Programa E.3 *Pesquisa binária*

```
TipoIndice Binaria(TipoChave x, TipoTabela *T)
{ TipoIndice i, Esq, Dir;
  if (T->n == 0)
  return 0;
  else
  { Esq = 1;
    Dir = T->n;
    do
      { i = (Esq + Dir) / 2;
        if (x > T->Item[i].Chave)
        Esq = i + 1;
        else Dir = i - 1;
      } while (x != T->Item[i].Chave && Esq <= Dir);
    if (x == T->Item[i].Chave) return i; else return 0;
  }
}
```

Programa E.4 *Estrutura do dicionário para árvores sem balanceamento*

```
typedef long TipoChave;
typedef struct TipoRegistro {
  TipoChave Chave;
  /* outros componentes */
} TipoRegistro;
typedef struct TipoNo * TipoApontador;
typedef struct TipoNo {
  TipoRegistro Reg;
  TipoApontador Esq, Dir;
} TipoNo;
typedef TipoApontador TipoDicionario;
```

Programa E.5 *Procedimento para pesquisar na árvore*

```
void Pesquisa(TipoRegistro *x, TipoApontador *p)
{ if (*p == NULL)
  { printf("Erro: Registro nao esta presente na arvore\n");
    return;
  }
  if (x->Chave < (*p)->Reg.Chave)
  { Pesquisa(x, &(*p)->Esq);
    return;
  }
  if (x->Chave > (*p)->Reg.Chave)
  Pesquisa(x, &(*p)->Dir);
  else *x = (*p)->Reg;
}
```


Programa E.6 *Procedimento para inicializar*

```
void Inicializa(TipoApontador *Dicionario)
{ *Dicionario = NULL; }
```

Programa E.7 *Procedimento para inserir na árvore*

```
void Insere(TipoRegistro x, TipoApontador *p)
{ if (*p == NULL)
  { *p = (TipoApontador)malloc(sizeof(TipoNo));
    (*p)->Reg = x;
    (*p)->Esq = NULL;
    (*p)->Dir = NULL;
    return;
  }
  if (x.Chave < (*p)->Reg.Chave)
  { Insere(x, &(*p)->Esq);
    return;
  }
  if (x.Chave > (*p)->Reg.Chave)
  Insere(x, &(*p)->Dir);
  else printf("Erro : Registro ja existe na arvore\n");
}
```

Programa E.8 *Programa para criar árvore*

```
/*-- Entra aqui a definicao dos tipos do Programa E.4 --*/
/*-- Entram aqui os Programas E.6 e E.7               --*/
int main(int argc, char *argv[])
{ TipoDicionario Dicionario; TipoRegistro x;
  Inicializa(&Dicionario);
  scanf("%d%*[^\n]", &x.Chave);
  while(x.Chave > 0)
    { Insere(x,&Dicionario);
      scanf("%d%*[^\n]", &x.Chave);
    }
}
```

Programa E.9 *Procedimentos para retirar x da árvore*

```
void Antecessor(TipoApontador q, TipoApontador *r)
{ if ((*r)->Dir != NULL)
  { Antecessor(q, &(*r)->Dir);
    return;
  }
  q->Reg = (*r)->Reg;
  q = *r;
  *r = (*r)->Esq;
  free(q);
}
```