωττιστές, συνήθως βάζουν αυτόματα inline κάποιες (συνήθως μικρές) συναρτήσεις σαν βελτιστοποίηση

inline Συναρτήσεις - Παράδειγμα

```
void print(int arg)
{
    cout << "value: "
         << arg;
}


int main() {
    int i = 10;
    print(i);
    …
    i = 20;
    print(i);
}
```

```
inline void print(int arg) {
    cout << "value: " << arg;
}

int main() {
   int i = 10;
   print(i);
   { int arg = i;
     cout << "value:" << arg;}
    …
    i = 20;
    print(i);
    { int arg = i;
      cout << "value:" << arg;}
}
```