Czy procesor musi się grzać?

Np. dodawanie 2 liczb 1-bitowych.

Możliwych stanów początkowych: cztery

Możliwych stanów końcowych: dwa

to można opisać jako zmianę **entropii**

zmiana entropii = (stała Boltzmanna) razy

(ln 4 – ln 2)

Czy procesor musi się grzać?

przyrost entropii = przyrost ciepła / (temperatura w kelvinach)

(dla komputera pracującego w temp. pokojowej 20 stopni Celsjusza temperatura w kelwinach wynosi 293.15)

przyrost ciepła = 293.15 * 1.38e-23 * (ln4-ln2)

= 2.80e-21 joula

1 foton światła żołtego ma energię

3.32e-19 joula (długość fali 6.e-7 metra)

czyli około 100 takich dodawań to energia 1 fotonu

Czy procesor musi się grzać?

granica termodynamiczna dla 1 wata grzania

1wat/ 2.80e-21/ 32 = 1.1e19

dodawań na polu 32 bitowym

krótka dygresja nt. struktury
struct

jedyne operacje jakie można wykonać na strukturze to znalezienie jej adresu (za pomocą operatora **&**) i odwołanie się do jej elementów (z użyciem operatora **.** lub kombinacji operatorów ***** i **.** albo ich odpowiednika **->**

(w niektórych kompilatorach można przekazać strukturę jako argument funkcji - także w gcc z "Free Software Foundation")

&alfa; alfa.pole ; (*ptr_alfa).pole;

ptr -> pole ;

dygresja nt. struct - ciąg dalszy

do pól bitowych struktury nie można stosować operatora &

(natomiast jest poprawnym odwołanie przez operator & do elementu struktury który nie jest polem bitowym)

```
& ptr->pole1 ;  /*  ptr jest wskaźnikiem do
                    struktury */
```

operator przecinkowy

```
/*   używany w celu grupowania instrukcji    */
k=7,f=13.4, funkcja(z);

k= (i=2,  j=14,  i*j);
    /*    wartość powyższego wyrażenia wynosi  28   */
```

operator przecinkowy

```
#include <stdio.h>
#include <stdlib.h>
  int main()
  {       int i,k;


      for(i=0 , k=11; i<=10 || k<=13; i+=2 , ++k)
            printf(" i=%3d  k=%3d\n", i,k);
      exit(0);
  } /*  koniec  funkcji  main  */
```

operator przecinkowy

Wynik programu:

```
i=  0    k= 11
i=  2    k= 12
i=  4    k= 13
i=  6    k= 14
i=  8    k= 15
i= 10    k= 16
```

operator spoza ortodoksyjnego C
&&

```
/*  adres  etykiety ;   operator &&  */
#include <stdio.h>
#include <stdlib.h>
     int  main()
{float n=1;
  char c;
  void *fp;
n:  ;
  ++n;
  printf("\n n=%f\n",n);    scanf("%c",&c);
  fp= && n;    /* to załadowanie fp można umieścić wcześniej... */
  goto *fp;
} /* koniec funkcji main*/
```

operator spoza ortodoksyjnego C
&&

Jedną z możliwości użycia adresów etykiet jest zainicjalizowanie tablicy o klasie pamięci static; tablica ta będzie przechowywać te adresy jako ”adresy skoków”.

```
static void *array[] = { &&etyk1, &&etyk2, &&etyk3 };

goto *array[i];

  /* oczywiście nie będzie sprawdzane czy  array[i] zostało zainicjalizowane! */
```

1.wydaje się, że instrukcja **switch** jest wystarczająco bogatym w możliwości rozgałęźnikiem

2.skoki wykonywać można tylko w ramach danej funkcji

funkcje zagnieżdżone – tego nie ma w ANSI C

```
 int main()
{ float func(float w, float y);    float a,b,c;
   b=4; c=3;
   a=func(b,c);   printf("\n a=%f\n",a);    exit(0);
} /* koniec funkcji main*/
float func ( float  x1,  float  x2)
{ float ff(float f1)
      { return(f1*f1);
      } /* koniec funkcji ff,  lokalnej wewnatrz func  */
   return(ff(x1)+ff(x2));
} /*  koniec  funkcji    func            gcc –ansi  -pedantic  ”nazwa zbioru”
*/
```

(wskaźniki)

```
    main()
{  void funkcja5 ( float** );    float n,*fp;    fp =  & n;
 printf("\n poczatek main fp=%p   &fp=%p\n",  fp,  &fp);
 funkcja5( &fp );
 printf("\n koniec   main fp=%p   &fp=%p\n",  fp,  &fp);  exit(0);
} /* koniec funkcji main*/
    void funkcja5(float ** wsk)
{ printf("\n poczatek funkcja5 *wsk=%p wsk=%p &wsk=%p\n",
                  *wsk,  wsk,  &wsk);
  *wsk=NULL;
  printf("\n koniec funkcja5 *wsk=%p wsk=%p &wsk=%p\n",
  *wsk, wsk, &wsk); }            /*  koniec funkcji   funkcja5  */
```

wynik programu :

poczatek main fp=0xbfffe754 &fp=0xbfffe750

poczatek funkcja5 *wsk=0xbfffe754 wsk=0xbfffe750 &wsk=0xbfffe740

koniec funkcja5 *wsk=(nil) wsk=0xbfffe750 &wsk=0xbfffe740

koniec main fp=(nil) &fp=0xbfffe750

inny ***wynik programu :***

poczatek main fp=0xbffff154 &fp=0xbffff150

poczatek funkcja5 *wsk=0xbffff154 wsk=0xbffff150 &wsk=0xbffff140

koniec funkcja5 *wsk=(nil) wsk=0xbffff150 &wsk=0xbffff140

koniec main fp=(nil) &fp=0xbffff150

Wskaźniki
(wskaźnik do funkcji)

```
#include <stdio.h>
#include <stdlib.h>
   int main()
{   float func(float w, float y);
    float a, b=6, c=5;
    a=(*func) (b,c);     printf("\n a=%f\n",a);
    exit(0);
} /* koniec funkcji main*/
float func(float x1, float x2)
{  return  (  x1 * x2 );
}    /*   koniec  funkcji  func   */
```

Wskaźniki
(wskaźnik do funkcji)

/ na poprzednim slajdzie mogloby być */*

```
float(*fp) (float a, float b);    /*  wskaźnik do funkcji */

fp=func;    a = (*fp)(b, c);       /*  wywołanie funkcji  */
```

```
/*  lecz nie może się pojawić  func = fp  !!!  to błąd  */

/*  func choć jest wskaźnikiem, nie może być zmieniane  !!  */

/*  musi wskazywać na właśnie tą funkcję  func  !!   */
```

kwalifikator volatile

```
volatile int alfa;

int volatile alfa;


/* gdzie umieszczać  kwalifikator ?   */
```

kwalifikator volatile

Czy kompilator może być „za mądry” i błędnie uznać, że zawartość zmiennej nie mogła się zmienić?

Kwalifikator volatile zapobiega takim sytuacjom…

(gdy kompilator mógłby błędnie uznać, że np. Zmienna nie miała powodu się zmienić!)

kwalifikator volatile

```
#include <stdio.h>
#include <stdlib.h>
 int main()
{   struct nowa { int f;
                  int k;
                };
          ..........................................
     volatile struct nowa  reprezentant2;
     struct nowa volatile  reprezentant3;
/* kolejność ?    */
```

kwalifikator const

```
const  float  liczba_pi = 3.14;

........

........

liczba_pi = 3.142;   /* kompilator to zauważa, zgłasza
fatalny błąd  */
```

kwalifikator const

```
#include <stdio.h>
#include <stdlib.h>
    int main ( )
{ char buf[] = "lasso";
  char * war;
  const char * wsk="Rak";
  printf("\n  %s\n",wsk);
  war=(char *) wsk;
  wsk=(const char *) buf;
  printf("\n  %s  %s\n",wsk,war);
  exit(0);   }                  /*   koniec funkcji    main  */
```

kwalifikator const

wynik programu (nie będzie żadnego zastrzeżenia):

Rak

lasso Rak

kwalifikator const

```
#include <stdio.h>
#include <stdlib.h>
    int main ( )
{ char buf[] = "lasso";
  char * war;
  const char * wsk="Rak";
  printf("\n  %s\n",wsk);
  war=(char *) wsk;      wsk=(const char *) buf;
  *war='S';  /* dla kompilatora fatalny błąd  */
  printf("\n  %s  %s\n",wsk,war);
  exit(0);   }                  /*   koniec funkcji    main  */
```

kwalifikator const

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
    int main ()
{ char buf[] = "lasso";
   const char rak[]="Rak";  /* staly lancuch znakowy */
   char *wsk, *war;
   wsk= (char *) rak;
   /* rak[0]='S'; */   /* bez komentarza to blad kompilacji ! */
   printf("\n  %s\n",wsk);
   war=(char *) wsk;
     strcpy(war,"fa"); /*  string copy */
   printf("\n\n  %s %s %s\n",wsk, war, rak);     exit(0);
   /* udalo sie zmienic lancuch const char ! */
}  /*   koniec funkcji    main  */
```

kwalifikator const

W wyniku wykonania programu z poprzedniego slajdu:

Rak

fa fa fa

/* czyli udało się zmienić łańcuch */

kwalifikator const

```
#include <stdio.h>

#include <stdlib.h>

      int main ()

{  const float f=3.14;

   printf("\n %10.3f\n",f);

   f=3.142;  /*  blad kompilacji  ! */

   printf("\n %10.3f\n",f);   exit(0);

}      /*   koniec funkcji    main  */
```

kwalifikator const
(stały wskaźnik)

```
#include <stdio.h>
#include <stdlib.h>
     int main ()
{  char bufa[40]="Instytucja";       char bufb[40]="Adres";
   char * const wsk1=bufa;     /*   stały wskaźnik   */
   char * const wsk2=bufb;    /*    stały wskaźnik   */
        printf("\n%s     %s     %p  %p",wsk1,wsk2,wsk1,wsk2);
        strcpy(bufa,"Poczta");   strcpy(bufb,"ul.Witosa 14/2");
        printf("\n%s     %s     %p  %p\n",wsk1,wsk2,wsk1,wsk2);
        exit(0);
}       /*   koniec funkcji    main  */
```

kwalifikator const
(stały wskaźnik)

wynik programu :

Instytucja Adres 0xbffffa40 0xbffffa18

Poczta ul.Witosa 14/2 0xbffffa40 0xbffffa18

kwalifikator const
(stały wskaźnik)

```
wsk1+=2; wsk2+=2;

/*  zmiana wskaźnika stałego,  kompilator ostrzega   */

printf("\n%s     %s     %p  %p\n",wsk1,wsk2,wsk1,wsk2);
```

..... error: assignment of read-only variable `wsk1'

..... error: assignment of read-only variable `wsk2‘

(gdyby wsk1 i wsk2 nie były wskaźnikami stałymi to printf wypisałby

czta .Witosa 14/2 0xbffffa42 0xbffffa1a)

kwalifikator const

```
const int suma;    /* zmienna suma jest stałą */

const int * wsk;   /* stałą jest zmienna, na którą wskazuje
                      wskaźnik modyfikowalny wsk */

int const * wsk;   /* jak powyżej */


int * const wsk;  /* wsk jest stałym wskaźnikiem */


const int * const wsk;  /* wsk jest stałym wskaźnikiem,

                           może wskazywać na niemodyfikowalną
                           zmienną typu int */
```

kwalifikator const

zajmijmy się ostatnim przykładem z poprzedniej folii; czy można jakoś wykorzystać taki wskaźnik ?

kwalifikator const

```
#include <stdio.h>
#include <stdlib.h>
      int main ()
{      const int * const wsk = 0;
      int ** wsk2;
      int k=-7;  int n=-8;
      wsk2= (int **) &wsk;
      (*wsk2)=&k;
      printf("\n *wsk=%d\n", *wsk);
        /*  wsk=&n;  to byłby fatalny blad  */
        ( *( (int**)  (&wsk) ) ) = &n;   /* to jest OK  !!   */
      printf("\n *wsk=%d\n", *wsk);
      exit(0);
}       /*   koniec funkcji    main  */
```

kwalifikator const

(program z poprzedniej folii wypisze to co poniżej; udało się zmienić stały wskaźnik !)

*wsk=-7

*wsk=-8

kwalifikator const

```
/* inny przykład ominiecia kwalifikatora const   */
#include <stdlib.h>
#include <stdio.h>
    int main ()
{ const char buf[] = "masa";
  char * wsk;
   wsk=(char *)buf;
   (*wsk)='k';
  printf("\n %s\n",buf);   /*   wydrukuje    kasa   */
  exit(0);
}       /*   koniec funkcji    main  */
```

kwalifikator const

Używać **const** , czy może raczej **#define** ?

const ma składnię języka C (można zdefiniować np. stałą strukturę **struct**)

#define ma składnię preprocesora

Długości wskaźników

```
#include <stdio.h>

#include <stdlib.h>

struct st{char buf[100000];};          /* to tylko deklaracja!          */

      int  main()

{ printf("\n%d”  , sizeof(  char *)   );

  printf("\n%d”   ,  sizeof(  int *)   );

  printf("\n%d”   ,  sizeof(  double *)   );

  printf("\n%d”   ,  sizeof(  struct st *)   );

  printf("\n%d”  ,   sizeof(  const struct st *)   );

  printf("\n%d”   ,  sizeof(  struct st (*)(float,float) )   );

  printf("\n%d\n",  sizeof(  short *)   );

} /* koniec funkcji main*/
```

argumenty wiersza poleceń

funkcja main ma dwa argumenty

```
main(int argc, char * argv [ ] )
```

(tak sa zwyczajowo nazywane, choć nie są to nazwy obowiązujące)

argumenty wiersza poleceń

funkcja main ma dwa argumenty

```
main(int argc, char * * argv )
```

(tak sa zwyczajowo nazywane, choć nie są to nazwy obowiązujące)

argumenty wiersza poleceń

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
 int   main  (int argc,  char *argv[] )
   /*    albo  int main( int argc,   char  **argv)    */
{ int n;
  for(n=0;n<argc; ++n)
  {  printf("\n            n=%d  param= %s",n, argv[n]);
     printf("\n to samo... n=%d  param= %s",n, *(argv+n) );
     printf("\n");
  }     exit(0);
 }  /*    koniec     funkcji     main     */
```

argumenty wiersza poleceń

```
#include <stdio.h>

#include <stdlib.h>


int   main(int argc,  char  **argv)

{

  if(argc>=3 &&  strcmp("alfa", *(argv+2))==0)

  printf("\n parametr alfa\n");

  exit(0);

} /* koniec funkcji main*/


    a.out    abcdef   alfa   /* jeśli takie wywołanie, to ..... */
```

argumenty wiersza poleceń

a.out ”jedna spacja”

a.out ”apostrof”’

a.out ’inny_apostrof”’ ’

np. aby przesłać łańcuch 4-znakowy a” ’

a.out ’a” ”””

(pojedynczy apos., a, podwójny apos., spacja, pojedynczy apos., podwójny apos., pojedynczy apos., podwójny apos.)