Προγραμματισμός Υπολογιστών με C++

ΒΑΣΙΚΟΙ ΤΥΠΟΙ ΚΑΙ ΠΙΝΑΚΕΣ

Γεώργιος Παπαϊωάννου (2013-20)
gepap@aueb.gr

Περιεχόμενο Παρουσίασης

- Περιγραφή:
  - Βασικοί Τύποι
  - Πίνακες (μέρος 1)
  - Συμβολοσειρές
  - Ο Προεπεξεργαστής
- Τελευταία ενημέρωση: Σεπτέμβριος 2016

Ακέραιοι: short, int και long

- **short:** Τουλάχιστον 16 bits (`-32768` ως `32767`).
- **int:** Τουλάχιστον όσα bits και ο short. Όχι περισσότερα από του long.
- **long:** Τουλάχιστον 32 bits (`-2147483648` ως `2147483647`).
- Η C++11 παρέχει και «**long long**» (64 bits).
- **unsigned short**: Όσα bits και ο short, χωρίς αρνητικούς (0 ως `65535`).
- **unsigned int:** Όσα bits και ο int, χωρίς αρνητικούς.
- **unsigned long:** Όσα bits και ο long, χωρίς αρνητικούς (0 ως `4294967295`).

Εξειδικευμένοι Ακέραιοι

- Η γλώσσα υποστηρίζει και τη χρήση ακεραίων με ρητή ακρίβεια, με τους σχετικούς τύπους (δηλώνονται στο <cstdint>):

signed type	unsigned type	description
intmax_t	uintmax_t	Integer type with the maximum width supported.
int8_t	uint8_t	Integer type with a width of exactly 8, 16, 32, or 64 bits. For signed types, negative values are represented using 2's complement. No padding bits. **Optional:** These typedefs are not defined if no types with such characteristics exist.*
int16_t	uint16_t	
int32_t	uint32_t	
int64_t	uint64_t	
int_least8_t	uint_least8_t	Integer type with a minimum of 8, 16, 32, or 64 bits. No other integer type exists with lesser size and at least the specified width.
int_least16_t	uint_least16_t	
int_least32_t	uint_least32_t	
int_least64_t	uint_least64_t	
int_fast8_t	uint_fast8_t	Integer type with a minimum of 8, 16, 32, or 64 bits. At least as fast as any other integer type with at least the specified width.
int_fast16_t	uint_fast16_t	
int_fast32_t	uint_fast32_t	
int_fast64_t	uint_fast64_t	
intptr_t	uintptr_t	Integer type capable of holding a value converted from a void pointer and then be converted back to that type with a value that compares equal to the original pointer. **Optional:** These typedefs may not be defined in some library implementations.*

Ποια είναι τα όρια στον Η/Υ μου;

```
#include <iostream>
#include <climits>    // Ορίζει τα όρια των ακεραίων τύπων
using namespace std;

int main( ) {
      short smax = SHRT_MAX;
      int imax = INT_MAX;
      unsigned uimax = UINT_MAX;
      long lmax = LONG_MAX;
      unsigned long ulmax = ULONG_MAX;
      cout << smax << endl << imax << endl << …;
      return 0;
}
```

Υπερχείλιση Ορίων

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

int main( ) {
    unsigned short s = USHRT_MAX;
    cout << "short = " << s << endl;
    s++;
    cout << "short = " << s << endl;
    return 0;
}
```

- Θα τυπώσει:

```
short = 65535
short = 0    (υπερχείλιση)
```

Προσδιορισμός Ακεραίων Τιμών

```
int i = 1500;                // Τιμή int σε μεταβλητή int.
long l2 = 1500L;             // Τιμή long σε μεταβλητή long.
long l1 = 1500;              // Τιμή int σε μεταβλητή long.
                             // Μετατροπή.
int j = l1;                  // Απόδοση τιμής long σε
                             // μεταβλητή int →
                             // Warning: Απώλεια ακριβείας

1 + 1500                     // Το αποτέλεσμα είναι int.
1L + 1500                    // Το αποτέλεσμα είναι long.
1500U                        // Τιμή unsigned int.
1500UL                       // Τιμή unsigned long.
```

Ο Τύπος Ακεραίων char

- Στη συντριπτική πλειοψηφία αρχιτεκτονικών είναι 8 bits (0 ως 255)
- Χρησιμοποιείται κυρίως:
  - Για την αναπαράσταση ενός **χαρακτήρα**.
  - Στην αναπαράσταση και αποθήκευση ενός **byte** και άρα και
  - Σε buffers από bytes (βλ. ρεύματα δεδομένων)
- Προσοχή: Ο char παραμένει ένας απλός ακέραιος και δεν ταυτίζεται απαραίτητα με έναν ASCII χαρακτήρα, απλά παρέχει το χώρο που απαιτείται για την αποθήκευσή του

Προσδιορισμός Τιμών char

```
char c1 = 'M';  // Αν ASCII, αποθηκεύεται ως 77 και
char c2 = 77;    // τα c1, c2 έχουν την ίδια τιμή.
char c3 = '\n';  // Χαρακτήρας αλλαγής γραμμής.
```

- Άλλοι ειδικοί χαρακτήρες: **\', \", \?**, ...
- Οι ίδιοι και σε συμβολοσειρές: "Hi\n".

```
cout << c1 << c2 << c3 << 'A' << '\n'
     << "\"Hi\n" << "there!\"" << '\n';
```

```
MM
A
"Hi
there!"
```

Πράξεις με Χαρακτήρες

- Επειδή ο char είναι αριθμός κατά βάση, μπορούμε να τον χειριστούμε και ανάλογα:

```
char toUpperCase(char input) {
    if (input >='a' && input <='z')
        return input - 32;
    else
        return input;
}

char uppercase = toUpperCase('v');
```

(αριθμητικές) συγκρίσεις με άλλους χαρακτήρες

Πράξεις με χαρακτήρες

Πράξεις με Χαρακτήρες: Προαγωγή

```
#include <iostream>
using namespace std;
int main( ) {
   char c = 77;              // Κωδικός ASCII του M
   cout << c << endl;        // Τυπώνει "M".
   c++;
   cout << c << endl;        // Τυπώνει "N".
   cin >> c;                 // Γράφω "F". c = 70 (ASCII του F).
   cout << c << endl;        // Τυπώνει "F".
   cout << c + 0 << endl;
}
```

Προσοχή: Προσθέτουμε **int** σε **char** οπότε το αποτέλεσμα της πράξης **προάγεται σε int**. Τυπώνεται 70

Ο Τύπος bool

```
bool b1 = true;
bool b2 = false;
bool b3 = 0;      // Το μηδέν μετατρέπεται σε false.
bool b4 = 3;      // Μη μηδενικές τιμές μετατρέπονται
                  // σε true.

cout << b2 << '\n' << b4;
```

```
0
1
```

Αριθμοί Κινητής Υποδιαστολής

- `56.78 = 5.678 * 10 = 0.5678 * 10^2`

  Σημαντικά Ψηφία — Εκθέτης

- `0.00125 = 1.25 * 10^-3 = 0.125 * 10^-2`

  Σημαντικά Ψηφία — Εκθέτης

- Μεγαλύτερο εύρος από ό,τι αν είχαμε σταθερό αριθμό ψηφίων για ακέραιο/δεκαδικό μέρος.
- Δεν παριστάνονται ακριβώς όλοι οι αριθμοί (π.χ. πώς παριστάνω το `0.00125` με δύο σημαντικά;).

Τύποι Κινητής Υποδιαστολής

Στους σύγχρονους υπολογιστές υλοποιείται το πρότυπο **IEEE 754**, τουλάχιστο για τους float και double:

- **float:** (32bits) 23 σημαντικά bits, 8 bits εκθέτη, 1bit πρόσημο
- **double:** (64bits) 52 σημαντικά bits, 11 bits εκθέτη, 1bit πρόσημο
- **long double:** Τουλάχιστον όσα σημαντικά bits και ο double (64 bits), ή (80 bits): 64 σημαντικά bits, 15 bits εκθέτη, 1bit πρόσημο

Προσδιορισμός Τιμών Κινητής Υποδιαστολής

```
-12.34;          // Εκλαμβάνεται ως double.
-12.34F;         // Τιμή float
-12.34L;         // Τιμή long double
-1234E-2;        // double, σημαίνει -1234 * 10-2.
-1234E-2F;       // float, σημαίνει -1234 * 10-2.
-1234E-2L;       // long double, -1234 * 10-2.
```

- Η C++11 παρέχει μηχανισμούς δημιουργίας ειδικών καταλήξεων και για άλλους τύπους τιμών (π.χ. μιγαδικούς αριθμούς).

Ακρίβεια Υπολογισμών Κινητής Υποδιαστολής

- Ο μεταγλωττιστής ενδέχεται να αλλάξει την ακρίβεια υπολογισμών και αποθήκευσης, ανάλογα με τις ρυθμίσεις ακριβείας αριθμών κιν. υποδιαστολής.
- Για να μάθουμε τι ισχύει στο σύστημά μας για τη συγκεκριμένη ρύθμιση του compiler: *#include <cfloat>*
- Η ακρίβεια των υπολογισμών κιν. υποδιαστολής μπορεί να χαθεί στις πράξεις μας, αν δε μπορεί να εκφραστεί ο αριθμός με την πεπερασμένη ακρίβεια που έχουμε:
  - Πράξεις μεταξύ αριθμών με μεγάλη διαφορά στην κλίμακα
  - Άρρητοι αριθμοί ως αποτελέσματα πράξεων

Μετατροπές Αριθμητικών Τύπων

- Κατά την **εκχώρηση** σε μεταβλητές:
  ```
  int i = 3.84;  // Αποθηκεύεται 3
  ```
- Κατά τη διάρκεια **υπολογισμών**:

  ```
  int i = 10; cout << i / 4.0F;  // Τυπώνει "2.5".
  cout << i / 4;                 // Τυπώνει "2".
  ```
- Κατά τη **μεταβίβαση ορισμάτων** και την **επιστροφή τιμών** από συναρτήσεις.
- **Ρητώς**: `int i = 3; cout << 10 / float(i); // ή (float)i`
- Κατά τη μετατροπή σε τύπους μικρότερου εύρους ή ακρίβειας μπορεί να χαθεί πληροφορία

Μετατροπή Αριθμών: Παρατήρηση

- Η δήλωση float(10) είναι ισοδύναμη με την (float)10 για τους βασικούς τύπους
- Από πλευράς σημασιολογίας όμως είναι διαφορετικά πράγματα:
  - «κατασκευή» αριθμού τύπου float με όρισμα «κατασκευαστή» ακέραιο
  - Καλούπωμα ακεραίου σε δεκαδικό
- Όπως θα δούμε όταν αναφερθούμε σε τρόπους καλουπώματος στη C++, υπάρχουν 4 (!) διαφορετικοί τρόποι να γίνει αυτή η μετατροπή

Πίνακες

```
int a[5];          // Μονοδιάστατος πίνακας 5 θέσεων
a[0] = 10;         // Αρχικοποίηση 1ης θέσης
a[1] = 20;
…
a[4] = 50;         // Αρχικοποίηση 5ης θέσης

double b[5] = {10.0, 20.0, 30.0, 40.0, 50.0};
float c[5] = {10.0F, 20.0F};  // Οι υπόλοιπες
                              // θέσεις μηδενίζονται

float d[5];        // Απροσδιόριστα περιεχόμενα
```

Παρατήρηση: Αρχικοποίηση Μεταβλητών

- Προσοχή: Στη C και στη C++ οι μεταβλητές που χρησιμοποιούμε οπουδήποτε, **δεν αρχικοποιούνται αυτόματα!**
- Για λόγους ταχύτητας, αυτό επαφίεται στον προγραμματιστή
- Τα περιεχόμενα των μεταβλητών που απλά δηλώνονται είναι απροσδιόριστα

Περισσότερα για τους Πίνακες

```
✓ int a[ ] = {1, 2, 3, 4, 5};    // Πίνακας 5 θέσεων.
X int b[ ];                      // Άγνωστο μέγεθος.
✓ int i = 5; float c[i];         // Μέγεθος γνωστό κατά
                                 // τη μεταγλώττιση.
X int i; cin >> i; float c[i];   // Άγνωστο μέγεθος κατά
                                 // τη μεταγλώττιση.
X int f[2] = {1, 2}; cout << f; // Πρέπει να τα
                                 // τυπώσουμε ένα-ένα.
```

- Προσοχή **δεν γίνονται έλεγχοι υπέρβασης ορίων**!

  π.χ. int c[5]; c[10] = 0; *// Απροσδιόριστη συμπεριφορά.*

Πίνακες Πολλών Διαστάσεων

```
int a[2][3] =          // Πίνακας 2X3
  { {1, 2, 3},         // a[0][0] = 1,  a[0][1] = 2,  …
    {4, 5, 6} };       // a[1][0] = 4,  a[1][1] = 5,  …

for(int i = 0; i < 2; i++) {
   for(int j = 0; j < 3; j++) {
      cout << a[i][j] << ' ';
   }
   cout << endl;
}
```

Πίνακες Χαρακτήρων

✓ `char s1[ ] = {'χ','α','ί','ρ','ε','τ','ε','\0'};`

Οι πίνακες χαρακτήρων μπορούν να χρησιμοποιηθούν ως συμβολοσειρές (strings). Στη C, ήταν ο μόνος τρόπος να έχουμε strings. Το '\0' (μηδέν - NULL) στο τέλος σηματοδοτεί το τέλος της συμβολοσειράς (null-terminated strings)

```
✓ char s2[ ] = "χαίρετε";  // Συντομογραφία του
                           // προηγούμενου.
✓ cout << s1 << endl       // Τυπώνει "χαίρετε"
          << s2 << endl;   // δύο φορές.
X char s3[7] = "χαίρετε";  // Δε χωράει. Μήνυμα λάθους.
✓ char s4[10] = "χαίρετε"; // Περισσεύει χώρος.
X char s5[10]; cin >> s5;  // Χωράει; Αν όχι αποθη-
                           // κεύεται σε λάθος χώρο.
```

Σταθερές, const και #define

✓ **const** int n = 1000;

Δήλωση σταθεράς τύπου int και αρχικοποίησή της με την τιμή 1000

X n++;

Η τιμή της σταθεράς είναι κλειδωμένη και είναι σφάλμα να προσπαθούμε να την αλλάξουμε

X const int m;

Δε μπορούμε να αφήσουμε μια σταθερά χωρίς τιμή

✓ **#define** N 1000.

Η εντολή προεπεξεργαστή #define αντικαθιστά το σύμβολο N με το δεξιό όρισμα (1000) όπου το συναντήσει. Η αλλαγή γίνεται κατά το στάδιο της προεπεξεργασίας.

Ψευδώνυμα Τύπων

- Μπορώ να κατασκευάσω ψευδώνυμα (aliases) σύνθετων (compound) αλλά και απλών τύπων δεδομένων με την εντολή typedef:

```
typedef unsigned short UINT16;
```

Ο τύπος στον οποίο δίνω νέο ψευδώνυμο

Το ψευδώνυμο

- Που χρησιμεύουν;
  - Κοινοί τύποι σε ετερογενείς αρχιτεκτονικές
  - Σύντομα ονόματα για μακροσκελείς τύπους. Π.χ.:

```
typedef map<foundations::timers::HighResTimer> timermap_t;
```

Δομές (structures)

- Ο βασικός σύνθετος τύπος είναι η «δομή» (struct) και κληρονομήθηκε από τη C
- Αποτελεί μια απλή σύνθεση άλλων τύπων:

```
struct Person
{
    string name;
    unsigned int age;
};
```

- Σημείωση: Στη C++ θα την αντικαταστήσουμε με την κλάση (class), αλλά στην πράξη είναι ισοδύναμες

Η Εντολή sizeof

- Η συνάρτηση της C++ sizeof(), μας επιστρέφει σε bytes το μέγεθος αποθήκευσης ενός τύπου, είτε βασικού είτε σύνθετου:

Π.χ.:

```
size_t sz = sizeof(float);   // 4
```

Ψευδώνυμο για έναν ακέραιο θετικό τύπο που χρησιμοποιείται στη C++ για αριθμήσιμα μεγέθη (προσδιορίζεται στο αρχείο της C++ crtdefs.h)

sizeof : Προσοχή στους Σύνθετους Τύπους

- Οι σύνθετοι τύποι αποθηκεύονται στη μνήμη Word-aligned (στοίχιση σε 32 bits).
- Άρα, ο χώρος που χρειάζονται είναι πολλαπλάσιο των 4 bytes, ανεξάρτητα από το μέγεθος των επί μέρους τύπων που περιέχουν:

```
typedef struct {
    float a;
    char b;
} Data;
size_t sz = sizeof(Data);  // 8, όχι 4+1
```