Podstawy Programowania

1. **Przykładowy cykl programowania**
   1) Opracowanie projektu: specyfikacja wymagań, danych wejściowych i wyjściowych;
   2) Opracowanie algorytmów: schemat blokowy, opis słowny;
   3) Napisanie programu → implementacja algorytmów;
   4) Testowanie programu i usuwanie błędów;
   5) Przekazanie użytkownikowi;
   6) Pielęgnacja programu.

2. **Algorytm**
   - Algorytm, dokładny przepis podający sposób rozwiązania określonego zadania w skończonej liczbie kroków.
   - Algorytm zapisany przy pomocy języka programowania jest programem.
   - **Program to nie algorytm**
   - Wyróżnia się algorytmy numeryczne, operujące na liczbach (np. algorytm Euklidesa), i nienumeryczne, operujące na obiektach nie liczbowych (np. sortowanie dokumentów). Istnieje również podział algorytmów na sekwencyjne (kolejność czynności jest określona jednoznacznie) i niesekwencyjne (równoległe, współbieżne - następstwo między pewnymi operacjami nie jest określone).
   - Algorytmy charakteryzują się możliwością zapisywania ich w różnych językach i przez skończoną liczbę symboli, bez odwoływania się do analogii, a także faktyczną wykonalnością i możliwością wielokrotnej realizacji.

3. **Język programowania**

   Definicja: jest to zbiór zasad składni, instrukcji, dzięki którym powstaje kod źródłowy programu.

4. **Kod źródłowy**

   Kod źródłowy, program komputerowy napisany w języku programowania. Jest to postać programu, która jest zrozumiała dla programisty. Kod źródłowy jest przekształcany na kod maszynowy w procesie translacji programu.

5. **Kod maszynowy**

   Kod maszynowy, język rozumiany przez procesor. Program w kodzie maszynowym składa się z ciągu wartości binarnych, które oznaczają zarówno instrukcje jak i dane. Program, który jest napisany w pewnym języku programowania, musi zostać skompilowany, aby mógł być wykonywany przez komputer. **Postać kodu maszynowego zależy od architektury procesora, na który dany program jest przeznaczony.** Dlatego program musi zostać skompilowany na konkretnej maszynie, ewentualnie na systemie kompatybilnym z systemem docelowym.

6. Programowanie strukturalne

Programowanie strukturalne, rodzaj programowania, w którym program podzielony jest na niewielkie moduły - procedury, bądź funkcje.
Pascal, C, Modula-2.

7. Programowanie obiektowe

Programowanie obiektowe to rodzaj programowania, w którym dane i wykonywane na nich operacje są połączone. Ten formalny zabieg umożliwia szybsze pisanie większych programów, przez "składanie ich" ze wzajemnie powiązanych obiektów, które odpowiadają za daną funkcję programu (np. przygotowanie danych, wykonanie obliczeń, zaprezentowanie wyników).
C++, Java, Effel.

8. Język obiektowy

Język obiektowy to język, który umożliwia realizowanie paradygmatu obiektowego, tzn. programowanie z użyciem hierarchii klas.
C++, Java, Smalltalk, obiektowy Pascal, Beta, Theta, CLOS, Eiffel, Ada98, Python

9. Język wysokiego poziomu

To język programowania, zazwyczaj o budowie blokowej, spełniający wymagania programowania strukturalnego, programowania z użyciem obiektów lub nawet programowania obiektowego.
C i C++, Smalltalk, Java, Pascal i Lisp, SQL, Logo

10. Interpreter

Interpreter analizuje instrukcje programu źródłowego a przeanalizowane fragmenty (zazwyczaj jedna instrukcja) są wykonywane. Wykonanie powtórnie tego samego fragmentu wymaga powtórnej analizy. O błędach programista dowiaduje się dopiero w chwili wykonywania błędnej instrukcji.
Typowymi interpreterami są systemy programowania Basic, Java, Perl.

11. Kompilator

Kompilator, to program tłumaczący program w języku wysokiego poziomu, który tworzy programy wynikowe, uruchamialne dopiero po zakończeniu tłumaczenia (w odróżnieniu od interpretera). Niektóre kompilatory tłumaczą kod źródłowy na kod język asemblera a dopiero później odbywa się tłumaczenie przez asembler na kod wynikowy.

12. Kompilacja

Kompilacja (ang. compiling), termin określający proces tłumaczenia programu z postaci źródłowej na kod wynikowy. Obejmując fazy analizy leksykalnej, składniowej i generowania kodu, kompilacja nie musi przekształcać programu źródłowego w wykonywalny kod maszynowy komputera. Produktem kompilacji jest najczęściej kod z adresami względnymi, obliczonymi z osobna dla każdego skompilowanego modułu. Podczas łączenia modułów następuje ostateczne przeliczenie adresów względem wspólnej bazy i doprowadzenie przetłumaczonego programu do postaci ładowalnej.

13. Program binarny

Program binarny (ang. binary program), program przetłumaczony na język maszynowy, gotowy do ładowania i wykonania.

14. Deasembler

Deasembler, program narzędziowy o działaniu odwrotnym do asemblera, przetwarzający dwójkową reprezentację programu w kodzie maszynowym na rozkazy języka asemblera. Deasembler zazwyczaj wchodzi w skład pakietu uruchomieniowego.

15. Deasemblacja

Deasemblacja, w informatyce, utworzenie postaci kodu programu zapisanej w języku asemblera, z wersji binarnej (wykonywalnej) tego programu. Deasemblacja jest jedną z funkcji często oferowanych przez programy uruchomieniowe.

16. Dekompilacja

Dekompilacja, odtworzenie postaci źródłowej programu na podstawie jego kodu wynikowego. Dekompilacja nie odtworzy, komentarzy i układu tekstu źródłowego programu.

17. Program wynikowy

Jest to efekt pracy kompilatora, półprodukt tłumaczenia w postaci nadającej się do dalszego tłumaczenia przez asembler lub do konsolidacji. W systemie MS-DOS pliki programu wynikowego oznacza się konwencjonalnie godłem obj.

18. Konsolidacja

1) W procesie tłumaczenia przebieg, polegający na wykonaniu programu łączącego (łączenie modułów statyczne), prowadzący do utworzenia programu ładowalnego,
2) Dołączanie modułów bibliotecznych do wykonywanego programu (łączenie modułów dynamiczne).

19. Preprocesor

W systemach programowania C i C++ program wykonujący pierwszy przebieg analizowania pliku z kodem źródłowym. Preprocesor rozpoznaje i interpretuje zamieszczone w tekście programu dyrektywy, takie jak **#include, #define, #ifdef, #endif**, umożliwiające zagnieżdżanie plików na wejściu dla kompilatora, definiowanie napisów (m. in. stałych), kompilację warunkową itp. Tekst wytworzony przez preprocesor jest podawany na wejściu kompilatora;

20. System programowania

Oprogramowanie, w którego skład wchodzi kompilator lub interpretator języka programowania, system wykonawczy oraz biblioteki.

21. IDE

Jest to połączenie w jedną funkcjonalną całość programów używanych do wytwarzania oprogramowania: **edytora, kompilatora, konsolidatora**, zazwyczaj realizowane w formie **środowiska okienkowego**.

22. RAD

Programowanie RAD, Rapid Application Development (“błyskawiczne opracowywanie aplikacji”), termin określający możliwości pakietu oprogramowania działającego w graficznym środowisku okienkowym, służącego do zestawiania aplikacji, w szczególności – ich interfejsów, z modułów wybieranych z bibliotek udostępnianych w postaci drzew tematycznych.

Aplikacje wytworzone w środowiskach RAD są z reguły słabo przenośne i zajmują dużo pamięci, jednak możliwość automatycznego oprogramowania ich najżmudniejszych elementów (interfejs graficzny) oraz ułatwiony kontakt ze standardowymi bazami danych powoduje, że programowanie RAD staje się coraz popularniejsze.

23. API

Aplication Programming Interface, polecenia, dzięki którym użytkownik ma dostęp do funkcji systemu operacyjnego, takich jak na przykład tworzenie interfejsu graficznego. Za pomocą API programista może również otrzymywać informacje o systemie, a także kontaktować się z urządzeniami peryferyjnymi.

24. Operatory wskaźnikowe

Istnieją operatory wskaźnikowe:

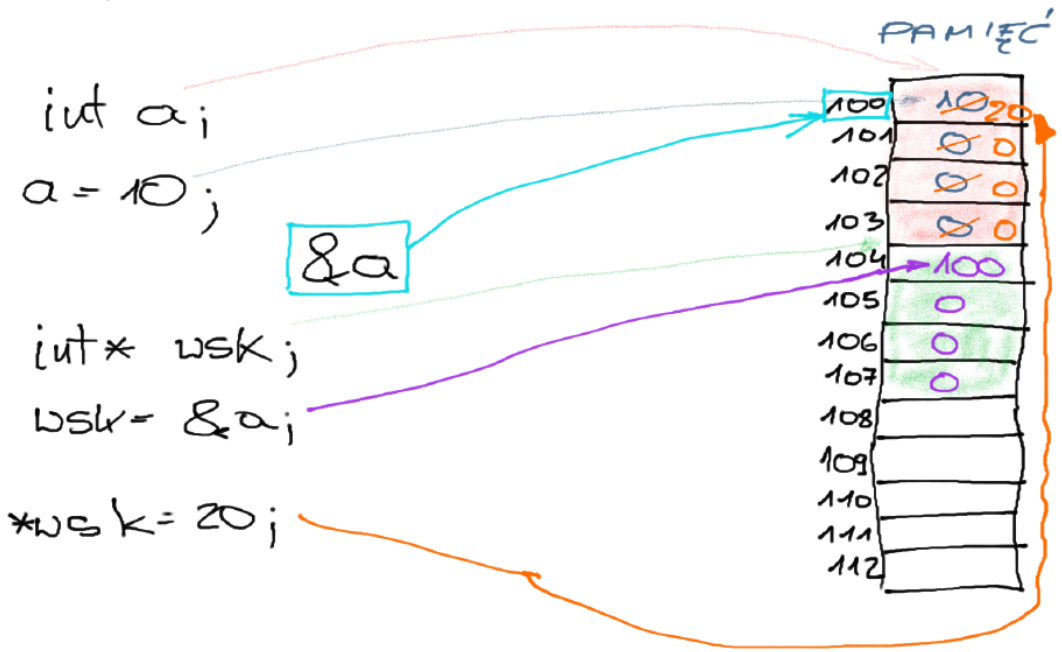
- **w = &z** przypisuje wskaźnikowi **w** adres zmiennej **z**.
- **z = *w** przypisuje zmiennej **z** wartość zmiennej na jaką wskazuje **w**.
- **w1 = w2** przypisuje wskaźnikowi **w1** adres tej samej zmiennej, na jaką wskazuje wskaźnik **w2**.
- **w++** oraz **wskaznik--** odpowiednio zwiększa lub zmniejsza adres zmiennej, na jaką wskazuje w o 1.
- **w+=n** oraz **wk-=n** odpowiednio zwiększa lub zmniejsza adres zmiennej, na jaką wskazuje **w** o n bajtów.
- **w1==w2, w1!=w2, w1<w2, w1>w2, w1<=w2, w1>=w2** odpowiednio porównuje adresy wskazywane przez poszczególne wskaźniki w1 i w2.

25. Typy danych

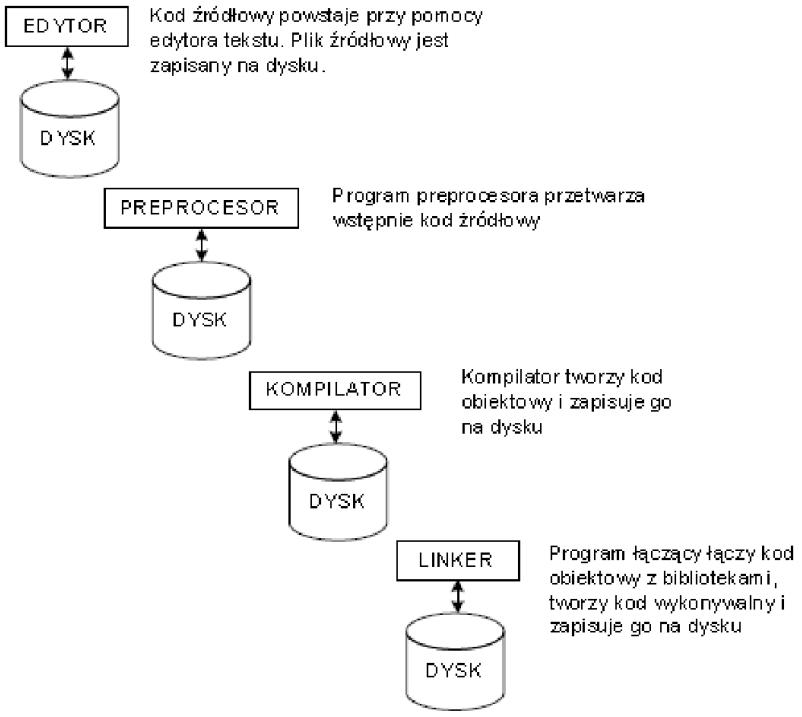
```
sizeof(bool) = 1
sizeof(char) = 1
sizeof(unsigned char) = 1
sizeof(wchar_t) = 2
sizeof(short) = 2
sizeof(unsigned short) = 2
sizeof(int) = 4
sizeof(unsigned int) = 4
sizeof(long) = 4
sizeof(unsigned long) = 4
sizeof(long long) = 8
sizeof(float) = 4
sizeof(double) = 8
sizeof(long double) = 12
```

+ wskaźnik 4 Bajty

26. Zapamiętać!



27. Fazy przetwarzania programu



28. Dyrektywy preprocesora

Odpowiadają za:

- Wstępne przetwarzanie programu,
- Dołączanie plików,

- Zamianę skrótów symbolicznych na ich definicje;
- Kod warunkowy.

29. Parametry otwarcia pliku

Tryb	Opis
"r"	Otwiera plik tekstowy do czytania
"w"	Otwiera plik tekstowy do zapisu. Jeżeli plik istniej, usuwa zawartość i umieszcza nowe dane. Jeżeli plik nie istnieje, zostaje utworzony.
"a"	Otwiera plik do zapisu. Jeżeli plik istnieje, dopisuje nowe dane na końcu istniejących danych. Jeżeli plik nie istnieje zostaje utworzony.
"r+"	Otwiera plik tekstowy do uaktualnienia, zezwala na zapis i czytanie
"w+"	Otwiera plik tekstowy do uaktualnienia, zezwala na zapis i czytanie. Jeżeli plik istnieje, usuwa zawartość. Jeżeli plik nie istnieje jest tworzony.
"a+"	Otwiera plik tekstowy do uaktualnienia, zezwala na zapis i czytanie. Jeżeli plik istnieje, dopisuje nowe dane na końcu. Jeżeli plik nie istnieje jest tworzony. Odczyt obejmuje cały plik, zapis polega na dodawaniu nowego tekstu.
"rb", "wb", "ab", "rb+", "r+b", "wb+", "w+b", "ab+", "a+b"	Wymienione specyfikatory mają takie samo znaczenie jak powyższe, dotyczą **plików binarnych**

30. Operacje na plikach

Wprowadzanie tekstu

```
FILE *f;
char zn;
f = fopen ("test35.txt","w");
printf("Wprowadz tekst :\n");
while ( (zn=getche()) != '\r')
putc(zn,f);
fclose(f);
return 0;
```

Kontrola otwarcia pliku

```
FILE *f;
f = fopen(”test35.txt”,”r”);
if ( f == NULL)
{
printf(“\n Nie moge otworzyc
pliku”);
exit(1);
}else{
//operacje na pliku
fclose(f);
}
```

Wprowadzanie łańcuchów

```
FILE *f;
f = fopen(”test35.txt”,”a”);
char napis[100];
while (strlen ( gets(napis)) > 0 )
{
fputs(napis,f);
fputs(”\n”,f);
}
fclose(f);
```

Czytanie łańcuchów z pliku

```
FILE *f;
f = fopen(”test35.txt”,”a”);
while (fgets ( napis, 80, f) !=
NULL )
printf("%s", napis);
fclose(f);
```

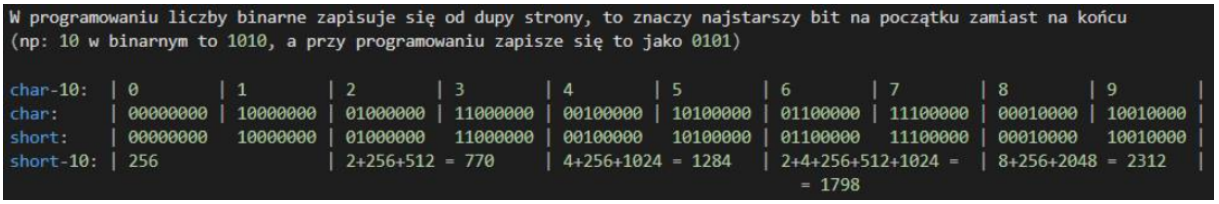
Formatowanie zapisu do pliku

```
FILE *f;
f = fopen(”test35.txt”,”a”);
int a; float b;
fprintf(f,"\n%d %f",a,b);
int num;
fscanf(f, "%d",&num);
fclose(f);
```

Zapis i odczyt w trybie binarnym

```
FILE *f;
f = fopen(”test35.txt”,”a”);
size_t fread(void *ptr,size_t
size,size_t n,FILE *stream);
fclose(f);
```

31. Zapamiętać mocno!



32. Malloc

```
int rozmiar;
cout << "Podaj rozmiar tablicy : ";
cin >> rozmiar;
int *wsk= (int*)malloc(rozmiar*sizeof(int));
if (wsk != NULL){
for (int i=0;i<rozmiar;i++) wsk[i]=i;
for (int i=0;i<rozmiar;i++)
cout << "\nelement [" << i <<"] = " << wsk[i];
free(wsk);
```

33. Metody programowania w C++

- **Proceduralne | Strukturalne**
  - Program traktowany jako seria instrukcji i procedur działających na danych,
  - Dane całkowicie odseparowane od procedur.
- **Zorientowane obiektowo.**
  - Stosowane do dużych programów,
  - Dostarcza technik zarządzania złożonymi elementami,
  - Łączy w logiczną całość dane i funkcje.

34. Konstruktor i Destruktor

- **Konstruktor** to specjalna funkcja składowa, która nazywa się tak samo jak klasa. W ciele konstruktora zamieszcza się instrukcje, które powodują ustawienie wartości początkowych przy tworzeniu obiektu danej klasy.
- **Destruktorem** klasy jest funkcja poprzedzona ~, której zadaniem jest „posprzątanie” w pamięci po obiekcie, który został zniszczony. Destruktor jest wywoływany automatycznie.