問2　探索アルゴリズムであるハッシュ法の一つ，チェイン法に関する次の記述を読んで，設問1～4に答えよ。

配列に対して，データを格納すべき位置（配列の添字）をデータのキーの値を引数とする関数（ハッシュ関数）で求めることによって，探索だけではなく追加や削除も効率よく行うのがハッシュ法である。

通常，キーのとり得る値の数に比べて，配列の添字として使える値の範囲は狭いので，衝突（collision）と呼ばれる現象が起こり得る。衝突が発生した場合の対処方法の一つとして，同一のハッシュ値をもつデータを線形リストによって管理するチェイン法（連鎖法ともいう）がある。

8個のデータを格納したときの例を図1に示す。

このとき，キー値は正の整数，配列の添字は0～6の整数，ハッシュ関数は引数を7で割った剰余を求める関数とする。

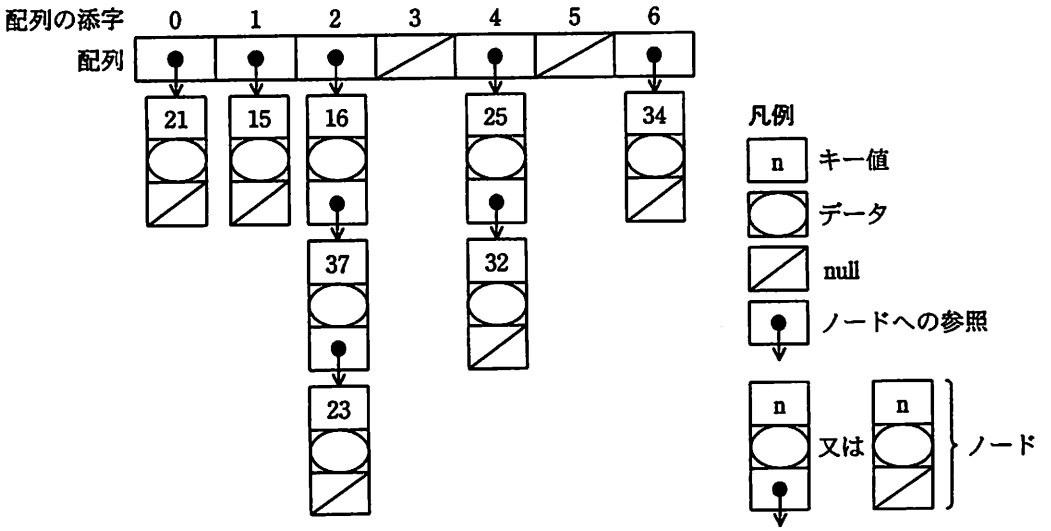

図1　チェイン法のデータ格納例

このチェイン法を実現するために，表に示す構造体，配列及び関数を使用する。

表　使用する構造体，配列及び関数

名称	種類	内容
Node	構造体	線形リスト中の各ノードのデータ構造で，次の要素から構成される。 key…キー値 data…データ nextNode…後続ノードへの参照
table	配列	ノードへの参照を格納する。この配列をハッシュ表という。配列の各要素は，table[n]と表記する（n は配列の添字)。配列の添字は 0 から始めるものとする。各要素の初期値は null である。
hashValue(key)	関数	キー値 key を引数として，ハッシュ値を返す。

構造体を参照する変数からその構造体の構成要素へのアクセスには“.”を用いる。例えば，図 1 のキー値 25 のデータは table[4].data でアクセスできる。また，構造体の実体を生成するには，次のように書き，生成された構造体への参照が値になる。

```
new Node(key, data, nextNode)
```

〔探索関数 search〕

探索のアルゴリズムを実装した関数 search の処理手順を次の(1)～(3)に，そのプログラムを図 2 に示す。

(1) 探索したいデータのキー値からハッシュ値を求める。

(2) ハッシュ表中のハッシュ値を添字とする要素が参照する線形リストに着目する。

(3) 線形リストのノードに先頭から順にアクセスする。キー値と同じ値が見つかれば，そのノードに格納されたデータを返す。末尾まで探して見つからなければ null を返す。

```
function search(key)
  hash ← hashValue(key);        // 探索するデータのハッシュ値
  node ← table[hash];           // 着目する線形リストへの参照
  while(node が null でない)
    if( [ア] )
      return node.data;         // 探索成功
    endif
    [イ] ;                      // 後続ノードに着目
  endwhile
  return [ウ] ;                 // 探索失敗
endfunction
```

図 2　探索関数 search のプログラム

〔追加関数 addNode〕

データの追加手続を実装した関数 addNode の処理手順を次の(1)～(3)に，プログラムを図 3 に示す。図 3 中の［ ア ］，［ イ ］には，図 2 中の［ ア ］，［ イ ］と同一のものが入る。

(1) 追加したいデータのキー値からハッシュ値を求める。

(2) ハッシュ表中のハッシュ値を添字とする要素が参照する線形リストに着目する。

(3) 線形リストのノードに先頭から順にアクセスする。キー値と同じ値が見つかれば，キー値は登録済みであり追加失敗として false を返す。末尾まで探して見つからなければ，リストの先頭にノードを追加して true を返す。

```
function addNode(key, data)
  hash ← hashValue(key);                  // 追加するデータのハッシュ値
  node ← table[hash];                     // 着目する線形リストへの参照
  while (node が null でない)
    if( [ ア ] )                          // このキー値は登録済み
      return false;
    endif
    [ イ ] ;                              // 後続ノードに着目
  endwhile
  tempNode ← new Node( [ エ ] );          // ノードを生成
  [ オ ] ← tempNode;                      // ノードを追加
  return true;
endfunction
```

図 3　追加関数 addNode のプログラム

〔チェイン法の計算量〕

チェイン法の計算量を考える。

計算量が最悪になるのは，［ カ ］場合である。しかし，ハッシュ関数の作り方が悪くなければ，このようなことになる確率は小さく，実際上は無視できる。

チェイン法では，データの個数を n とし，表の大きさ（配列の長さ）を m とすると，線形リスト上の探索の際にアクセスするノードの数は，線形リストの長さの平均 n／m に比例する。m の選び方は任意なので，n に対して十分に大きくとっておけば，計算量が［ キ ］となる。この場合の計算量は 2 分探索木による $O(\log n)$より小さい。

設問1　衝突（collision）とはどのような現象か。“キー”と“ハッシュ関数”という単語を用いて，35字以内で述べよ。

設問2　〔探索関数 search〕について，(1)，(2)に答えよ。

(1)　図1の場合，キー値が 23 のデータを探索するために，ノードにアクセスする順序はどのようになるか。“key1→key2→…→23”のように，アクセスしたノードのキー値の順序で答えよ。

(2)　図2中の［　ア　］～［　ウ　］に入れる適切な字句を答えよ。

設問3　〔追加関数 addNode〕のプログラム，図3中の［　エ　］，［　オ　］に入れる適切な字句を答えよ。

設問4　〔チェイン法の計算量〕について，(1)，(2)に答えよ。

(1)　［　カ　］に入れる適切な字句を25字以内で答えよ。

(2)　［　キ　］に入れる計算量を O 記法で答えよ。