ΗΥ352 : ΕΡΓΑΣΤΗΡΙΟ ΤΕΧΝΟΛΟΓΙΑΣ ΛΟΓΙΣΜΙΚΟΥ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ,
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ,
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

1ο ΦΡΟΝΤΙΣΤΗΡΙΟ

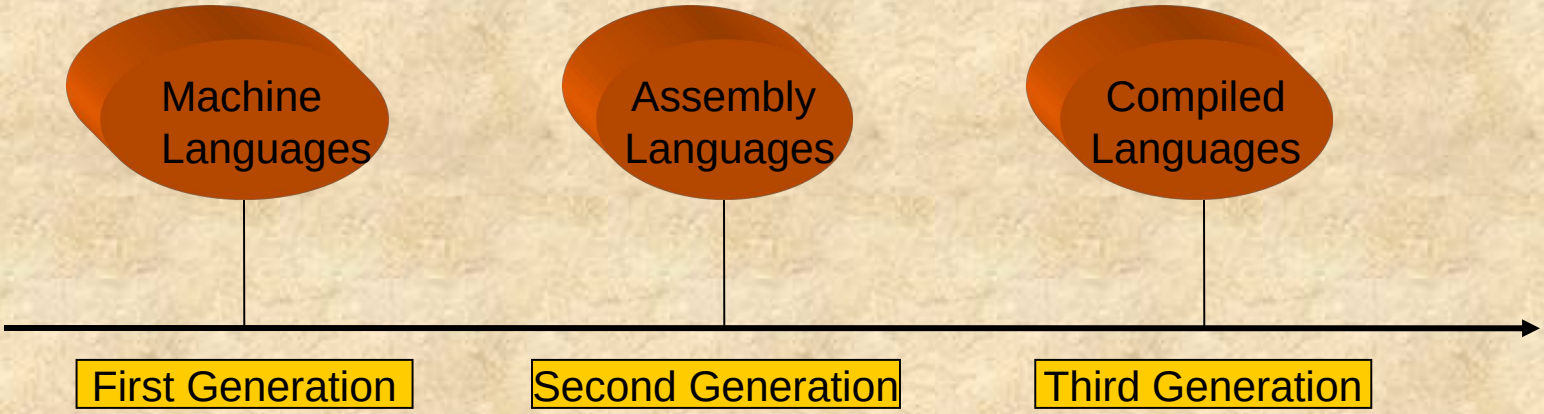
ΔΙΔΑΣΚΟΝΤΑΣ
Αντώνιος Σαββίδης

Περιεχόμενα

- Εισαγωγή στη C++
- Γράφοντας και μεταγλωττίζοντας ένα πρόγραμμα σε C++
- Επανάληψη στη C και βασικά στοιχεία της C++
  - Βασικοί τύποι δεδομένων
  - Μεταβλητές και εμβέλεια
  - Σταθερές
  - Τελεστές
  - Εντολές ροής ελέγχου
  - Συναρτήσεις
  - Δυναμική διαχείριση μνήμης
  - Κλάσεις

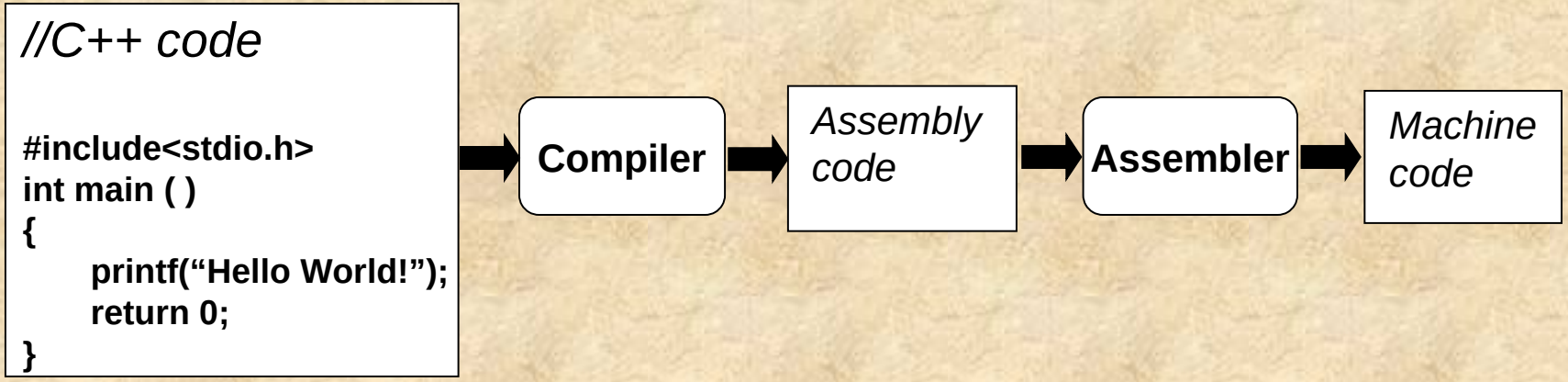
Εισαγωγή στη C++ (1/3)

- Η C++ είναι μια γλώσσα τρίτης γενιάς και μπορεί να τρέξει σε κάθε σύστημα.



Εισαγωγή στη C++ (2/3)

- Κατά τη μεταγλώττιση περνάμε από αρκετά σταδία μέχρι να πάρουμε το εκτελέσιμο αρχείο.



Εισαγωγή στη C++ (3/3)

- Προγραμματίζουμε με C++ επειδή:
  - Αν και πρόκειται για μια γλώσσα τρίτης γενιάς μας δίνει τη δυνατότητα να δουλεύουμε σε χαμηλό επίπεδο (π.χ. λεπτομερής χειρισμός της μνήμης)
  - Είναι αντικειμενοστραφής (object oriented)
  - Υποστηρίζει γενικό προγραμματισμό
  - Χρησιμοποιείται ευρέως
  - Είναι πολύ γρήγορη στην εκτέλεση
  - Υπάρχει ISO standard

Γράφοντας και μεταγλωττίζοντας ένα πρόγραμμα σε C++

```
#include <stdio.h>
int main ( ) {
    printf(“Hello World!”);
    return 0;
}
```

```
#include <iostream>
using namespace std;
int main () {
    cout << "Hello World!";
    return 0;
}
```

- Γράφουμε τον κώδικα με τη χρήση κάποιου κειμενογράφου (vi, pico, …)
- Κατόπιν τον μεταφράζουμε χρησιμοποιώντας τον g++ compiler δίνοντας στη γραμμή εντολών:
  - ***g++ hello.cpp***
- Εάν ο κώδικας δεν περιέχει λάθη τότε δημιουργείτε ένα εκτελέσιμο με όνομα a.out .
- Τρέχουμε το πρόγραμμα μας δίνοντας στη γραμμή εντολών:
  - ***./a.out***
- Το αποτέλεσμα που θα φανεί στην οθόνη μας στο παράδειγμά μας είναι:
  - **Hello World!**

Βασικοί τύποι δεδομένων

Όνομα	Περιγραφή	Μέγεθος	Εύρος τιμών
char	Χαρακτήρας	1 byte	signed: -128 to 127 unsigned: 0 to 255
short int	Μικρός ακέραιος	2 bytes	signed: -32768 to 32767 unsigned: 0 to 65535
int	Ακέραιος	4 bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
long int	Μεγάλος ακέραιος	4 bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
bool	Τιμή αληθείας	1 byte	*true* or *false*
float	Αριθμός κινητής υποδιαστολής	4 bytes	+/- 3.4e +/- 38 (~7 digits)
double	Αριθμός κινητής υποδιαστολής διπλής ακρίβειας	8 bytes	+/- 1.7e +/- 308 (~15 digits)

Μεταβλητές

- Τα ονόματα των μεταβλητών μπορούν να περιέχουν χαρακτήρες, νούμερα και underscores.
  - Δε μπορούν όμως να ξεκινάνε από νούμερο
  - Π.χ. *x*, *xyz123*, *_123*, *_myvar* αλλά όχι *123var*
- Η δήλωση μιας μεταβλητής γίνεται ως εξής:
  - *type identifier;*
    - *unsigned int x; float f; char * str;*
  - *type identifier = value;*
    - *double d = 2.5; bool flag = true;*
- Δεν είναι απαραίτητο οι δηλώσεις μεταβλητών να βρίσκονται μόνο στην αρχή του block

```
int main ( ) {
    int a;
    a = 8;
    int b = 3;
    int sum = a + b, x = 0;
    for(int i = 0; i < 5; ++i)

       printf("%d\n, i);
    return 0;
}
```

Εμβέλειες (scopes) μεταβλητών

- Μια μεταβλητή μπορεί να είναι *τοπική (local)* ή *καθολική (global)*
  - Οι καθολικές μεταβλητές είναι ορατές παντού από το σημείο της δήλωσής τους
  - Οι τοπικές μεταβλητές είναι ορατές μόνο στο scope το οποίο ορίζονται (και πιθανά εσωτερικά scopes αυτού).
- Μπορούμε να αναφερθούμε σε μια global μεταβλητή με τη χρήση του τελεστή ***::***

```
int a = 1;       //global variable a

int f() {        //entering function scope
    int a = 3;   //local a only visible in f
    return a;
}                //exiting function scope

int main() {   //entering function scope
    printf("a=%d\n", a);//global a, prints 1
    int a = 2;   //local variable a
    printf("a=%d\n", a);//local a, prints 2
    {   //entering block scope
      int a=f(); //local a shadows previous
      printf("a=%d\n", a);//prints 3
      printf("a=%d\n", ::a);//::a is global a
    }   //exiting block scope
    printf("a=%d\n", a); //local a, prints 2
}       //exiting function scope
```

Σταθερές τιμές (1/2)

- **Σταθερές ακεραίων**
  - Με βάση το 10 (decimal), π.χ. 1024, 9345, -123
  - Με βάση το 8 (octal), π.χ. 0113 *(= 75)*, 0456
  - Με βάση το 16 (hexadecimal), π.χ. 0x4b *(= 75)*, 0X9ca
- **Σταθερές αριθμών κινητής υποδιαστολής**
  - 3.14159, 6.02e23 *(= 6.02 x 10^{23})*, 1.6E-19 *(=1.6 x 10^{-19})*
- By default, ο τύπος των ακέραιων σταθερών είναι int, ενώ των σταθερών κινητής υποδιαστολής double
- Μπορούμε να αλλάξουμε αυτό τον τύπο βάζοντας δίπλα από τη σταθερά κάποιους χαρακτήρες (case insensitive)
  - 75 (int), 75u (unsigned int), 75L (long int), 75ul (unsigned long)
  - 1.1 (double), 3.14159L (long double), 6.02e23f (float)

Σταθερές τιμές (2/2)

- **Χαρακτήρες και συμβολοσειρές (strings)**
  - π.χ. ‘a’, ‘\n’, “hello world”, “tab\t separated \t words”
  - Κάποιοι ειδικοί χαρακτήρες (escape characters)
    - \n, \t, \b, \a, \’, \”, \\
  - Συνεχόμενα strings συνενώνονται από τον compiler
    - π.χ. το “hello” “ “ “ world” γίνεται “hello world”
- **Τιμές αληθείας**
  - *true* και *false*
- **Δηλωμένες σταθερές μεταβλητές (consts)**
  - Κανονικές μεταβλητές που απλά δε μπορούν να αλλάξουν τιμή
  - *π.χ. const int x = 5; const char tab = ‘\t’;*

Τελεστές (1/3)

- **Εκχώρηση (=)**
  - π.χ. a = 1; a = b = c = 5; a = 2 + (b = 5);
- **Αριθμητικοί τελεστές (+, -, *, /, %)**
  - π.χ. 1 + 2, a = 3 * 4 / 5 % 6;
- **Μοναδιαία αύξηση και μείωση (++, --)**
  - Prefix και postfix
    - a = b++; //equal to a = b; b = b + 1;
    - a = --b; //equal to b = b -1 1; a = b;
- **Συσχετιστικοί τελεστές (==, !=, >=, >, < <=)**
  - π.χ. x = a > 2; if (x % 2 == 0) …

Τελεστές (2/3)

- **Λογικοί τελεστές (&&, ||, !)**
  - π.χ. x = a > b || c < d; while(!finished && c != 0) …
- **Bitwise τελεστές (&, |, ^, ~, <<, >>)**
  - x = 0xf0 | 0x0f; // 11110000 | 00001111 = 11111111 = 255
  - x = 3 << 1; //11 << 1 = 110 (doubles the number)
- **Τριαδικός τελεστής (?:)**
  - char *s = a % 2 == 0 ? “even” : “odd”;
- **Επιπλέον εκχωρήσεις (+=, -=, *=, /=, %=, >>=, <<=, &=, ^=, |=)**
  - x += 1; // x = x + 1;
  - y *= x + 2; // y = y * (x + 2);
  - z >>= 2; // z = z >> 2;

Τελεστές (3/3)

- **Type cast (*(type)*)**
  - Μετατρέπει την τιμή ενός τύπου σε ένα άλλο τύπο
  - π.χ. int x = (int) 3.14; // x = 3
- **sizeof()**
  - Επιστρέφει το μέγεθος σε bytes ενός τύπου ή κάποιας μεταβλητής
  - Δεν κάνουμε ποτέ υποθέσεις για το μέγεθος ενός τύπου ή μεταβλητής; πάντα χρησιμοποιούμε το sizeof
  - π.χ. int x = sizeof(char); //x = 1
- **Τελεστής κόμμα (,)**
  - Χρησιμοποιείται για να χωρίσει δύο ή παραπάνω εκφράσεις όταν στο σημείο χρήσης αναμένεται μόνο μια.
  - Για την τιμή της έκφρασης, χρησιμοποιείται η δεξιότερη έκφραση
  - π.χ. a = (b = 3, b+2); //b = 3, a = b + 2 ➔ a = 5

Εντολές ροής ελέγχου (1/2)

- **if (***condition***)** *statement1*
  else *statement2*
- **while (***condition***)** *statement*
- **do** *statement* **while (***condition***);**
- **for (***init; condition; increment***)** *statement*
- **break**, **continue**

```
if (x > 0)
   printf("positive");
else if (x < 0)
   printf("negative");
else
   printf("zero");
```

```
while(x > 0) {
   printf("%d\n", x);
   --x;
}
```

```
int n;
do {
   scanf("%d", &n);
   printf("n=%d\n", n);
} while(n != 0);
```

```
for(int i=0; i<5; ++i)
   printf("i=%d\n", i);
for (n=0,i=100; n!=i; n++,i--)
  {/*do something*/}
```

```
while(true) {

if(getchar()==EOF)
   break;
 else continue;
}
```

Εντολές ροής ελέγχου (2/2)

- **goto** *label*
- **switch (***expression***) {**
  case *const1***:** *statement group1*
  case *const2***:** *statement group2*
  ...
  default: *statement group*
  }

```
for(...)
   for(...) {
     doSomething();
     if (errorOccured())
       goto error;
   }
error: printf("Error!");
```

```
switch (x) {
  case 1: printf("x is 1"); break;
  case 2: printf("x is 2"); break;
  case 3: //fallthrough
  case 4: printf("x is 3 or 4"); break;
  default: printf("value of x unknown"); break;
}
```

Συναρτήσεις

- Οι συναρτήσεις δηλώνονται ως εξής:

  <return type> functionName (type$_1$ arg$_1$, …, type$_n$ arg$_n$) { body }

- π.χ. int add(int x, int y) { return x + y; }
- Η κλήση τους γίνεται αποτιμώντας τα πραγματικά ορισμάτων και αντιστοιχίζοντάς τα με τα τυπικά
- **Προσοχή:** Η αντιστοίχιση των ορισμάτων δεν γίνεται πάντα από το πρώτο προς το τελευταίο

```
int a = 0;
int f() { return a++; }
void printargs(int x, int y) {
   printf("x = %d, y = %d\n", x, y);
}
printargs(f(), f()); //prints x = 1, y = 0
```

Δυναμική Δέσμευση μνήμης (1/2)

- Οι operator new & delete μας δίνουν τη δυνατότητα να έχουμε δυναμική δέσμευση και αποδέσμευση μνήμης
  - **new** *type <initialization>* και **new** *type [size]*
    - new int;
    - new int(2);
    - new char[10];
  - **delete** *variable* και **delete** [] *variable*
    - delete i;
    - delete [ ] array;

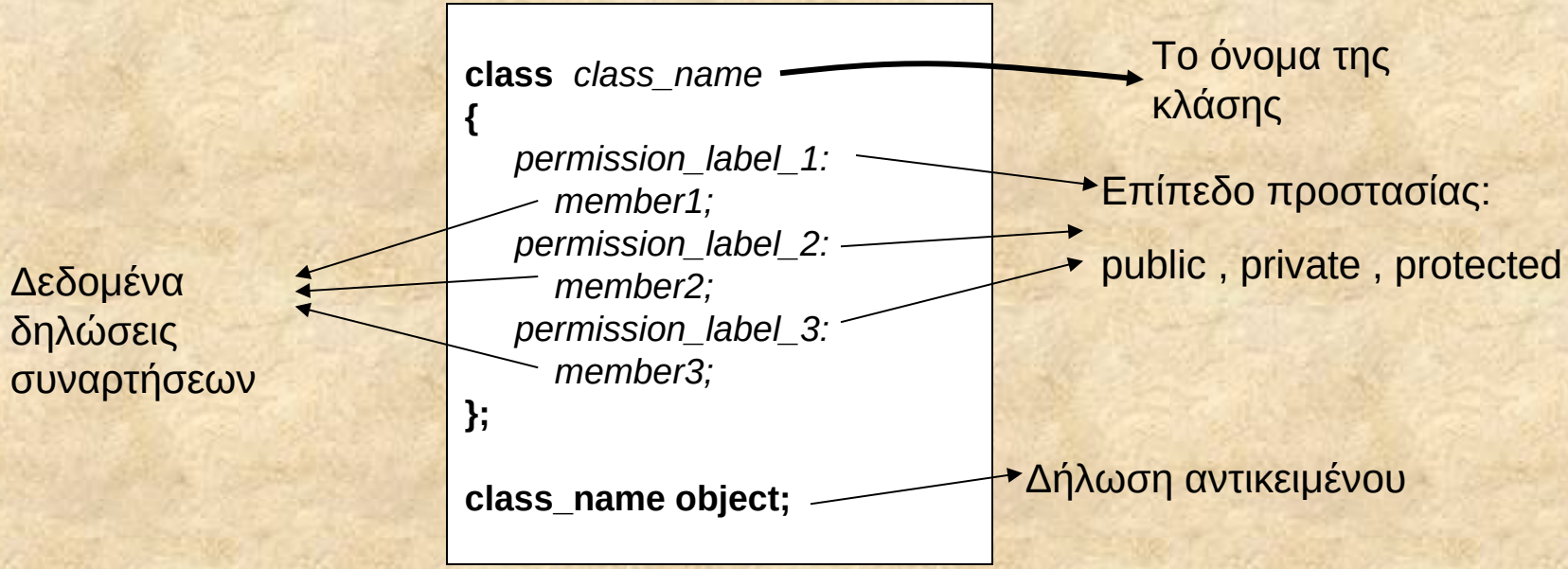
Δυναμική Δέσμευση μνήμης (2/2)

- Οι *malloc* & *free* της C εξακολουθούν να υπάρχουν συνήθως δεν τις χρησιμοποιούμε
- Αν το κάνουμε, **ΠΟΤΕ** δεν πρέπει να μπλέκουμε κλήσεις *malloc/free* με τους τελεστές *new/delete*.
- Το πιο πιθανό είναι ότι θα προκαλέσουμε segmentation fault.

```
int main ( ) {
    int* p;
    p=(int*)malloc(sizeof(int)*5);
    // some code
    free ( ( void* ) p ) ;
    // avoid using malloc & free
    int* pa;
    pa = new int [5];
    // some code
    delete [ ] pa;
    char *s = new char;
    delete s;
}
```

Κλάσεις (1/3)

- Η κλάση είναι ένας τρόπος ώστε να οργανώνουμε δεδομένα και λειτουργίες στην ίδια δομή
- Δηλώνεται χρησιμοποιώντας το keyword *class*



Κλάσεις (2/3)

- Τα επίπεδα προστασίας μιας μεταβλητής ή μίας συνάρτησης είναι τα παρακάτω και έχουν τις εξής ιδιότητες
  - **private:** *Αυτές οι μεταβλητές είναι προσπελάσιμες μόνο μέσα από την ίδια την κλάση ή από μία άλλη 'φιλική' κλάση*
  - **protected:** *Είναι προσπελάσιμες όπως τις private μεταβλητές και επιπλέον από derived classes*
  - **public:** *Είναι προσπελάσιμες από όπου η κλάση είναι ορατή*
- Όταν δεν χρησιμοποιούμε access modifier τα μέλη της κλάσης by default θεωρούνται private.

Κλάσεις (3/3)

```
#include <stdio.h>
class Point {
    int x, y;        // by default private
public:
    Point(int _x, int _y) : x(_x), y(_y) {}
    //same as Point(int _x, int _y) { x = _x; y = _y; }
    int GetX(void) { return x; }
    int GetY(void) { return y; }
    void Move(int dx, int dy);
};
void Point::Move(int dx, int dy) { x += dx, y += dy; }

int main(void) {
    Point p(10, 20);
    printf("p =(%d, %d)\n", p.GetX(), p.GetY());
    p.x = 10;    ///< compile error private member
}
```