ASD—6

Abstrakcyjne typy danych (listy) c.d.

Użyteczną metodą rozwiązywania problemów programistycznych jest projektowanie zstępujące (ang. top-down design). Rozpoczynamy od precyzyjnego zdefiniowania problemu, który chcemy rozwiązać. Problem dzieli się na podproblemy. Podproblemy dzielimy następnie na jeszcze mniejsze podproblemy, aż otrzymane problemy staną się tak łatwe, że możemy je rozwiązać bezpośrednio. Proces dzielenia na podproblemy nazywamy stopniowym uszczegó"lowianiem. Do zalet projektowania zstępującego zaliczamy:

- Projektowanie zstępujące stanowi systematyczną metodę rozwiązywania problemów.
- Otrzymane rozwiązanie jest modularne. Poszczególne kroki mogą być kodowane, uruchamiane, modyfikowane i ulepszane niezależnie od pozostałych.
- Otrzymane rozwiązanie jest łatwiejsze do zrozumienia.
- Dobrze zaprojektowane fragmenty mogą być ponownie zastosowane w innych zadaniach.

Inną metodą projektowania programów jest metoda wstępująca. Przy projektowaniu wstępującym nie dzieli się problemu na kroki. Zamiast tego rozbudowuje się narzędzia (język) za pomocą których problem będzie można rozwiązać bezpośrednio.

1 Projektowanie do wielokrotnego użytku: oddzielna kompilacja

Jak dotychczas tworzyliśmy programy zapisywane w jednym pliku. Język C umożliwia grupowanie funkcji o podobnym przeznaczeniu w biblioteki, które mogą być kompilowane niezależnie i wykorzystywane w różnych programach. Program w języku C składa się z plików nagłówkowych (z rozszerzeniem .h) i źródłowych (z rozszerzeniem .c). Pakiet (biblioteka) funkcji składa się w dwóch plików: nagłówkowego zawierającego deklaracje i źródłowego zawierającego definicje funkcji. Poniżej przedstawiono plik nagłówkowy dla pakietu operacji na ułamkach.

```
#ifndef ULAMKI
#define ULAMKI

#include <stdio.h>

typedef struct{
  int licznik, mianownik;
} Ulamek;

int wczytajUlam(Ulamek *);
void wypiszUlam(Ulamek);
Ulamek dodajUlam(Ulamek, Ulamek);
Ulamek odejmijUlam(Ulamek, Ulamek);
Ulamek wymnozUlam(Ulamek, Ulamek);
Ulamek podzielUlam(Ulamek, Ulamek);
```

```
#endif ULAMKI
```

Plik nagłówkowy włącza plik stdio.h potrzebny w funkcjach wczytywania i wypisywania ulamków. Test `#ifndef` ma na celu zapewnienie, że deklaracje zawarte w tym pliku będą przetwarzane tylko raz, chociaż plik może być włączany do programu wielokrotnie. Plik źródłowy ulamki.c zawiera definicje funkcji zadeklarowanych w ulamki.h. Dodatkowo trzeba będzie w nim zdefiniować funkcję obliczającą największy wspólny dzielnik dwóch liczb całkowitych i inną skracającą ułamek.

```
#include "ulamki.h"

int NWD(int a, int b)
{
}

void skrocUlam(Ulamek *u)
{
}

int wczytajUlam(Ulamek *u)
{
  printf("Licznik = ");
  scanf("%d",&u->licznik);
  printf("Mianownik = ");
  scanf("%d",&u->mianownik);
  if (u->mianownik == 0)
    return 1;
  else{
```

```
    skrocUlam(u);
    return 0;
  }
}
void wypiszUlam(Ulamek u)
{
  printf("Licznik = %d; Mianownik = %d",
  u.licznik, u.mianownik);
}

Ulamek dodajUlam(Ulamek u1, Ulamek u2)
{
}
Ulamek odejmijUlam(Ulamek u1, Ulamek u2)
{
}
Ulamek wymnozUlam(Ulamek u1, Ulamek u2)
{
}
Ulamek podzielUlam(Ulamek u1, Ulamek u2)
{
}
```

Program głowny, testujący powyższe funkcje mógłby mieć postać.

```
#include "ulamki.h"

main()
{
  Ulamek u;
```

```
  wczytajUlam(&u);
  wypiszUlam(u);

  return 0;
}
```

Oto jeszcze jeden przykład modularyzacji programu. Na poprzednim wykładzie mówiliśmy o listach. Oto przykład wykorzystania list do tworzenia skorowidza tekstu:

Należy wczytać tekst ze standardowego wejścia, a nstępnie wypisać każde słowo występujące na wejściu wraz z informacją, ile razy się ono pojawiło.

Oto schemat programu:
ZbiórSłów = $\emptyset$*; Dopóki nie koniec danych wejściowych*
Wczytaj słowo.
Znajdź wczytane słowo w ZbiorzeSłów;
Jeśli słowo jest w zbiorze zwiększ licznik wystąpień o jeden.
Jeśli słowa nie ma w zbiorze, to dodaj je do zbioru i ustaw licznik na 1.

Do implementacji zbioru użyjemy list. W skład struktury na liście wchodzi przechowywana informacja (licznik wystąpień), indeks względem, którego lokalizujemy dane (słowo) i wskaźnik do następnego elementu listy.

```
/* listadef.h */
#ifndef LISTADEF
```

```
#define LISTADEF
#include <stdio.h>
#include <string.h>

typedef char Tklucz[21];
typedef int Tinfo;

#define porownajklucze(s1,s2) ((int)(strcmp((s1),(s2))))
#define skopiujklucz(s1, s2)  (strcpy((s1),(s2)))
#define skopiujinfo(i1, i2)   ((i1) = (i2))

#endif


/* lista.h */
#ifndef LISTA
#define LISTA
#include <stdio.h>
#include "listadef.h"

typedef struct ElListy{
  Tklucz klucz;
  Tinfo info;
  struct ElListy *nast;
} *PtrElListy;

PtrElListy pustaLista(void);
PtrElListy znajdzRec(PtrElListy gl,
                     Tklucz kl, PtrElListy *poprz);
```

```
wstawRec(PtrElListy *gl,
         PtrElListy poprz, Tklucz kl, Tinfo nowainfo);
void aktualRec(PtrElListy ptr, Tinfo nowainf);
void kazdyRec(PtrElListy gl,
              void (*fn)(Tklucz, Tinfo));

#endif


/* lista.c */
#include "lista.h"

/*
typedef struct ElListy{
  Tklucz klucz;
  Tinfo info;
  struct ElListy *nast;
} *PtrEllisty;
*/
PtrElListy pustaLista(void)
{
  return NULL;
}

PtrElListy znajdzRec(PtrElListy gl,
                     Tklucz kl, PtrElListy *poprz)
/* Lista o glowie gl jest
   uporzadkowana rosnaco wzgledem kluczy */
{
```

```
  PtrElListy p, q;

  p = NULL; q = gl;
  while ( (q != NULL)
           &&
           porownajklucze(kl,q->klucz) ){
    p = q;
    q = q->nast;
  }
  *poprz = p;
  return q;
}

wstawRec(PtrElListy *gl,
         PtrElListy poprz, Tklucz kl, Tinfo info)
{
  PtrElListy p;

  p = malloc(sizeof(struct ElListy));
  skopiujklucz(p->klucz,kl);
  skopiujinfo(p->info,info);
  if (poprz == NULL){
    p->nast = *gl;
    *gl = p;
  }
  else{
    p->nast = poprz->nast;
    poprz->nast = p;
  }
```

```
}



void aktualRec(PtrElListy ptr, Tinfo nowainfo)
{
  skopiujinfo(ptr->info,nowainfo);
}

void kazdyRec(PtrElListy gl,
              void (*fn)(Tklucz, Tinfo))
{
  while (gl != NULL){
    (*fn)(gl->klucz,gl->info);
    gl = gl->nast;
  }
}
```

```
#include "lista.h"

void wypisz(Tklucz kl, Tinfo inf)
{
  printf("%s %d\n",kl, inf);
}
```

```
main()
/* Program oblicza skorowidz slow w pliku
   wejsciowym. Jedno slowo jest zapisane
   w jednym wierszu. */
{
  int i;
  char ch;
  Tklucz slowo;
  PtrElListy skor, p, q;

  skor = pustaLista();
  i = 0;
  while ( (ch = getchar()) != '!')
   if ( ch != '\n' )
      slowo[i++] = ch;
    else{
      slowo[i] = '\0';
      q = znajdzRec(skor,slowo,&p);
      if (q != NULL)
        aktualRec(q,q->info+1);
      else
        wstawRec(&skor,p,slowo,1);
      i = 0;
    }
  kazdyRec(skor,&wypisz);
  return 0;
}
```