Principali programmi da sapere

Informatica A
anno 2020/2021

Aggiunta in coda

```
list tailInsertIte(list l, int content){
    list currentTail = l;
    node *n = (node *)malloc(sizeof(node));
    n->val = content;
    n->next = NULL;

    if (l==NULL)
        return n;
    while(currentTail->next != NULL){
        currentTail = currentTail->next;
    }
    currentTail->next = n;
    return l;
}
```

Inserimento ordinato

Ricorsivo

```
list sortInsertRic(list l, int content){
    node *n;

    if (l!=NULL && content > l->content){
        l->next = sortInsertRic(l->next, content);
    }
    else{
        n = (node *)malloc(sizeof(node));
        n->content = content;
        n->next = l;
        return n;
    }
}
```

Iterativo

```
list sortInsertIte(list l, int content){
    list currentNode = l;
    node *n = (node *)malloc(sizeof(node));

    if (l==NULL || content<=l->content){
        //return headInsert(l, content);
        n->next = l;
        n->content = contentM
    }

    while(currentNode->next!=NULL && content > currentNode->next->content)
        currentNode = currentNode->next;

    n->next = currentNode->next;
    n->content = content;
    currentNode->next = n;
    return l;

}
```

Clear list

Ricorsivo

```
void clearListRic(list l){
    node *temp;

    if (l!=NULL){
        temp = l->next;
        free(l);
        clearListRic(temp);
    }
}
```

Iterativo

```
list clearListIte(list l){
    list temp;
    while(l!=NULL){
        temp = l;
        l = l->next;
        free(temp);
    }
    return NULL;
}
```

Ordinamento lista

Ricorsivo

```c
void ordinaListaRic(ptrNode lista, ptrNode prec, ptrNode *testa, int swap){
    ptrNode temp;

    if ((lista == NULL) || (lista->next == NULL)){
        if (swap == 1)
            //inizio da capo
            ordinaListaRic(*testa, NULL, testa, 0);
        else
            return;
    }
    else{
        if (lista->num > lista->next->num){
            temp = lista;
            lista = lista->next;
            temp->next = lista->next
            lista->next = temp;
            if (prec == NULL)
                *testa = lista;
            else
                prec->next = lista;
            swap = 1;

        }
        ordinaListaRic(lista->next, lista, testa, swap);
    }
}
```

Iterativo

```c
ptrNode ordinaLista(ptrNode lista){
    ptrNode temp;
    ptrNode testa = lista;
    ptrNode prec = NULL;
    int swap;

    if ((lista == NULL) || (lista->next==NULL))
        return lista;

    do{
        swap = 0;
        prec = NULL;
        lista = testa;
        while(lista->next != NULL){
            if(lista->num > lista->next->num){
                temp = lista;
                lista = lista->next;
                temp->next = lista->next;
                lista->next = temp;
                swap = 1;
                if (prec == NULL)
                    testa = lista;
                else
                    prec->next = lista;
            }
            prec = lista;
            lista = lista->next;
        }
    }while(swap==1);
    return testa;
}
```

Rimozione primo elemento con una data caratteristica

Ricorsivo

```
list removeFirstRic(list l, int content){
    node *n;
    if (l == NULL){
        return l;
    }
    else if (l->content == content){
        n = l->next;
        free(l);
        return n;
    }
    else{
        l->next = removeFirstRic(l->next, content);
        return l;
    }
}
```

Iterativo

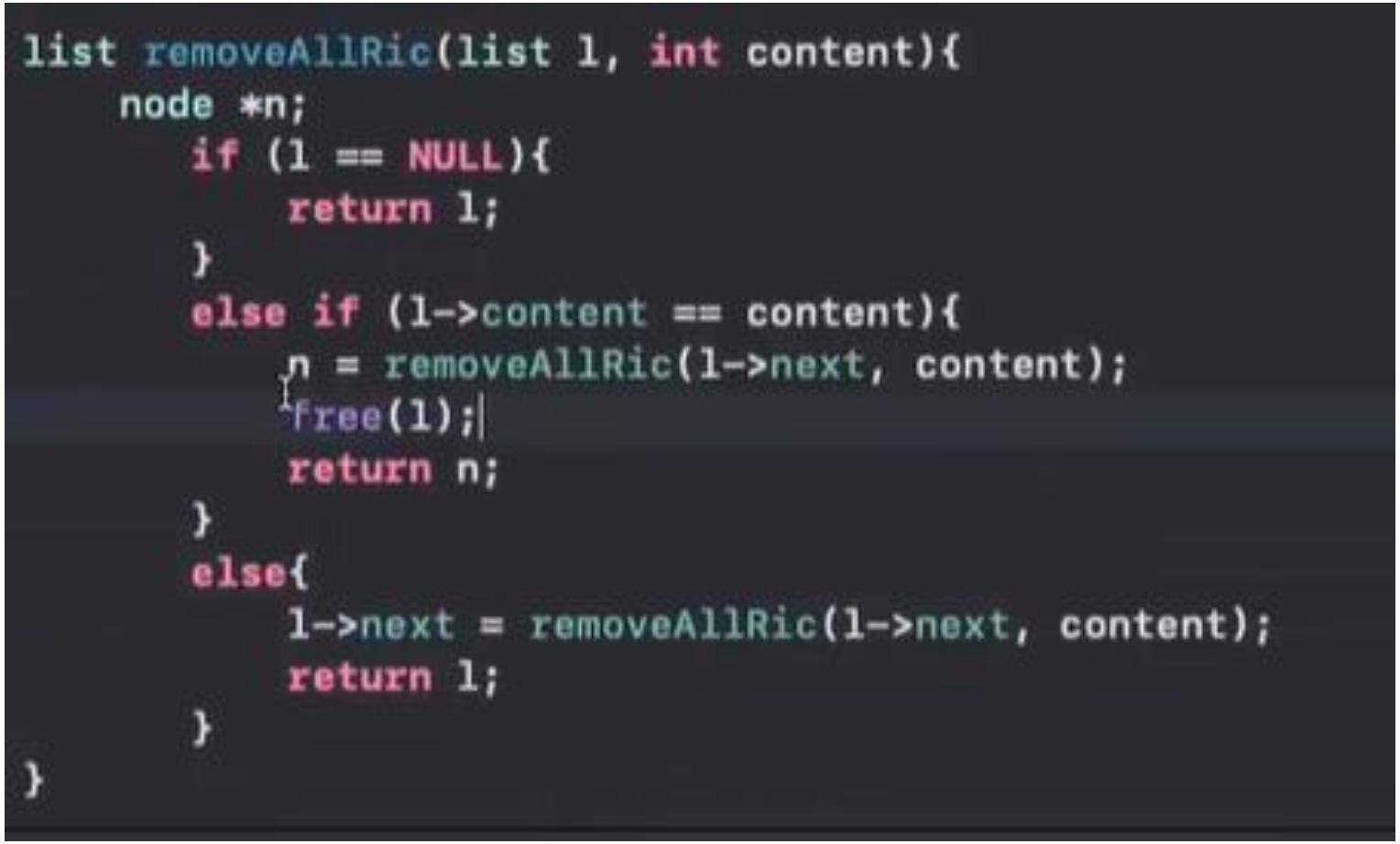
```
list removeFirstIte(list l, int content){
    list previousNode = NULL;
    list currentNode = l;
    list head = l;

    while( currentNode != NULL && currentNode->content != content){
        previousNode = currentNode;
        currentNode = currentNode->next;
    }

    if (currentNode != NULL){
        if (previousNode == NULL)
            head = currentNode->next;
        else
            previousNode->next = currentNode->next;
        free(currentNode);
    }
    return head;
}
```

Rimozione di tutti gli elementi con una data caratteristica

Ricorsivo



```
list removeAllRic(list l, int content){
    node *n;
        if (l == NULL){
            return l;
        }
        else if (l->content == content){
            n = removeAllRic(l->next, content);
            free(l);
            return n;
        }
        else{
            l->next = removeAllRic(l->next, content);
            return l;
        }
}
```

Rimozione dalla testa con restituzione valore rimosso

```c
ptrNode rimuoviDaPila(ptrNode lista, char *letto){
    ptrNode temp;

    if (lista == NULL)
    {
        *letto = ' ';
        return NULL;
    }
    else{
        temp = lista;
        lista = lista->next;
        *letto = temp->carattere;
        free(temp);
        return lista;
    }
}
```

```c
char rimuodiDaPila2(ptrNode *lista){
    char c;
    ptrNode temp;

    if (*lista == NULL)
    {
        return ' ';
    }
    else{
        temp = *lista;
        *lista = (*lista)->next;
        c = temp->carattere;
        free(temp);
        return c;
    }

}
```

Shuffle merge

Ricorsivo

```
ptrNode shuffle_merge_ric(ptrNode l1, ptrNode l2, ptrNode newLista){
    if (l1==NULL && l2==NULL)
        return newLista;

    if (l1!=NULL)
        newLista = inserisciInTesta(newLista, l1->num);
    if (l2!=NULL)
        newLista = inserisciInTesta(newLista, l2->num);

    //Chimata ricorsiva
    if(l1!=NULL && l2==NULL)
        //l1
        newLista = shuffle_merge_ric(l1->next, l2, newLista);
    else if(l1==NULL && l2!=NULL)
        //l2
        newLista = shuffle_merge_ric(l1, l2->next, newLista);
    else
        ///
        //
        newLista = shuffle_merge_ric(l1->next, l2->next, newLista);
    return newLista;
}
```

Iterativo

```
ptrNode shuffle_merge_ite(ptrNode l1, ptrNode l2, ptrNode newLista){

    while(l1 != NULL && l2!=NULL){
        newLista = inserisciInTesta(newLista, l1->num);
        newLista = inserisciInTesta(newLista, l2->num);
        l1 = l1->next;
        l2 = l2->next;
    }
    while(l1 != NULL){
        newLista = inserisciInTesta(newLista, l1->num);
        l1 = l1->next;
    }
    while(l2 != NULL){
        newLista = inserisciInTesta(newLista, l2->num);
        l2 = l2->next;
    }
    return newLista;
}
```