1. **Punti 2** – Indicare quali delle seguenti affermazioni sugli header file sono vere:
   * 1. Un header file implementa l’interfaccia di un modulo
     2. In un makefile, nella regola per la compilazione di un singolo file sorgente è necessario specificare come pre-requisito tutti gli header file dei moduli del progetto
     3. Il file .cc di un modulo fornitore deve includere tutti gli header file dei moduli che utilizzano i suoi servizi
     4. Gli header file vengono compilati
2. **Punti 2** – Nella compilazione separata di un progetto composto da più file sorgenti (sono possibili 0 o più risposte):
   * 1. La fase di traduzione dà luogo ad un file oggetto
     2. Nella fase di traduzione vengono segnalati gli errori dovuti a mancate ridefinizioni
     3. Viene generato un solo file oggetto
     4. Le macro vengono risolte nella fase di linking
3. **Punti 4** – Domanda a risposta aperta  
   Cosa stampa il seguente programma: \_\_\_\_\_10 20 2 21\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#include <iostream>

using namespace std ;

int f(int i, int \*p, int &ri, int \*&rp) {

i = 10 ;

\*p = i++ ;

ri = \*p\*2;

rp = p+1;

return i+\*p;

}

int main() {

int a[3] = {1,2,3}, b=6, c=12, d=8;

int \*prp;

prp = &d;

int sol =f(\*(a+1), a, c, prp);

cout << \*a << " " << c << " " << \*prp << " " << sol << endl;}

1. **Punti 3** – Domanda a risposta aperta  
   Cosa stampa il seguente programma: \_\_\_\_\_2 4 2 6 2 4 2\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

using namespace std;

void stampa(int n){

if(n==0) return;

if (n % 2)

stampa(n-1);

else{

stampa(n-1);

cout << n <<" ";

stampa(n-2);

return;}}

int main(){

int x = 7;

stampa(x);}

1. **Punti 4** – Scrittura di codice

Date le seguenti dichiarazioni per un binary search tree con chiave intera, scrivere il codice della primitiva per la ricerca di un nodo **bst\_search** in *forma ricorsiva*, assumendo che i nodi siano ordinati in *ordine decrescente* (dal più grande al più piccolo):

**typedef** **int** tipo\_key;

**struct** bnode {

int key;

bnode\* left;

bnode\* right;

bnode\* parent;

};

bnode\* **bst\_search**(bnode\* b, int k){

**if**(b == NULL)

**return** NULL;

**else**{

**if** (k == b->key)

**return** b;

**if** (k>b->key)

return bst\_search(b->left,k);

**else**

return bst\_search(b->right, k);

}

}

1. **Punti 4** – Scrittura di codice

Date la seguente dichiarazione di coda e primitive su tipo\_inf, scrivere il codice della primitiva dequeue:

**struct** elem

{

tipo\_inf inf;

elem\* pun ;};

**typedef** elem\* lista;

**typedef** **struct**{

lista head;

elem\* tail;} coda;

**int** compare(tipo\_inf,tipo\_inf);

**void** copy(tipo\_inf&,tipo\_inf);

**void** print(tipo\_inf);

tipo\_inf dequeue(coda& c){

tipo\_inf ris;

copy(ris,c.head->inf);

lista app = c.head;

c.head=(c.head)->pun;

**delete** app;

**return** ris;

};

1. **Punti 4** – Scrittura di codice

Data una lista *l* semplice di interi definita come sotto e un valore *v*, scrivere una funzione che restituisca una lista *l’* ottenuta da *l* che contenga tutti gli elementi con valori minori o uguali a *v*. La funzione deve far uso delle primitive specificate sotto, non deve creare o distruggere elementi ma semplicemente effettuare operazioni sui collegamenti e ha come effetto collaterale l’eliminazione degli elementi da *l*.

**struct** elem

{

**int** v;

elem\* pun ;

} ;

**typedef** elem\* lista ;

tipo\_inf head(lista);

lista tail(lista);

lista **select**(lista& l, int v){

/\* l’: testa della nuova lista, curr: elemento corrente della nuova lista \*/

lista l’ = curr = prev = NULL;

lista app = l;

while(app!=NULL){

if(head(app)<=v){

if(l’!=NULL){ /\* l’ già inizializzato \*/

curr->pun = app;

curr = curr->pun;}

else

l’=curr=app; /\* inizializzo l’ \*/

if (app == l) /\* app è la testa della lista \*/

l = app = tail(l); /\* aggiorno la testa \*/

else

prev->pun = tail(app); /\* sgancio app \*/

curr -> pun = NULL;}

else prev = app; /\* aggiorno prev \*/

if (prev != NULL) /\* Se app non è la testa di l \*/

app = tail(prev);}

return l’;}

1. **Punti 4** – Scrittura di codice

Dati due alberi binari definiti come sotto, scrivere la funzione booleana same che restituisca true se i due alberi sono uguali, false altrimenti.

**struct** bnode {

tipo\_inf inf;

bnode\* parent;

bnode\* left;

bnode\* right;

};

**typedef** bnode\* btree;

bool same(btree b1, btree b2){

if(b1== NULL || b2 == NULL)

if( b1 == NULL && b2 == NULL)

return true;

else

return false;

else {

bool left = same(b1->left,b2->left);

bool right = same(b2->right, b2->right);

return b1->inf == b2 -> inf && left && right;}}

1. **Punti 5** – Scrittura di codice

Dato un grafo orientato, si definisce grado medio il numero medio degli archi in entrata o in uscita da ogni nodo ovvero cardinalità(E)/cardinalità(N).

Data la seguente definizione di grafo, scrivere la funzione mean che restituisca il grado medio del grafo:

**struct** adj\_node {

**int** node;

**float** weight;

**struct** adj\_node\* next;

};

**typedef** adj\_node\* adj\_list;

**typedef** **struct**

{adj\_list\* nodes;

**int** dim;

} graph;

float mean(graph g){

int E=0;

for (int i=1; i<=get\_dim(G); i++) {

adj\_list tmp;

tmp=;

for(tmp= get\_adjlist(G,i);tmp!=NULL; tmp=get\_nextadj(tmp))

E++;

}

return (float) E/get\_dim(G);

}

}